The 30 Year Horizon

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

Volume 5: Axiom Interpreter
Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan
Portions Copyright (c) 2007 Alfredo Portes
Portions Copyright (c) 2007 Arthur Ralfs
Portions Copyright (c) 2005 Timothy Daly

Portions Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of The Numerical ALgorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
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
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 Cawness  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 Coul
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 Duham  Martin Dunstan
Brian Dupee  Dominique Duval  Robert Edwards
Heow Eide-Goodman  Lars Ericksson  Richard Fateman
Bertfried Fauser  Stuart Feldman  John Fletcher
Brian Ford  Albrecht Fortenbacher  George Frances
Constantine Frangos  Timothy Freeman  Korinn Fu
Marc Gaetano  Rudiger Gebauer  Van de Geijn
Kathy Gerber  Patricia Gianni  Samantha Goldrich
Holger Gollan  Teresa Gomez-Diaz  Laureano Gonzalez-Vega
Stephen Gortler  Johannes Grabmeier  Matt Grayson
Klaus Ebbe Grue  James Griesmer  Vladimir Grinberg
Oswald Gschnitzer  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 Kai Kaminski
Grant Keady Wilfrid Kendall Tony Kennedy
Ted Kosan Paul Kosinski Klaus Kusche
Bernhard Kutzler Tim Lahey Larry Lambe
Kaj Laurson George L. Legendre Franz Lehner
Frederic Lehobey Michel Levaud Howard Levy
Ren-Cang Li Rudiger Loos Michael Lucks
Richard Luczak Cammi Maguire Francois Maltey
Alasdair McAndrew Bob McElrath Michael McGettrick
Edi Meier Ian Meikle David Mentre
Victor S. Miller Gerard Milmeister Mohammed Mobarak
H. Michael Moeller Michael Monagan Marc Moreno-Maza
Scott Morrison Joel Moses Mark Murray
William Naylor Patrice Naudin C. Andrew Neff
John Nelder Godfrey Nolan Arthur Norman
Jinzhong Niu Michael O’Connor Summat Oemrawsingh
Kostas Oikonomou Humberto Ortiz-Zuazaga Julian A. Padget
Bill Page David Parnas Susan Pelzel
Michel Petitot Didier Pinchon Ayal Pinkus
Frederick H. Pitts Jose Alfredo Portes Gregorio Quintana-Orti
Claude Quitte Arthur C. Ralfs Norman Ramsey
Anatoly Raportirenko Albert D. Rich Michael Richardson
Guilherme Reis Huan Ren Renaud Ribo
Jean Rivlin Nicolas Robidoux Simon Robinson
Raymond Rogers Michael Rothstein Martin Rubey
Philip Santas Alfred Scheerhorn William Schelter
Gerhard Schneider Martin Schoener Marshall Schor
Frithjof Schulze Fritz Schwarz Steven Segletes
V. Sina Nick Simicich William Sit
Elena Smirnova Jonathan Steinbach Fabio Stumbo
Christine Sundaresan Robert Sutor Moss E. Sweedler
Eugene Surowitz Max Tegmark T. Doug Telford
James Thatcher Balbir Thomas Mike Thomas
Dylan Thurston Steve Toleque Barry Trager
Themos T. Tsikas Gregory Vanuxem Bernhard Wall
Stephen Watt Jaap Weel Juergen Weiss
M. Weller Mark Wegman James Wen
Thorsten Werther Michael Wester R. Clint Whaley
John M. Wiley Berhard Will Clifton J. Williamson
Stephen Wilson Shmuel Winograd Robert Wisbauer
Sandra Wityak Waldemar Wiwianka Knut Wolf
Liu Xiaojun Clifford Yapp David Yun
Vadim Zhytnikov Richard Zippel Evelyn Zoernack
Bruno Zuercher Dan Zwilinger
Contents

1 The Interpreter ........................................... 1

2 The Fundamental Data Structures ......................... 3
  2.1 The global variables ................................... 3
    Credits ................................................. 3
    defvar $creditlist .................................... 3
    defvar $current-directory ............................ 3
    defvar $defaultMsgDatabaseName .................... 5
    defvar $directory-list ................................ 6
    defvar $initialModemapFrame ...................... 6
    defvar $library-directory-list .................. 7
    defvar $msgDatabaseName ......................... 7
    defvar $openServerIfTrue ......................... 8
    defvar $relative-directory-list ............... 8
    defvar $relative-library-directory-list ...... 9
    defvar $spadroot ...................................... 9
    defvar $SpadServer .................................. 10
    defvar $SpadServerName ........................... 10
    defvar $IOindex ...................................... 10

3 Starting Axiom ............................................ 13
  3.1 Variables Used ....................................... 13
  3.2 Data Structures .................................... 13
  3.3 Functions ........................................... 13
    Set the restart hook ................................ 13
    restart function (The restart function) .......... 14
    defun Non-interactive restarts .................. 16
    defun The startup banner messages .............. 17
    defun Make a vector of filler characters ....... 18
    Starts the interpreter but do not read in profiles .. 18
    defvar $quitTag ....................................... 18
    defun runspad ....................................... 18
    defun Reset the stack limits .................... 19
4 Handling Terminal Input

4.1 Streams

defvar $curinstream

defvar $curoutstream

defvar $errorinstream

defvar $erroroutstream

defvar $*eof*

defvar $*whitespace*

defvar $InteractiveMode

defvar $boot

Top-level read-parse-eval-print loop

defun ncIntLoop

defvar $intTopLevel

defvar $intRestart

defun intloop

defvar $ncMsgList

defun SpadInterpretStream

defvar $promptMsg

defun GCL cmpnote function

defvar $newcompErrorCount

defvar $nopos

4.2 The Read-Eval-Print Loop

defun intloopReadConsole

4.3 Helper Functions

Get the value of an environment variable

defvar $intCoerceFailure

defvar $intSpadReader

defun InterpExecuteSpadSystemCommand

defun ExecuteInterpSystemCommand

defun Handle Synonyms

defun Synonym File Reader

defun init-memory-config

Set spadroot to be the AXIOM shell variable

Does the string start with this prefix?

defun Interpret a line of lisp code

Get the current directory

Prepend the absolute path to a filename

Make the initial modemap frame

defun ncloopEscaped

defun intloopProcessString

defun ncloopParse

defun next

defun next1

defun incString

Call the garbage collector

defun reroot
Contents

- defun setCurrentLine ........................................ 40
- Show the Axiom prompt ...................................... 40
- defvar $frameAlist ........................................... 41
- defvar $frameNumber ........................................ 41
- defvar $currentFrameNum .................................... 41
- defvar $EndServerSession ................................... 42
- defvar $NeedToSignalSessionManager ....................... 42
- defvar $sockBufferLength ................................... 42
- READ-LINE in an Axiom server system ...................... 42
- defun protectedEVAL .......................................... 45
- defvar $QuietCommand ....................................... 45
- defun executeQuietCommand .................................. 45
- defun parseAndInterpret ..................................... 46
- defun parseFromString ....................................... 46
- defvar $interpOnly .......................................... 47
- defvar $minivectorNames ..................................... 47
- defvar $domPvar ............................................. 47
- defun processInteractive .................................... 48
- defvar $ProcessInteractiveValue ............................ 50
- defvar $HTCompanionWindowID ................................ 50
- defun processInteractive1 ................................... 50
- defun interpretTopLevel ..................................... 51
- defvar $genValue ............................................ 51
- defun Type analyzes and evaluates expression x, returns object 52
- defun Dispatcher for the type analysis routines ........... 52
- defun interpret2 ............................................. 53
- defun Result Output Printing ................................ 54
- defun printStatisticsSummary ................................ 56
- defun printStorage .......................................... 56
- defun printTypeAndTime ...................................... 56
- defun printTypeAndTimeNormal ................................ 57
- defun printTypeAndTimeSaturn ................................. 58
- defun printAsTeX ............................................. 59
- defun sameUnionBranch ...................................... 59
- defun msgText ................................................. 60
- defun Right-justify the Type output .......................... 60
- defun Destructively fix quotes in strings .................... 61
- Include a file into the stream ................................ 61
- defun intloopInclude0 ...................................... 61
- defun intloopProcess ........................................ 62
- defun intloopSpadProcess .................................... 63
- defun intloopSpadProcess,interp ............................. 64
- defun phParse ................................................ 64
- defun phIntReportMsgs ....................................... 64
- defun phInterpret ............................................ 65
- defun intInterpretPform ...................................... 65
defun zeroOneTran ............................................ 66
defun ncConversationPhase .................................. 66
defun ncConversationPhase,wrapup ......................... 66
defun ncError .................................................. 67
defun intloopEchoParse ....................................... 67
defun ncloopPrintLines ....................................... 68
defun mkLineList .............................................. 68
defun nonBlank ............................................... 69
defun ncloopDQlines ......................................... 70
defun poGlobalLinePosn ...................................... 70
defun streamChop .............................................. 70
defun ncloopInclude0 ......................................... 71
defun incStream .............................................. 71
defun incRenumber ........................................... 72
defun incZip .................................................. 72
defun incZip1 ................................................ 72
defun inclgen .................................................. 73
defun inclgen1 ................................................ 73
defun incRenumberLine ....................................... 73
defun incRenumberItem ....................................... 74
defun incHandleMessage ..................................... 74
defun include ................................................ 75
defmacro Rest ............................................... 75
defvar $Top ................................................. 75
defvar $IfSkipToEnd ....................................... 75
defvar $IfKeepPart ......................................... 76
defvar $IfSkipPart ......................................... 76
defvar $ElseifSkipToEnd ................................... 76
defvar $ElseifKeepPart .................................... 76
defvar $ElseifSkipPart .................................... 76
defvar $ElseSkipToEnd ...................................... 77
defvar $ElseKeepPart ....................................... 77
defvar $ElseTop ............................................. 77
defvar $ElseTop? .......................................... 77
defvar $If? ............................................... 77
defvar $Elseif? ........................................... 78
defvar $Else ............................................... 78
defvar $SkipEnd ............................................ 78
defvar $KeepPart .......................................... 78
defvar $SkipPart .......................................... 78
defvar $Skipping ........................................... 79
defun include1 .............................................. 79
defun xlPrematureEOF ....................................... 84
defun xlMsg .................................................. 84
defun xlOK ................................................... 84
defun xlOK1 ................................................ 85
defun incAppend .............................................. 85
defun incAppend1 ............................................. 85
defun incLine .............................................. 86
defun incLine1 .............................................. 86
defun incmsgPrematureEOF .................................. 86
defun theorigin .............................................. 86
defun porigin ................................................ 87
defun ifCond ................................................ 87
defun xlSkip ............................................... 87
defun xlSay .................................................. 88
defun inclmsgSay ........................................... 88
defun theid .................................................. 88
defun xlnoSuchFile ......................................... 89
defun inclmsgNoSuchFile ..................................... 89
defun thefname .............................................. 89
defun pfname ................................................. 89
defun xlCannotRead ......................................... 90
defun inclmsgCannotRead .................................... 90
defun xlfileCycle .......................................... 90
defun inclmsgFileCycle ...................................... 90
defun xlConActive .......................................... 91
defun inclmsgConActive ..................................... 92
defun xlConStill ............................................ 92
defun inclmsgConStill ....................................... 92
defun xlConsole .............................................. 92
defun inclmsgConsole ........................................ 93
defun xsKippingFin ......................................... 93
defun inclmsgFinSkipped .................................... 93
defun xlPrematureFin ....................................... 93
defun inclmsgPrematureFin .................................. 94
defun assertCond ............................................ 94
defun xlfSyntax ............................................. 94
defun inclmsgIfSyntax ...................................... 95
defun xlfBug ............................................... 95
defun inclmsgIfBug ........................................... 96
defun xlcmdBug .............................................. 96
defun inclmsgCmdBug ......................................... 96
defun SincComands .......................................... 96
defun $pfMacros ............................................ 97
defun incClassify ........................................... 97
defun incCommand? .......................................... 98
defun incPrefix? ............................................ 99
defun incCommandTail ....................................... 99
defun incDrop ............................................... 100
defun inclFname ............................................ 100
defun incFileInput .......................................... 100
defun incConsoleInput ....................................... 100
5 The Token Scanner

defun incNConsoles ........................................................................ 101
defun incActive? ........................................................................ 101
defun incRgen ............................................................................ 101
defun Delay ................................................................................ 102
defvar $StreamNil .................................................................. 102
defun incRgen1 ........................................................................ 102

defvar $SPACE ........................................................................ 105
defvar $ESCAPE ...................................................................... 105
defvar $STRINGCHAR .............................................................. 105
defvar $PLUSCOMMENT ............................................................... 106
defvar $MINUSCOMMENT ............................................................. 106
defvar $RADIXCHAR ................................................................. 106
defvar $DOT ............................................................................... 106
defvar $EXPONENT1 .................................................................. 107
defvar $EXPONENT2 .................................................................. 107
defvar $CLOSEPAREN ................................................................. 107
defvar $QUESTION ................................................................... 107
defvar $scanKeyWords .................................................................. 108
defvar $infgeneric ...................................................................... 110
defun lineoftoks ........................................................................ 111
defun nextline ............................................................................. 112
defun scanIgnoreLine ................................................................. 113
defun constoken ......................................................................... 113
defun scanToken ........................................................................ 114
defun lid ..................................................................................... 115
defun startsComment? ................................................................. 115
defun scanComment ..................................................................... 116
defun lfcomment ......................................................................... 116
defun startsNegComment? ............................................................ 117
defun scanNegComment ............................................................... 117
defun lfnegcomment .................................................................... 117
defun punctuation? ..................................................................... 118
defun lfkey .................................................................................. 118
defun spleI ................................................................................ 118
defun spleI1 ............................................................................... 119
defun scanEsc ............................................................................. 119
defun subMatch ........................................................................... 119
defun substringMatch ................................................................. 119
defun scanKeyTr ........................................................................ 120
defun keyword? ......................................................................... 121
defun keyword .......................................................................... 121
defun scanPossFloat ................................................................. 121
defun digit .................................................................................. 122
defun lfkey ................................................................................ 122
defun spleI ................................................................................. 122
defun spleI1 .............................................................................. 122
defun scanEsc ............................................................................ 123
### 6 Input Stream Parser

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun Input Stream Parser</td>
<td>141</td>
</tr>
<tr>
<td>defun npItem</td>
<td>142</td>
</tr>
<tr>
<td>defun npItem1</td>
<td>142</td>
</tr>
<tr>
<td>defun npFirstTok</td>
<td>143</td>
</tr>
<tr>
<td>defun Push one item onto $stack</td>
<td>143</td>
</tr>
<tr>
<td>defun Pop one item off $stack</td>
<td>144</td>
</tr>
<tr>
<td>defun Pop the second item off $stack</td>
<td>144</td>
</tr>
<tr>
<td>defun Pop the third item off $stack</td>
<td>144</td>
</tr>
<tr>
<td>defun npQualDef</td>
<td>145</td>
</tr>
<tr>
<td>defun Advance over a keyword</td>
<td>145</td>
</tr>
<tr>
<td>defun Advance the input stream</td>
<td>145</td>
</tr>
<tr>
<td>defun npComma</td>
<td>146</td>
</tr>
<tr>
<td>defun npTuple</td>
<td>146</td>
</tr>
<tr>
<td>defun npCommaBackSet</td>
<td>146</td>
</tr>
<tr>
<td>defun npQualifiedDefinition</td>
<td>147</td>
</tr>
</tbody>
</table>
defun npQualified ............................................. 147
defun npDefinitionOrStatement .............................. 147
defun npBackTrack ........................................... 148
defun npGives ................................................ 148
defun npLambda ................................................. 148
defun npType .................................................. 149
defun npMatch ................................................. 150
defun npSuch .................................................. 150
defun npWith .................................................. 150
defun npCompMissing ......................................... 151
defun npMissing .............................................. 151
defun npRestore .............................................. 152
defun Peek for keyword s, no advance of token stream ... 152
defun npCategoryL ............................................. 152
defun npCategory .............................................. 153
defun npSCategory ............................................. 153
defun npSignature ............................................ 154
defun npSigItemlist .......................................... 154
defun npListing ................................................ 155
defun Always produces a list, fn is applied to it ......... 155
defun npSigItem ............................................... 156
defun npTypeVariable ......................................... 156
defun npSignatureDefinee ..................................... 156
defun npTypeVariablelist .................................... 157
defun npSigDecl ............................................... 157
defun npPrimary .............................................. 158
defun npPrimary2 ............................................. 158
defun npADD ................................................... 159
defun npADD ................................................... 159
defun npAtom2 ............................................... 159
defun npInfixOperator ....................................... 160
defun npInfixOp .............................................. 161
defun npPrefixColon ......................................... 161
defun npApplication ......................................... 162
defun npDotted ............................................... 162
defun npAnyNo ................................................. 162
defun npSelector ............................................. 163
defun npApplication2 ........................................ 163
defun npPrimary1 ............................................ 164
defun npMacro ................................................. 164
defun npMdef .................................................. 164
defun npMDEF .................................................. 165
defun npMDEFinition ......................................... 165
defun npFix .................................................... 166
defun npLet .................................................... 166
defun npLetQualified ........................................ 166
defun npAngleBared ........................................... 186
defun npDefn .................................................. 187
defun npDef .................................................. 187
defun npBPileDefinition ..................................... 188
defun npPileBracketed .................................... 188
defun npPileDefinitionlist ................................ 189
defun npListAndRecover ................................... 189
defun npMoveTo .............................................. 190
defun syIgnoredFromTo ..................................... 191
defun syGeneralErrorHere .................................. 192
defun sySpecificErrorHere ................................ 192
defun sySpecificErrorAtToken ........................... 192
defun npDefinitionlist ..................................... 193
defun npSemiListing ....................................... 193
defun npSemiBackSet ....................................... 193
defun npRule .................................................. 193
defun npSingleRule ........................................ 194
defun npDefTail ............................................. 194
defun npDefaultValue ...................................... 194
defun npWConditional ...................................... 195
defun npConditional ....................................... 195
defun npElse .................................................. 196
defun npBacksetElse ...................................... 197
defun npLogical .............................................. 197
defun npDisjand ............................................. 197
defun npDiscrim ............................................. 197
defun npQuiver .............................................. 198
defun npRelation ........................................... 198
defun npSynthetic ......................................... 198
defun npBy .................................................... 199
defun .......................................................... 199
defun npSegment ............................................. 200
defun npArith ............................................... 200
defun npSum ................................................ 201
defun npTerm ................................................ 201
defun npRemainder ........................................ 201
defun npProduct ............................................ 202
defun npPower ............................................... 202
defun npAmpersandFrom .................................... 202
defun npFromdom ............................................ 202
defun npFromdom1 .......................................... 203
defun npAmpersand ......................................... 204
defun npName ................................................. 204
defvar $npTokToNames .................................... 204
defun npId ..................................................... 204
7 Pftrees

7.1 Abstract Syntax Trees Overview

7.2 Structure handlers

defun pfGlobalLinePosn

defun pfCharPosn

defun pfLinePosn

defun pfFileName

defun pfCopyWithPos

defun pfMapParts

defun pf0ApplicationArgs

defun pf0FlattenSyntacticTuple

defun pfSourcePosition

defun Convert a Sequence node to a list

defun pfSpread

defun Deconstruct nodes to lists

defun pfCheckMacroOut

defun pfCheckArg

defun pfCheckId

defun pfFlattenApp

defun pfCollect1?

defun pfCollectVariable1

defun pfPushMacroBody

defun pfSourceStok

defun pfTransformArg

defun pfTaggedToTyped1

defun pfSuch

7.3 Special Nodes

defun Create a Listof node

defun pfNothing

defun Is this a Nothing node?
7.4 Leaves

- defun Create a Document node ........................................... 246
- defun Construct an Id node ................................................. 246
- defun Is this an Id node? .................................................. 246
- defun Construct an Id leaf node .......................................... 246
- defun Return the Id part .................................................... 247
- defun Construct a Leaf node ................................................. 247
- defun Is this a leaf node? .................................................. 247
- defun Return the token position of a leaf node ...................... 248
- defun Return the Leaf Token ............................................... 248
- defun Is this a Literal node? .............................................. 248
- defun Create a LiteralClass node ....................................... 248
- defun Return the LiteralString ............................................ 249
- defun Return the parts of a tree node ................................ 249
- defun Return the argument unchanged ................................ 249
- defun pfPushBody ................................................................ 249
- defun An S-expression which people can read. ...................... 250
- defun Create a human readable S-expression .......................... 250
- defun Construct a Symbol or Expression node ....................... 251
- defun Construct a Symbol leaf node ..................................... 251
- defun Is this a Symbol node? .............................................. 252
- defun Return the Symbol part .............................................. 252

7.5 Trees

- defun Construct a tree node ................................................. 252
- defun Construct an Add node ................................................. 252
- defun Construct an And node ................................................. 253
- defun pfAttribute ................................................................ 253
- defun Return an Application node ........................................ 253
- defun Return the Arg part of an Application node ................. 254
- defun Return the Op part of an Application node ................... 254
- defun Is this an And node? .................................................. 254
- defun Return the Left part of an And node ............................ 254
- defun Return the Right part of an And node ......................... 255
- defun Flatten a list of lists ................................................ 255
- defun Is this an Application node? ...................................... 255
- defun Create an Assign node ................................................. 255
- defun Is this an Assign node? .............................................. 255
- defun Return the parts of an LhsItem of an Assign node .......... 256
- defun Return the LhsItem of an Assign node ........................ 256
- defun Return the RHS of an Assign node .............................. 256
- defun Construct an application node for a brace ................... 257
- defun Construct an Application node for brace-bars .............. 257
- defun Construct an Application node for a bracket ................ 257
- defun Construct an Application node for bracket-bars ............ 257
- defun Create a Break node .................................................. 258
- defun Is this a Break node? ................................................ 258
<table>
<thead>
<tr>
<th>Function Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the From part of a Break node</td>
<td>258</td>
</tr>
<tr>
<td>Construct a CoerceTo node</td>
<td>259</td>
</tr>
<tr>
<td>Is this a CoerceTo node?</td>
<td>259</td>
</tr>
<tr>
<td>Return the Expression part of a CoerceTo node</td>
<td>259</td>
</tr>
<tr>
<td>Return the Type part of a CoerceTo node</td>
<td>259</td>
</tr>
<tr>
<td>Return the Body of a Collect node</td>
<td>260</td>
</tr>
<tr>
<td>Return the Iterators of a Collect node</td>
<td>260</td>
</tr>
<tr>
<td>Create a Collect node</td>
<td>260</td>
</tr>
<tr>
<td>Is this a Collect node?</td>
<td>260</td>
</tr>
<tr>
<td>pfDefinition</td>
<td>261</td>
</tr>
<tr>
<td>Return the Lhs of a Definition node</td>
<td>261</td>
</tr>
<tr>
<td>Return the Rhs of a Definition node</td>
<td>261</td>
</tr>
<tr>
<td>Is this a Definition node?</td>
<td>261</td>
</tr>
<tr>
<td>Return the parts of a Definition node</td>
<td>262</td>
</tr>
<tr>
<td>Create a Do node</td>
<td>262</td>
</tr>
<tr>
<td>Is this a Do node?</td>
<td>262</td>
</tr>
<tr>
<td>Return the Body of a Do node</td>
<td>262</td>
</tr>
<tr>
<td>Construct a Sequence node</td>
<td>263</td>
</tr>
<tr>
<td>Construct an Exit node</td>
<td>263</td>
</tr>
<tr>
<td>Is this an Exit node?</td>
<td>263</td>
</tr>
<tr>
<td>Return the Cond part of an Exit</td>
<td>263</td>
</tr>
<tr>
<td>Return the Expression part of an Exit</td>
<td>264</td>
</tr>
<tr>
<td>Create an Export node</td>
<td>264</td>
</tr>
<tr>
<td>Construct an Expression leaf node</td>
<td>264</td>
</tr>
<tr>
<td>pfFirst</td>
<td>264</td>
</tr>
<tr>
<td>Create an Application Fix node</td>
<td>265</td>
</tr>
<tr>
<td>Create a Free node</td>
<td>265</td>
</tr>
<tr>
<td>Is this a Free node?</td>
<td>265</td>
</tr>
<tr>
<td>Return the parts of the Items of a Free node</td>
<td>266</td>
</tr>
<tr>
<td>Return the Items of a Free node</td>
<td>266</td>
</tr>
<tr>
<td>Construct a ForIn node</td>
<td>266</td>
</tr>
<tr>
<td>Is this a ForIn node?</td>
<td>266</td>
</tr>
<tr>
<td>Return all the parts of the LHS of a ForIn node</td>
<td>267</td>
</tr>
<tr>
<td>Return the LHS part of a ForIn node</td>
<td>267</td>
</tr>
<tr>
<td>Return the Whole part of a ForIn node</td>
<td>267</td>
</tr>
<tr>
<td>pfFromDom</td>
<td>267</td>
</tr>
<tr>
<td>Construct a Fromdom node</td>
<td>267</td>
</tr>
<tr>
<td>Is this a Fromdom node</td>
<td>268</td>
</tr>
<tr>
<td>Return the What part of a Fromdom node</td>
<td>268</td>
</tr>
<tr>
<td>Return the Domain part of a Fromdom node</td>
<td>269</td>
</tr>
<tr>
<td>pfConstruct a Hide node</td>
<td>269</td>
</tr>
<tr>
<td>Return the Cond part of an If</td>
<td>269</td>
</tr>
<tr>
<td>Return the Then part of an If</td>
<td>270</td>
</tr>
<tr>
<td>pfIfThenOnly</td>
<td>270</td>
</tr>
<tr>
<td>defun Return the Else part of an If</td>
<td>270</td>
</tr>
<tr>
<td>defun Construct an Import node</td>
<td>271</td>
</tr>
<tr>
<td>defun Construct an Iterate node</td>
<td>271</td>
</tr>
<tr>
<td>defun Is this an Iterate node?</td>
<td>271</td>
</tr>
<tr>
<td>defun Handle an infix application</td>
<td>271</td>
</tr>
<tr>
<td>defun Create an Inline node</td>
<td>272</td>
</tr>
<tr>
<td>defun pfLam</td>
<td>272</td>
</tr>
<tr>
<td>defun pfLambda</td>
<td>273</td>
</tr>
<tr>
<td>defun Return the Body part of a Lambda node</td>
<td>273</td>
</tr>
<tr>
<td>defun Return the Rets part of a Lambda node</td>
<td>273</td>
</tr>
<tr>
<td>defun Is this a Lambda node?</td>
<td>273</td>
</tr>
<tr>
<td>defun Return the Args part of a Lambda node</td>
<td>274</td>
</tr>
<tr>
<td>defun Return the Args of a Lambda node</td>
<td>274</td>
</tr>
<tr>
<td>defun Construct a Local node</td>
<td>274</td>
</tr>
<tr>
<td>defun Is this a Local node?</td>
<td>274</td>
</tr>
<tr>
<td>defun Return the parts of Items of a Local node</td>
<td>275</td>
</tr>
<tr>
<td>defun Return the Items of a Local node</td>
<td>275</td>
</tr>
<tr>
<td>defun Construct a Loop node</td>
<td>275</td>
</tr>
<tr>
<td>defun pfLoop1</td>
<td>275</td>
</tr>
<tr>
<td>defun Is this a Loop node?</td>
<td>276</td>
</tr>
<tr>
<td>defun Return the Iterators of a Loop node</td>
<td>276</td>
</tr>
<tr>
<td>defun p0LoopIterators</td>
<td>276</td>
</tr>
<tr>
<td>defun pfLp</td>
<td>276</td>
</tr>
<tr>
<td>defun Create a Macro node</td>
<td>277</td>
</tr>
<tr>
<td>defun Is this a Macro node?</td>
<td>277</td>
</tr>
<tr>
<td>defun Return the Lhs of a Macro node</td>
<td>277</td>
</tr>
<tr>
<td>defun Return the Rhs of a Macro node</td>
<td>277</td>
</tr>
<tr>
<td>defun Construct an MLambda node</td>
<td>278</td>
</tr>
<tr>
<td>defun Is this an MLambda node?</td>
<td>278</td>
</tr>
<tr>
<td>defun Return the Args of an MLambda</td>
<td>278</td>
</tr>
<tr>
<td>defun Return the parts of an MLambda argument</td>
<td>278</td>
</tr>
<tr>
<td>defun pfMLambdaBody</td>
<td>279</td>
</tr>
<tr>
<td>defun Is this a Not node?</td>
<td>279</td>
</tr>
<tr>
<td>defun Return the Arg part of a Not node</td>
<td>279</td>
</tr>
<tr>
<td>defun Construct a NoValue node</td>
<td>279</td>
</tr>
<tr>
<td>defun Is this a NoValue node?</td>
<td>280</td>
</tr>
<tr>
<td>defun Return the Expr part of a NoValue node</td>
<td>280</td>
</tr>
<tr>
<td>defun Construct an Or node</td>
<td>280</td>
</tr>
<tr>
<td>defun Is this an Or node?</td>
<td>280</td>
</tr>
<tr>
<td>defun Return the Left part of an Or node</td>
<td>281</td>
</tr>
<tr>
<td>defun Return the Right part of an Or node</td>
<td>281</td>
</tr>
<tr>
<td>defun Return the part of a parenthesised expression</td>
<td>281</td>
</tr>
<tr>
<td>defun pfPretend</td>
<td>281</td>
</tr>
<tr>
<td>defun Is this a Pretend node?</td>
<td>282</td>
</tr>
<tr>
<td>defun Return the Expression part of a Pretend node</td>
<td>282</td>
</tr>
<tr>
<td>defun Return the Type part of a Pretend node</td>
<td>282</td>
</tr>
</tbody>
</table>
defun Construct a QualType node ................................ 282
defun Construct a Restrict node .................................. 283
defun Is this a Restrict node? .................................. 283
defun Return the Expr part of a Restrict node .................. 283
defun Return the Type part of a Restrict node .................. 283
defun Construct a RetractTo node .................................. 284
defun Construct a Return node ................................ 284
defun Is this a Return node? ................................ 284
defun Return the Expr part of a Return node .................. 284
defun pReturnNoName ........................................ 285
defun Construct a ReturnTyped node .............................. 285
defun Construct a Rule node .................................... 285
defun Return the Lhs of a Rule node .............................. 286
defun Return the Rhs of a Rule node .............................. 286
defun Is this a Rule node? .................................... 286
defun pSecond ............................................... 286
defun Construct a Sequence node .................................. 287
defun Return the Args of a Sequence node ...................... 287
defun Is this a Sequence node? .................................. 287
defun Return the parts of the Args of a Sequence node .......... 287
defun Create a SuchThat node .................................. 288
defun Is this a SuchThat node? .................................. 288
defun Return the Cond part of a SuchThat node ................. 288
defun Create a Tagged node .................................... 288
defun Is this a Tagged node? .................................. 289
defun Return the Expression portion of a Tagged node ....... 289
defun Return the Tag of a Tagged node .......................... 289
defun pfTaggedToTyped .......................................... 289
defun pfTweakIf ............................................. 290
defun Construct a Typed node .................................. 290
defun Is this a Typed node? .................................. 291
defun Return the Type of a Typed node .......................... 291
defun Return the Id of a Typed node ............................. 291
defun Construct a Typing node .................................. 291
defun Return a Tuple node .................................... 292
defun Return a Tuple from a List ................................. 292
defun Is this a Tuple node? .................................. 292
defun Return the Parts of a Tuple node .......................... 293
defun Return the parts of a Tuple ................................. 293
defun Return a list from a Sequence node ....................... 293
defun The comment is attached to all signatutres .............. 293
defun Construct a WDeclare node .................................. 294
defun Construct a Where node .................................. 294
defun Is this a Where node? .................................. 294
defun Return the parts of the Context of a Where node ....... 295
defun Return the Context of a Where node ..................... 295
defun Return the Expr part of a Where node ............................................. 295
defun Construct a While node ................................................................. 295
defun Is this a While node? .................................................................. 296
defun Return the Cond part of a While node ......................................... 296
defun Construct a With node ................................................................. 296
defun Create a Wrong node ................................................................. 296
defun Is this a Wrong node? ................................................................. 297

8 Pftree to s-expression translation ....................................................... 299
  defun Pftree to s-expression translation ............................................. 299
  defun Pftree to s-expression translation inner function ....................... 300
  defun Convert a Literal to an S-expression ....................................... 304
  defun Convert a float to an S-expression ......................................... 305
  defun Change an Application node to an S-expression ....................... 305
  defun Convert a SuchThat node to an S-expression .......................... 307
  defun pfOp2Sex ............................................................................ 308
  defun pmDontQuote? ................................................................. 309
  defun hasOptArgs? ................................................................. 309
  defun Convert a Sequence node to an S-expression ......................... 310
  defun pfSequence2Sex0 .......................................................... 310
  defun Convert a loop node to an S-expression .................................. 311
  defun Change a Collect node to an S-expression .............................. 314
  defun Convert a Definition node to an S-expression ......................... 315
  defun Convert a Lambda node to an S-expression .............................. 316
  defun pfCollectArgTran .............................................................. 317
  defun Convert a Lambda node to an S-expression .............................. 317
  defun Convert a Rule node to an S-expression ................................... 318
  defun Convert the Lhs of a Rule to an S-expression ............................ 318
  defun Convert the Rhs of a Rule to an S-expression ......................... 319
  defun Convert a Rule predicate to an S-expression ............................ 319
  defun patternVarsOf .................................................................... 321
  defun patternVarsOf1 .................................................................... 321
  defun pvarPredTran ....................................................................... 322
  defun Convert the Lhs of a Rule node to an S-expression .................... 322
  defun $dotdot ............................................................................ 323
  defun Translate ops into internal symbols ....................................... 323

9 Keyed Message Handling .................................................................. 325
  defvar $cacheMessages .................................................................. 326
  defvar $msgAlist ........................................................................ 326
  defvar $testingErrorPrefix .......................................................... 326
  defvar $texFormatting ............................................................... 327
  defvar $*msghash* ....................................................................... 327
  defvar $msgdbPrims ...................................................................... 327
  defvar $msgdbPunct ..................................................................... 327
  defvar $msgdbNoBlanksBeforeGroup ............................................... 328
defvar $msgdbNoBlanksAfterGroup .......................... 328
defun Fetch a message from the message database .......... 328
defun Cache messages read from message database ........ 328
defun getKeyedMsg .......................................... 329
defun Say a message using a keyed lookup ................. 329
defun Handle msg formatting and print to file ............ 330
defun Break a message into words ........................ 330
defun Write a msg into spadmsg.listing file ............... 331
defun sayMSG ................................................. 331

10 Stream Utilities ........................................ 333
defun npNull ................................................. 333
defun StreamNull ............................................ 333

11 Code Piles ................................................ 335
defun insertpile ............................................. 335
defun pilePlusComment ..................................... 336
defun pilePlusComments ................................... 336
defun pileTree ............................................... 337
defun pileColumn ............................................ 337
defun pileForests .......................................... 337
defun pileForest ............................................ 338
defun pileForest1 .......................................... 338
defun eqpileTree ........................................... 339
defun pileCtree ............................................. 340
defun pileCforest .......................................... 340
defun enPile ................................................. 340
defun firstTokPosn ........................................ 341
defun lastTokPosn ......................................... 341
defun separatePiles ........................................ 341

12 Dequeue Functions ........................................ 343
defun dqUnit ................................................. 343
defun dqConcat .............................................. 343
defun dqAppend ............................................. 344
defun dqToList .............................................. 344

13 Message Handling .......................................... 345
13.1 The Line Object ........................................ 345
defun Line object creation .................................. 345
defun Line element 0; Extra blanks ........................ 345
defun Line element 1; String ................................ 345
defun Line element 2; Globlal number ....................... 346
defun Line element 2; Set Global number ................... 346
defun Line element 3; Local number ........................ 346
defun Line element 4; Place of origin ....................... 346
defun Line element 4: Is it a filename? ........................................ 347
defun Line element 4: Is it a filename? ........................................ 347
defun Line element 4: Get filename .......................................... 347

13.2 Messages ................................................................. 347
defun msgCreate .......................................................... 347
defun getMsgPosTagOb ...................................................... 348
defun getMsgKey ............................................................ 348
defun getMsgArgL ............................................................ 349
defun setMsgPrefix .......................................................... 349
defun getMsgText ............................................................ 349
defun setMsgText ............................................................ 349
defun getMsgPrefix? ......................................................... 350
defun getMsgTag ............................................................... 350
defun getMsgTag? .............................................................. 350
defun line? .................................................................. 351
defun leader? .................................................................. 351
defun toScreen? ................................................................ 351
defun ncSoftError ............................................................... 351
defun ncHardError ............................................................... 352
defun desiredMsg ............................................................... 352
defun processKeyedError ...................................................... 353
defun msgOutputter .............................................................. 353
defun listOutputter .............................................................. 354
defun getStFromMsg ............................................................ 354
defvar $preLength .............................................................. 355
defun getPreStL ................................................................. 355
defun getPosStL ................................................................. 356
defun ppos .................................................................. 357
defun remFile ................................................................. 357
defun showMsgPos? ............................................................ 357
defvar $imPrGuys ............................................................... 358
defun msgImPr? ................................................................. 358
defun getMsgCatAttr ........................................................... 358
defun getMsgPos ............................................................... 359
defun getMsgFTTag? ............................................................ 359
defun decideHowMuch ........................................................ 359
defun poNopos? ............................................................... 360
defun poPosImmediate? ....................................................... 360
defun poFileName .............................................................. 360
defun poGetLineObject ......................................................... 361
defun poLinePosn ............................................................... 361
defun listDecideHowMuch ..................................................... 361
defun remLine ................................................................. 362
defun getMsgKey? .............................................................. 362
defun getMsgLitSym ........................................................... 362
## CONTENTS

### 14 The Interpreter Syntax
- 14.1 syntax assignment ........................................... 383
- 14.2 syntax blocks .................................................. 386
- 14.3 system clef ..................................................... 388
- 14.4 syntax collection ............................................... 389
- 14.5 syntax for ...................................................... 391
- 14.6 syntax if ....................................................... 395
- 14.7 syntax iterate ............................................... 397
- 14.8 syntax leave .................................................. 398
- 14.9 syntax parallel ............................................... 399
- 14.10 syntax repeat ................................................. 402
- 14.11 syntax suchthat ............................................ 406
- 14.12 syntax syntax ............................................... 407
- 14.13 syntax while ................................................. 407

### 15 Abstract Syntax Trees (ptrees)
- defun Construct a leaf token ................................... 411
- defun Return a part of a node .................................... 412
- defun Compare a part of a node ................................ 412
- defun pfNoPosition? ................................................ 412
- defun poNoPosition? .............................................. 413
- defun tokType ...................................................... 413
- defun tokPart ....................................................... 413
- defun tokPosn ....................................................... 413
- defun pfNoPosition ................................................ 414
- defun poNoPosition ............................................... 414

### 16 Attributed Structures
- defun ncTag .......................................................... 415
- defun ncAlist ....................................................... 415
- defun ncEltQ ......................................................... 416
- defun ncPutQ ......................................................... 416

### 17 Function Selection
- defun ofCategory ................................................... 419
- defun isPartialMode ................................................ 420
- defun hasCaty ...................................................... 420
- defun domArg ....................................................... 422
- defun domArg2 ..................................................... 422
- defun hasSig ....................................................... 423
- defun hasAtt ....................................................... 424
- defun hasSigAnd ................................................... 425
- defun hasSigOr ..................................................... 426
- defun hasAttSig .................................................... 426
- defun hasCate1 ..................................................... 427
- defun hasCatExpression .......................................... 427
18 System Command Handling .... 441

18.1 Variables Used .... 443
- defvar $systemCommands .... 443
- defvar $syscommands .... 444
- defvar $noprojectCommands .... 444

18.2 Functions .... 445
- defun handleNoParseCommands .... 445
- defun Handle a top level command .... 446
- defun Split block into option block .... 447
- defun Tokenize a system command .... 447
- defun Handle system commands .... 448
- defun Select commands matching this user level .... 448
- defun No command begins with this string .... 449
- defun No option begins with this string .... 449
- defvar $oldline .... 449
- defun No command/option begins with this string .... 449
- defun Option not available at this user level .... 450
- defun Command not available at this user level .... 450
- defun Command not available error message .... 450
- defun satisfiesUserLevel .... 451
- defun hasOption .... 451
- defun terminateSystemCommand .... 452
- defun Terminate a system command .... 452
- defun commandAmbiguityError .... 452
- defun getParserMacroNames .... 453
- defun clearParserMacro .... 453
- defun displayMacro .... 453
- defun displayWorkspaceNames .... 454
- defun getWorkspaceNames .... 455
- defun fixObjectForPrinting .... 456
- defun displayProperties,sayFunctionDeps .... 456
- defun displayValue .... 459
21 )browse help page Command 489
  21.1 browse help page man page ................................. 489
  21.2 Overview ................................................... 489
  21.3 Browsers, MathML, and Fonts ............................... 490
  21.4 The axServer/multiServ loop ............................... 491
  21.5 The )browse command ...................................... 492
  21.6 Variables Used ............................................. 493
  21.7 Functions ................................................ 493
  21.8 The server support code ................................. 493

22 )cd help page Command 495
  22.1 cd help page man page ....................................... 495
  22.2 Variables Used ............................................. 496
  22.3 Functions ................................................ 496

23 )clear help page Command 497
  23.1 clear help page man page .................................... 497
  23.2 Variables Used ............................................. 499
defvar $clearOptions ........................................... 499
  23.3 Functions ................................................ 499
defun clear .....................
defvar $clearExcept ..................
defun clearSpad2Cmd ..................
defun clearCmdSortedCaches ..........
defun compiledLookupCheck ..........
defvar $functionTable ..................
defun clearCmdCompletely ..........
defun clearCmdAll ..................
defun clearMacroTable ............
defun clearCmdExcept ............
defun clearCmdParts .............

24 )close help page Command 509
  24.1 close help page man page ................................. 509
  24.2 Functions ................................................ 510
defun queryClients ..................
defun close .............................

25 )compile help page Command 513
  25.1 compile help page man page ............................. 513
  25.2 Functions ........................................... 515
defvar $/editfile ..................
## CONTENTS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 )copyright help page Command</td>
<td>517</td>
</tr>
<tr>
<td>26.1 copyright help page man page</td>
<td>517</td>
</tr>
<tr>
<td>26.2 Functions</td>
<td>522</td>
</tr>
<tr>
<td>defun copyright</td>
<td>522</td>
</tr>
<tr>
<td>defun trademark</td>
<td>523</td>
</tr>
<tr>
<td>27 )credits help page Command</td>
<td>525</td>
</tr>
<tr>
<td>27.1 credits help page man page</td>
<td>525</td>
</tr>
<tr>
<td>27.2 Variables Used</td>
<td>525</td>
</tr>
<tr>
<td>27.3 Functions</td>
<td>525</td>
</tr>
<tr>
<td>defun credits</td>
<td>525</td>
</tr>
<tr>
<td>28 )describe help page Command</td>
<td>527</td>
</tr>
<tr>
<td>28.1 describe help page man page</td>
<td>527</td>
</tr>
<tr>
<td>defvar $describeOptions</td>
<td>528</td>
</tr>
<tr>
<td>28.2 Functions</td>
<td>528</td>
</tr>
<tr>
<td>defun Print comment strings from algebra libraries</td>
<td>528</td>
</tr>
<tr>
<td>defun describeSpad2Cmd</td>
<td>528</td>
</tr>
<tr>
<td>defun cleanline</td>
<td>529</td>
</tr>
<tr>
<td>defun flatten</td>
<td>531</td>
</tr>
<tr>
<td>29 )display help page Command</td>
<td>533</td>
</tr>
<tr>
<td>29.1 display help page man page</td>
<td>533</td>
</tr>
<tr>
<td>defvar $displayOptions</td>
<td>535</td>
</tr>
<tr>
<td>29.2 Functions</td>
<td>535</td>
</tr>
<tr>
<td>defun display</td>
<td>535</td>
</tr>
<tr>
<td>displaySpad2Cmd</td>
<td>535</td>
</tr>
<tr>
<td>defun abbQuery</td>
<td>536</td>
</tr>
<tr>
<td>defun displayOperations</td>
<td>537</td>
</tr>
<tr>
<td>defun yesanswer</td>
<td>537</td>
</tr>
<tr>
<td>defun displayMacros</td>
<td>538</td>
</tr>
<tr>
<td>defun sayExample</td>
<td>539</td>
</tr>
<tr>
<td>defun cleanupLine</td>
<td>540</td>
</tr>
<tr>
<td>30 )edit help page Command</td>
<td>543</td>
</tr>
<tr>
<td>30.1 edit help page man page</td>
<td>543</td>
</tr>
<tr>
<td>defun editSpad2Cmd</td>
<td>544</td>
</tr>
<tr>
<td>defun edit</td>
<td>544</td>
</tr>
<tr>
<td>defun updateSourceFiles</td>
<td>545</td>
</tr>
<tr>
<td>31 )fin help page Command</td>
<td>547</td>
</tr>
<tr>
<td>31.1 fin help page man page</td>
<td>547</td>
</tr>
<tr>
<td>defun Exit from the interpreter to lisp</td>
<td>548</td>
</tr>
<tr>
<td>31.2 Functions</td>
<td>548</td>
</tr>
</tbody>
</table>
CONTENTS

32 )frame help page Command ................................................. 549
  32.1 frame help page man page ........................................ 549
  32.2 Variables Used ..................................................... 551
    Primary variables .................................................. 551
    Used variables ..................................................... 552
  32.3 Data Structures .................................................... 552
    Frames and the Interpreter Frame Ring ............................. 552
  32.4 Accessor Functions ................................................ 552
    0th Frame Component – frameName .................................. 552
    defun frameName ................................................... 552
    1st Frame Component – frameInteractive ............................ 553
    2nd Frame Component – frameIOIndex ................................ 553
    3rd Frame Component – frameHiFiAccess ............................. 553
    4th Frame Component – frameHistList ................................ 553
    5th Frame Component – frameHistListLen ............................ 554
    6th Frame Component – frameHistListAct ............................ 554
    7th Frame Component – frameHistRecord ............................. 554
    8th Frame Component – frameHistoryTable ........................... 554
    9th Frame Component – frameExposureData ........................... 555
  32.5 Functions ........................................................... 555
    Initializing the Interpreter Frame Ring .......................... 555
    Creating a List of all of the Frame Names ....................... 556
    Get Named Frame Environment (aka Interactive) .................. 556
    Create a new, empty Interpreter Frame ........................... 556
    Collecting up the Environment into a Frame ..................... 557
    Update from the Current Frame .................................... 558
    Find a Frame in the Frame Ring by Name ....................... 559
    Update the Current Interpreter Frame ............................ 559
    Move to the next Interpreter Frame in Ring ..................... 560
    Change to the Named Interpreter Frame .......................... 560
    Move to the previous Interpreter Frame in Ring ............... 561
    Add a New Interpreter Frame .................................... 561
    Close an Interpreter Frame ...................................... 562
    Display the Frame Names .......................................... 563
    Import items from another frame .................................. 563
    The top level frame command ..................................... 565
    The top level frame command handler ............................. 566
  32.6 Frame File Messages ................................................ 567

33 )help help page Command ................................................. 569
  33.1 help help page man page ........................................ 569
  33.2 Functions ........................................................... 572
    The top level help command ....................................... 572
    The top level help command handler ............................. 572
    defun newHelpSpad2Cmd ......................................... 572
## Contents

### 34 History Help Page Command

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.1</td>
<td>575</td>
</tr>
<tr>
<td>34.2</td>
<td>578</td>
</tr>
<tr>
<td>34.3</td>
<td>579</td>
</tr>
<tr>
<td>34.4</td>
<td>579</td>
</tr>
<tr>
<td>34.5</td>
<td>584</td>
</tr>
</tbody>
</table>

#### 34.1 History Help Page Man Page
- defvar $oldHistoryFileName
- defvar $historyFileType
- defvar $historyDirectory
- defvar $useInternalHistoryTable

#### 34.2 Initialized History Variables
- defun makeHistFileName
- defun oldHistFileName
- defun histFileName
- defun histInputFileName
- defun initHist
- defun initHistList

#### 34.3 Data Structures

#### 34.4 Functions
- defun setHistoryCore
- defvar $underbar
- defun writeInputLines
- defun resetInCoreHist
- defun changeHistListLen
- defun updateHist
- defun updateInCoreHist
- defun putHist
- defun recordNewValue
- defun recordNewValue0
- defun recordOldValue
- defun recordOldValue0
- defun undoInCore
- defun undoChanges
- defun undoFromFile
- defun saveHistory
- defun restoreHistory
- defun setIOindex
- defun showInput
- defun showInOut
- defun fetchOutput

#### 34.5 LispLib Output Transformations
- defun spadrwrite0
- defun Random write to a stream
- defun spadrwrite
<table>
<thead>
<tr>
<th>Defun/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun spadrread</td>
<td>605</td>
</tr>
<tr>
<td>defun Random read a key from a stream</td>
<td>605</td>
</tr>
<tr>
<td>defun unwritable?</td>
<td>606</td>
</tr>
<tr>
<td>defun writifyComplain</td>
<td>606</td>
</tr>
<tr>
<td>defun safeWritify</td>
<td>606</td>
</tr>
<tr>
<td>defun writify, writifyInner</td>
<td>607</td>
</tr>
<tr>
<td>defun writify</td>
<td>610</td>
</tr>
<tr>
<td>defun spadClosure?</td>
<td>611</td>
</tr>
<tr>
<td>defvar $NonNullStream</td>
<td>611</td>
</tr>
<tr>
<td>defvar $NullStream</td>
<td>611</td>
</tr>
<tr>
<td>defun dewritify, dewritifyInner</td>
<td>612</td>
</tr>
<tr>
<td>defun dewritify</td>
<td>615</td>
</tr>
<tr>
<td>defun ScanOrPairVec, ScanOrInner</td>
<td>615</td>
</tr>
<tr>
<td>defun ScanOrPairVec</td>
<td>616</td>
</tr>
<tr>
<td>defun gensymInt</td>
<td>616</td>
</tr>
<tr>
<td>defun charDigitVal</td>
<td>617</td>
</tr>
<tr>
<td>defun histFileErase</td>
<td>617</td>
</tr>
</tbody>
</table>

34.6 History File Messages ........................................................................ 618

35 )include help page Command .................................................................... 621
   35.1 include help page man page .......................................................... 621
   35.2 Functions ......................................................................................... 621
       defun nclloopInclude1 ....................................................................... 621
       Returns the first non-blank substring of the given string .......... 622
       Open the include file and read it in ........................................ 622
       Return the include filename ...................................................... 622
       Return the next token ............................................................ 623

36 )library help page Command .................................................................... 625
   36.1 library help page man page ................................................................ 625

37 )lisp help page Command .......................................................................... 627
   37.1 lisp help page man page ................................................................... 627
   37.2 Functions ......................................................................................... 628

38 )load help page Command .......................................................................... 629
   38.1 load help page man page ................................................................... 629
       defun The )load command (obsolete) ............................................ 629

39 )ltrace help page Command ........................................................................ 631
   39.1 ltrace help page man page .................................................................. 631
       defun The top level )ltrace function ........................................... 632
   39.2 Variables Used .................................................................................. 632
   39.3 Functions .......................................................................................... 632
CONTENTS

40 )pquit help page Command ........................................... 633
  40.1 pquit help page man page ....................................... 633
  40.2 Functions .......................................................... 634
    The top level pquit command ...................................... 634
    The top level pquit command handler ............................ 634

41 )quit help page Command ............................................. 637
  41.1 quit help page man page ......................................... 637
  41.2 Functions .......................................................... 638
    The top level quit command ....................................... 638
    The top level quit command handler ............................. 638
    Leave the Axiom interpreter ...................................... 639

42 )read help page Command ............................................. 641
  42.1 read help page man page ......................................... 641
    defun The )read command .......................................... 642
    defun Implement the )read command ................................ 642
    defun /read ................................................................ 644

43 )regress help page Command .......................................... 645
  43.1 regress help page man page ..................................... 645

44 )savesystem help page Command ..................................... 649
  44.1 savesystem help page man page .................................. 649
    defun The )savesystem command .................................... 650

45 )set help page Command .............................................. 651
  45.1 set help page man page ........................................... 651
  45.2 Overview .................................................................. 652
  45.3 Variables Used ...................................................... 653
  45.4 Functions ............................................................. 653
    Initialize the set variables ........................................ 653
    Reset the workspace variables .................................... 654
    Display the set option information ............................... 655
    Display the set variable settings .................................. 657
    Translate options values to t or nil ............................. 658
    Translate t or nil to option values ............................... 659
  45.5 The list structure .................................................. 659
  45.6 breakmode ............................................................. 660
    defvar $BreakMode ..................................................... 661
  45.7 debug .................................................................... 661
  45.8 debug lambda type .................................................. 662
    defvar $Lambdatype .................................................... 662
  45.9 debug dalymode ...................................................... 662
    defvar $dalmymode ..................................................... 663
  45.10 compile ............................................................... 663
45.11 compile output ........................................... 664
45.12 Variables Used ........................................... 664
45.13 Functions ............................................. 664
    The set output command handler ......................... 664
    Describe the set output library arguments ............ 665
    defvar $output-library ................................ 665
    Open the output library ................................ 666
45.14 compile input ........................................ 666
45.15 Variables Used ....................................... 667
45.16 Functions ................................ .......... 667
    The set input library command handler ................. 667
    Describe the set input library arguments .......... 668
    Add the input library to the list ................. 668
    defvar $input-libraries .............................. 668
    Drop an input library from the list ................ 669
45.17 expose .................................................. 669
45.18 Variables Used ...................................... 670
    defvar $globalExposureGroupAlist ....................... 670
    defvar $localExposureDataDefault ..................... 696
    defvar $localExposureData .............................. 697
45.19 Functions ........................................... 697
    The top level set expose command handler .......... 697
    The top level set expose add command handler .... 698
    Expose a group ........................................ 699
    The top level set expose add constructor handler .... 700
    The top level set expose drop handler ............. 701
    The top level set expose drop group handler ....... 702
    The top level set expose drop constructor handler .... 703
    Display exposed groups ............................. 705
    Display exposed constructors ..................... 705
    Display hidden constructors .......................... 705
45.20 functions ............................................... 706
45.21 functions cache ....................................... 706
45.22 Variables Used ...................................... 707
    defvar $cacheAlist .................................. 707
45.23 Functions ........................................ 708
    The top level set functions cache handler ........ 708
    defvar $compileDontDefineFunctions .................. 712
45.24 functions recurrence ............................... 712
    defvar $compileRecurrence ............... 712
45.25 fortran ................................................. 713
    ints2floats ........................................ 714
    defvar $fortInts2Floats ................................ 714
    fortiindent .......................................... 715
    defvar $fortIndent .................................. 715
    fortlength ........................................... 715
defvar $reportCoerceIfTrue  ........................................ 738
dropmap ............................................................. 738
defvar $displayDroppedMap ........................................ 738
expose ............................................................... 739
defvar $giveExposureWarning .................................... 739
file ................................................................. 740
defvar $printMsgsToFile ........................................... 740
frame ................................................................. 740
defvar $frameMessages ............................................. 741
highlighting .......................................................... 741
defvar $highlightAllowed .......................................... 742
instant .............................................................. 742
defvar $reportInstantiations ..................................... 742
insteach .............................................................. 743
defvar $reportEachInstantiation— ............................... 743
interponly ............................................................ 744
defvar $reportInterpOnly .......................................... 744
naglink ............................................................... 744
defvar $nagMessages ............................................... 745
number ............................................................... 745
defvar $displayMsgNumber ......................................... 746
prompt ............................................................... 746
defvar $inputPromptType ............................................ 746
selection ............................................................ 747
set ................................................................. 748
defvar $displaySetValue ............................................ 748
startup .............................................................. 748
defvar $displayStartMsgs ......................................... 749
summary ............................................................. 749
defvar $printStatisticsSummaryIfTrue ......................... 749
testing .............................................................. 750
defvar $testingSystem .............................................. 750
time ................................................................. 751
defvar $printTimeIfTrue ............................................ 751
type ................................................................. 752
defvar $printTypeIfTrue ............................................ 752
void ................................................................. 752
defvar $printVoidIfTrue ............................................ 753
45.30 naglink .......................................................... 753
host ................................................................. 754
defvar $nagHost ....................................................... 754
defun setNagHost ..................................................... 755
defun describeSetNagHost ......................................... 755
persistence .......................................................... 755
defvar $fortPersistence ............................................ 756
defun setFortPers ..................................................... 756
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun describeFortPersistence</td>
<td>757</td>
</tr>
<tr>
<td>messages</td>
<td>758</td>
</tr>
<tr>
<td>double</td>
<td>758</td>
</tr>
<tr>
<td>defvar $nagEnforceDouble</td>
<td>758</td>
</tr>
<tr>
<td>45.31output</td>
<td>759</td>
</tr>
<tr>
<td>abbreviate</td>
<td>760</td>
</tr>
<tr>
<td>defvar $abbreviateTypes</td>
<td>760</td>
</tr>
<tr>
<td>algebra</td>
<td>761</td>
</tr>
<tr>
<td>defvar $algebraFormat</td>
<td>761</td>
</tr>
<tr>
<td>defvar $algebraOutputFile</td>
<td>762</td>
</tr>
<tr>
<td>defvar $algebraOutputStream</td>
<td>762</td>
</tr>
<tr>
<td>defun setOutputAlgebra</td>
<td>763</td>
</tr>
<tr>
<td>defun describeSetOutputAlgebra</td>
<td>765</td>
</tr>
<tr>
<td>characters</td>
<td>766</td>
</tr>
<tr>
<td>defun setOutputCharacters</td>
<td>766</td>
</tr>
<tr>
<td>fortran</td>
<td>768</td>
</tr>
<tr>
<td>defvar $fortranFormat</td>
<td>769</td>
</tr>
<tr>
<td>defvar $fortranOutputFile</td>
<td>769</td>
</tr>
<tr>
<td>defun setOutputFortran</td>
<td>770</td>
</tr>
<tr>
<td>defun describeSetOutputFortran</td>
<td>772</td>
</tr>
<tr>
<td>fraction</td>
<td>773</td>
</tr>
<tr>
<td>defvar $fractionDisplayType</td>
<td>773</td>
</tr>
<tr>
<td>length</td>
<td>774</td>
</tr>
<tr>
<td>defvar $margin</td>
<td>774</td>
</tr>
<tr>
<td>defvar $lineLength</td>
<td>774</td>
</tr>
<tr>
<td>mathml</td>
<td>775</td>
</tr>
<tr>
<td>defvar $mathmlFormat</td>
<td>775</td>
</tr>
<tr>
<td>defvar $mathmlOutputFile</td>
<td>775</td>
</tr>
<tr>
<td>defun setOutputMathml</td>
<td>776</td>
</tr>
<tr>
<td>defun describeSetOutputMathml</td>
<td>778</td>
</tr>
<tr>
<td>html</td>
<td>779</td>
</tr>
<tr>
<td>defvar $htmlFormat</td>
<td>780</td>
</tr>
<tr>
<td>defvar $htmlOutputFile</td>
<td>780</td>
</tr>
<tr>
<td>defun setOutputHtml</td>
<td>781</td>
</tr>
<tr>
<td>defun describeSetOutputHtml</td>
<td>783</td>
</tr>
<tr>
<td>openmath</td>
<td>784</td>
</tr>
<tr>
<td>defvar $openMathFormat</td>
<td>784</td>
</tr>
<tr>
<td>defvar $openMathOutputFile</td>
<td>784</td>
</tr>
<tr>
<td>defun setOutputOpenMath</td>
<td>785</td>
</tr>
<tr>
<td>defun describeSetOutputOpenMath</td>
<td>788</td>
</tr>
<tr>
<td>script</td>
<td>788</td>
</tr>
<tr>
<td>defvar $formulaFormat</td>
<td>789</td>
</tr>
<tr>
<td>defvar $formulaOutputFile</td>
<td>789</td>
</tr>
<tr>
<td>defun setOutputFormula</td>
<td>790</td>
</tr>
<tr>
<td>defun describeSetOutputFormula</td>
<td>792</td>
</tr>
<tr>
<td>scripts</td>
<td>793</td>
</tr>
</tbody>
</table>
defvar $linearFormatScripts ............................................. 793
showeditor ................................................................. 794
defvar $useEditorForShowOutput ........................................... 794
tex ............................................................................ 795
defvar $texFormat ........................................................... 795
defvar $texOutputFile ......................................................... 796
defun setOutputTex ................................................................ 796
defun describeSetOutputTex ..................................................... 798

45.32 quit ............................................................................ 799
defvar $quitCommandType ....................................................... 800

45.33 streams ....................................................................... 800
calculate ........................................................................... 801
defvar $streamCount ............................................................. 801
defun setStreamsCalculate ....................................................... 801

defun describeSetStreamsCalculate .............................................. 802
showall ............................................................................ 802
defvar $streamsShowAll .......................................................... 803

45.34 system .......................................................................... 803
functioncode ..................................................................... 804
defvar $reportCompilation ....................................................... 804
optimization ....................................................................... 805
defvar $reportOptimization ....................................................... 805
prettyprint ........................................................................ 805
defvar $prettyprint ............................................................... 806

45.35 userlevel ...................................................................... 806
defvar $UserLevel ................................................................ 807
defvar $setOptionNames ........................................................ 808

45.36 Set code ........................................................................ 808
defun set ............................................................................ 808
defun set1 .......................................................................... 808

46 )show help page Command ....................................................... 813
46.1 show help page man page ...................................................... 813
defun The )show command ....................................................... 814
defun The internal )show command ........................................... 814
defun reportOperations ......................................................... 815
defun reportOpsFromLisplib0 ................................................. 817
defun reportOpsFromLisplib1 .................................................... 817
defun reportOpsFromLisplib ...................................................... 818
defun isExposedConstructor ...................................................... 820
defun displayOperationsFromLisplib ........................................ 820
defun reportOpsFromUnitDirectly0 .......................................... 821

defun reportOpsFromUnitDirectly .............................................. 821
defun getOplistForConstructorForm ........................................... 824
defun getOplistWithUniqueSignatures ....................................... 825
defun reportOpsFromUnitDirectly1 ............................................ 825
defun sayShowWarning . 826

47 )spool help page Command 827
  47.1 spool help page man page 827

48 )summary help page Command 829
  48.1 summary help page man page 829
  defun summary 830

49 )synonym help page Command 831
  49.1 synonym help page man page 831
  defun The )synonym command 832
  defun The )synonym command implementation 832
  defun Return a sublist of applicable synonyms 833
  defun Get the system command from the input line 833
  defun Remove system keyword 834
  defun processSynonymLine 835

50 )system help page Command 837
  50.1 system help page man page 837

51 )tangle help page Command 839
  51.1 tangle help page man page 839

52 )trace help page Command 841
  52.1 trace help page man page 841
  The trace global variables 845
  defvar $traceNoisely 846
  defvar $reportSpadtrace 846
  defvar $optionAlist 846
  defvar $tracedMapSignatures 846
  defvar $traceOptionList 846
  defun trace 847
  defun traceSpad2Cmd 847
  defun trace1 848
  defun getTraceOptions 852
  defun saveMapSig 853
  defun getMapSig 853
  defun getTraceOption,ln 853
  defun getTraceOption 854
  defun traceOptionError 857
  defun resetTimers 858
  defun resetSpacers 858
  defun resetCounters 858
  defun ptimers 859
  defun pspacers 859
defun pcounters .................................................. 860
defun transOnlyOption .......................................... 860
defun stackTraceOptionError .................................. 861
defun removeOption .............................................. 861
defun domainToGenvar ........................................... 861
defun genDomainTraceName ...................................... 862
defun untrace ..................................................... 862
defun transTraceItem ............................................ 863
defun removeTracedMapSigs ..................................... 864
defun coerceTraceArgs2E ....................................... 864
defun coerceSpadArgs2E ......................................... 865
defun subTypes .................................................. 866
defun coerceTraceFunValue2E .................................. 867
defun coerceSpadFunValue2E .................................... 868
defun isListOfIdentifiers ....................................... 868
defun isListOfIdentifiersOrStrings ............................ 869
defun getMapSubNames ........................................... 869
defun getPreviousMapSubNames .................................. 870
defun lassocSub .................................................. 871
defun rassocSub .................................................. 871
defun isUncompiledMap .......................................... 871
defun isInterpOnlyMap .......................................... 872
defun augmentTraceNames ....................................... 872
defun untraceMapSubNames ...................................... 873
defun funfind,LAM ................................................ 873
defmacro funfind ................................................ 874
defun isDomainOrPackage ....................................... 874
defun isTraceGensym ............................................. 875
defun spadTrace,g ................................................ 875
defun spadTrace,isTraceable .................................... 875
defun spadTrace .................................................. 876
defun traceDomainLocalOps ..................................... 880
defun untraceDomainLocalOps ................................... 880
defun traceDomainConstructor .................................. 880
defun untraceDomainConstructor,keepTraced? ................ 882
defun untraceDomainConstructor ................................. 883
defun flattenOperationAlist ...................................... 883
defun mapLetPrint ................................................ 884
defun letPrint ................................................... 885
defun letPrint2 ................................................... 885
defun letPrint3 ................................................... 888
defun getAliasIfTracedMapParameter ............................. 889
defun recordFrame .......................... 923
defun diffAlist ................................ 924
defun reportUndo ................................ 927
defun clearFrame ................................ 929
Undo previous n commands ...................... 929
defun undoSteps ................................ 930
defun undoSingleStep ............................ 931
defun undoLocalModemapHack .................... 933
Remove undo lines from history write ......... 933

54 )what help page Command .......................... 937
54.1 what help page man page .................... 937
defvar $whatOptions .............................. 937
defun what ......................................... 939
defun whatSpad2Cmd,fixpat ....................... 939
defun whatSpad2Cmd ............................... 940
defun Show keywords for )what command ........ 941
defun The )what commands implementation .... 941
defun Find all names contained in a pattern .... 942
defun Find function of names contained in pattern 943
defun satisfiesRegularExpressions .............. 943
defun filterAndFormatConstructors ............. 944
defun whatConstructors .......................... 945
Display all operation names containing the fragment 945

55 )with help page Command ....................... 947
55.1 with help page man page ..................... 947
defun with ......................................... 947

56 )workfiles help page Command ................. 949
56.1 workfiles help page man page ............... 949
defun workfiles ................................. 949
defun workfilesSpad2Cmd ......................... 949

57 )zsystemdevelopment help page Command ...... 953
57.1 zsystemdevelopment help page man page .... 953
defun zsystemdevelopment ....................... 953
defun zsystemDevelopmentSpad2Cmd ................ 953
defun zsystemdevelopment1 ...................... 954

58 Handlers for Special Forms .................. 957
defun getAndEvalConstructorArgument .......... 958
defun replaceSharps .............................. 958
defun isDomainValuedVariable ................... 959
defun evalCategory ............................... 959
59 Handling input files
  defun Handle axiom.input file ........................................ 961
  defvar $boot-line-stack ............................................ 961
  defvar Sin-stream .................................................. 961
  defvar $out-stream ................................................ 962
  defvar $file-closed ................................................ 962
  defvar $echo-meta .................................................. 962
  defvar $noSubsumption .............................................. 962
  defvar $envHashTable ............................................... 963
  defun Dynamically add bindings to the environment .......... 963
  defun Fetch a property list for a symbol from CategoryFrame . 964
  defun Search for a binding in the environment list .......... 964
  defun Search for a binding in the current environment .... 964
  defun searchTailEnv .............................................. 965

60 File Parsing
  defun Bind a variable in the interactive environment .... 967
  defvar $line-handler .............................................. 967
  defvar $spad-errors ............................................... 967
  defvar $xtokenreader ............................................. 968
  defun Initialize the spad reader ............................... 968
  defun spad-syntax-error ......................................... 969
  defun spad-long-error ............................................ 970
  defun spad-short-error .......................................... 970
  defun spad-error-loc ............................................. 970
  defun iostat ....................................................... 970
  defun next-lines-show ............................................ 971
  defun token-stack-show .......................................... 971
  defun ioclear ..................................................... 972
  defun Set boot-line-stack to nil ................................ 972

61 Handling output
  61.1 Special Character Tables .................................... 975
  defvar $defaultSpecialCharacters ................................ 975
  defvar $plainSpecialCharacters0 .............................. 976
  defvar $plainSpecialCharacters1 .............................. 976
  defvar $plainSpecialCharacters2 .............................. 977
  defvar $plainSpecialCharacters3 .............................. 977
  defvar $plainRTspecialCharacters ......................... 978
  defvar $RTspecialCharacters .................................. 978
  defvar $specialCharacters ....................................... 979
  defvar $specialCharacterAlist ............................... 979
  defun Look up a special character code for a symbol .... 980
62 Stream and File Handling

- defun make-instream ........................................ 981
- defun make-outstream ......................................... 981
- defun make-appendstream ..................................... 982
- defun defiostream ............................................. 982
- defun shut ..................................................... 982
- defun eofp ..................................................... 983
- defun makeStream ............................................. 983
- defun Construct a new input file name ...................... 983
- defun getDirectoryList ....................................... 984
- defun probeName .............................................. 984
- defun makeFullNamestring ................................... 985
- defun Replace a file by erase and rename ................... 985

63 The Spad Server Mechanism

- defun openserver ............................................ 987

64 Axiom Build-time Functions

- defun spad-save .............................................. 989

65 Exposure Groups

66 Databases

66.1 Database structure ......................................... 993
- kaf File Format ............................................... 993
- Database Files ............................................... 994
- defstruct $database .......................................... 996
- defvar $*defaultdomain-list* ................................. 996
- defvar $*operation-hash* ..................................... 997
- defvar $*hasCategory-hash* ................................ 997
- defvar $*miss* ................................................ 997
- Database streams ............................................ 998
- defvar $*interp-stream* ...................................... 998
- defvar $*interp-stream-stamp* ............................... 998
- defvar $*operation-stream* .................................. 998
- defvar $*operation-stream-stamp* ......................... 999
- defvar $*browse-stream* .................................... 999
- defvar $*browse-stream-stamp* .............................. 999
- defvar $*category-stream* .................................. 999
- defvar $*category-stream-stamp* ......................... 1000
- defvar $*allconstructors* .................................. 1000
- defvar $*allOperations* .................................... 1000
- defun Reset all hash tables before saving system ........... 1000
- defun Preload algebra into saved system .................... 1001
- defun Open the interp database .............................. 1003
- defun Open the browse database ............................. 1005
defun Open the category database .......................... 1006
defun Open the operations database .......................... 1007
defun Add operations from newly compiled code ............... 1007
defun Show all database attributes of a constructor ............ 1008
defun Set a value for a constructor key in the database .......... 1009
defun Delete a value for a constructor key in the database ...... 1009
defun Get constructor information for a database key ............ 1010
defun The )library top level command ........................ 1013
defun Read a local filename and update the hash tables .......... 1014
defun Update the database from an nrlib index.kaf file .......... 1015
defun updateDatabase .................................. 1017
defun Make new databases .................................. 1018
defun saveDependentsHashTable ................................ 1022
defun saveUsersHashTable .................................. 1022
defun Construct the proper database full pathname ............... 1023
Building the interp database from hash tables .................... 1023
defun Write the interp database ................................ 1027
Building the browse database from hash tables ................. 1029
defun Write the browse database ................................ 1029
Building the category database from hash tables ................. 1030
defun Write the category database ................................ 1030
Building the operation database from hash tables ............... 1031
defun Write the operations database ................................ 1031
Database support operations .................................. 1032
defun Data preloaded into the image at build time ............... 1032
defun Return all constructors .................................. 1032
defun Return all operations .................................. 1033

67 System Statistics .......................................... 1035
   defun statisticsInitialization ................................ 1035
67.1 Lisp Library Handling ..................................... 1035
   defun loadLib ........................................ 1035
   defun isSystemDirectory .................................. 1037
   defun loadLibNoDirectory .................................. 1037
   defun loadFunctor ........................................ 1038

68 Special Lisp Functions ...................................... 1039
68.1 Axiom control structure macros .............................. 1039
   defun put ........................................ 1039
   defmacro while ....................................... 1039
   defmacro whileWithResult ................................ 1040
68.2 Filename Handling ........................................ 1040
   defun namestring ...................................... 1040
   defun pathnameName ..................................... 1040
   defun pathnameType ...................................... 1040
   defun pathnameTypeld ................................... 1041
## CONTENTS

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun mergePathnames</td>
<td>1041</td>
</tr>
<tr>
<td>defun pathnameDirectory</td>
<td>1041</td>
</tr>
<tr>
<td>defun Axiom pathnames</td>
<td>1042</td>
</tr>
<tr>
<td>defun makePathname</td>
<td>1042</td>
</tr>
<tr>
<td>defun Delete a file</td>
<td>1042</td>
</tr>
<tr>
<td>defun wrap</td>
<td>1043</td>
</tr>
<tr>
<td>defun lotsof</td>
<td>1043</td>
</tr>
<tr>
<td>defmacro startsId?</td>
<td>1044</td>
</tr>
<tr>
<td>defun hput</td>
<td>1044</td>
</tr>
<tr>
<td>defmacro hget</td>
<td>1044</td>
</tr>
<tr>
<td>defun hkeys</td>
<td>1044</td>
</tr>
<tr>
<td>defun digitp</td>
<td>1045</td>
</tr>
<tr>
<td>defun pname</td>
<td>1045</td>
</tr>
<tr>
<td>defun size</td>
<td>1045</td>
</tr>
<tr>
<td>defun strpos</td>
<td>1045</td>
</tr>
<tr>
<td>defun strposl</td>
<td>1046</td>
</tr>
<tr>
<td>defun qenum</td>
<td>1046</td>
</tr>
<tr>
<td>defmacro identp</td>
<td>1046</td>
</tr>
<tr>
<td>defun concat</td>
<td>1047</td>
</tr>
<tr>
<td>defun canFuncall?</td>
<td>1047</td>
</tr>
<tr>
<td>defun brightprint</td>
<td>1048</td>
</tr>
<tr>
<td>defun brightprint-0</td>
<td>1048</td>
</tr>
<tr>
<td>defun member</td>
<td>1048</td>
</tr>
<tr>
<td>defun messageprint</td>
<td>1048</td>
</tr>
<tr>
<td>defun messageprint-1</td>
<td>1049</td>
</tr>
<tr>
<td>defun messageprint-2</td>
<td>1049</td>
</tr>
<tr>
<td>defun sayBrightly1</td>
<td>1049</td>
</tr>
<tr>
<td>defmacro assq</td>
<td>1050</td>
</tr>
<tr>
<td>defun A version of GET that works with lists</td>
<td>1050</td>
</tr>
</tbody>
</table>

### 69 Record, Union, Mapping, and Enumeration

#### 70 Common Lisp Algebra Support

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>70.1 ApplicationProgramInterface</td>
<td>1053</td>
</tr>
<tr>
<td>defun Report what domains get instantiated</td>
<td>1053</td>
</tr>
<tr>
<td>70.2 InputForm</td>
<td>1054</td>
</tr>
<tr>
<td>defun unparselnputForm</td>
<td>1054</td>
</tr>
<tr>
<td>70.3 Void</td>
<td>1054</td>
</tr>
<tr>
<td>defun voidValue</td>
<td>1054</td>
</tr>
<tr>
<td>70.4 U8Vector</td>
<td>1054</td>
</tr>
<tr>
<td>defmacro qvlenU8</td>
<td>1054</td>
</tr>
<tr>
<td>defmacro eltU8</td>
<td>1055</td>
</tr>
<tr>
<td>defmacro seteltU8</td>
<td>1055</td>
</tr>
<tr>
<td>defun getRefvU8</td>
<td>1055</td>
</tr>
<tr>
<td>70.5 U16Vector</td>
<td>1055</td>
</tr>
<tr>
<td>defmacro qvlenU16</td>
<td>1055</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>70.16</td>
<td>SingleInteger</td>
</tr>
<tr>
<td></td>
<td>defun ncParseFromString</td>
</tr>
<tr>
<td></td>
<td>defun qquotient</td>
</tr>
<tr>
<td></td>
<td>defun qremainder</td>
</tr>
<tr>
<td></td>
<td>defmacro qsdifference</td>
</tr>
<tr>
<td></td>
<td>defmacro qslessp</td>
</tr>
<tr>
<td></td>
<td>defmacro qssub1</td>
</tr>
<tr>
<td></td>
<td>defmacro qsminus</td>
</tr>
<tr>
<td></td>
<td>defmacro qslplus</td>
</tr>
<tr>
<td></td>
<td>defmacro qstimes</td>
</tr>
<tr>
<td></td>
<td>defmacro qsabsval</td>
</tr>
<tr>
<td></td>
<td>defmacro qsoddp</td>
</tr>
<tr>
<td></td>
<td>defmacro qszerop</td>
</tr>
<tr>
<td></td>
<td>defmacro qsmax</td>
</tr>
<tr>
<td></td>
<td>defmacro qsmn</td>
</tr>
<tr>
<td>70.17</td>
<td>Boolean</td>
</tr>
<tr>
<td></td>
<td>defun The Boolean = function support</td>
</tr>
<tr>
<td>70.18</td>
<td>IndexedBits</td>
</tr>
<tr>
<td></td>
<td>defmacro truth-to-bit</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits new function support</td>
</tr>
<tr>
<td></td>
<td>defmacro bit-to-truth</td>
</tr>
<tr>
<td></td>
<td>defmacro bvec-elt</td>
</tr>
<tr>
<td></td>
<td>defmacro bvec-setelt</td>
</tr>
<tr>
<td></td>
<td>defmacro bvec-size</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits concat function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits copy function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits  = function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits &lt; function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits And function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits Or function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits xor function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits nand function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits nor function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits not function support</td>
</tr>
<tr>
<td>70.19</td>
<td>KeyedAccessFile</td>
</tr>
<tr>
<td></td>
<td>defun KeyedAccessFile defstream function support</td>
</tr>
<tr>
<td></td>
<td>defun KeyedAccessFile defstream function support</td>
</tr>
<tr>
<td>70.20</td>
<td>Table</td>
</tr>
<tr>
<td></td>
<td>defun Table InnerTable support</td>
</tr>
<tr>
<td></td>
<td>defun compiledLookup</td>
</tr>
<tr>
<td></td>
<td>defun basicLookup</td>
</tr>
<tr>
<td></td>
<td>defun lookupInDomainVector</td>
</tr>
<tr>
<td></td>
<td>defun basicLookupCheck Defaults</td>
</tr>
<tr>
<td></td>
<td>defun oldCompLookup</td>
</tr>
<tr>
<td></td>
<td>defun NRTevalDomain</td>
</tr>
<tr>
<td>Function/Method</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>defun FileName name function implementation</td>
<td>1092</td>
</tr>
<tr>
<td>defun FileName extension function implementation</td>
<td>1092</td>
</tr>
<tr>
<td>defun FileName exists? function implementation</td>
<td>1092</td>
</tr>
<tr>
<td>defun FileName readable? function implementation</td>
<td>1092</td>
</tr>
<tr>
<td>defun FileName writeable? function implementation</td>
<td>1093</td>
</tr>
<tr>
<td>defun FileName writeable? function support</td>
<td>1093</td>
</tr>
<tr>
<td>defun FileName new function implementation</td>
<td>1093</td>
</tr>
<tr>
<td>70.31 DoubleFloat</td>
<td>1094</td>
</tr>
<tr>
<td>defmacro DFLessThan</td>
<td>1094</td>
</tr>
<tr>
<td>defmacro DFUnaryMinus</td>
<td>1094</td>
</tr>
<tr>
<td>defmacro DFMinusp</td>
<td>1094</td>
</tr>
<tr>
<td>defmacro DFZerop</td>
<td>1094</td>
</tr>
<tr>
<td>defmacro DFAdd</td>
<td>1095</td>
</tr>
<tr>
<td>defmacro DFSubtract</td>
<td>1095</td>
</tr>
<tr>
<td>defmacro DFMultiply</td>
<td>1095</td>
</tr>
<tr>
<td>defmacro DFIntegerMultiply</td>
<td>1095</td>
</tr>
<tr>
<td>defmacro DFMax</td>
<td>1096</td>
</tr>
<tr>
<td>defmacro DFMin</td>
<td>1096</td>
</tr>
<tr>
<td>defmacro DFEql</td>
<td>1096</td>
</tr>
<tr>
<td>defmacro DFDivide</td>
<td>1096</td>
</tr>
<tr>
<td>defmacro DFIntegerDivide</td>
<td>1097</td>
</tr>
<tr>
<td>defmacro DFLogE</td>
<td>1097</td>
</tr>
<tr>
<td>defmacro DFLog</td>
<td>1097</td>
</tr>
<tr>
<td>defmacro DFIntegerExpt</td>
<td>1098</td>
</tr>
<tr>
<td>defmacro DFExpt</td>
<td>1098</td>
</tr>
<tr>
<td>defmacro DFExp</td>
<td>1098</td>
</tr>
<tr>
<td>defmacro DFSin</td>
<td>1098</td>
</tr>
<tr>
<td>defmacro DFCos</td>
<td>1099</td>
</tr>
<tr>
<td>defmacro DFTan</td>
<td>1099</td>
</tr>
<tr>
<td>defmacro DFAcos</td>
<td>1100</td>
</tr>
<tr>
<td>defmacro DFAtan</td>
<td>1100</td>
</tr>
<tr>
<td>defmacro DFAtan2</td>
<td>1100</td>
</tr>
<tr>
<td>defmacro DFSinh</td>
<td>1100</td>
</tr>
<tr>
<td>defmacro DFCosh</td>
<td>1101</td>
</tr>
<tr>
<td>defmacro DFTanh</td>
<td>1101</td>
</tr>
<tr>
<td>defmacro DFAcosh</td>
<td>1101</td>
</tr>
<tr>
<td>defmacro DFAtanh</td>
<td>1102</td>
</tr>
<tr>
<td>defmacro DFAtanh2</td>
<td>1102</td>
</tr>
<tr>
<td>defun Machine specific float numerator</td>
<td>1102</td>
</tr>
<tr>
<td>defun Machine specific float denominator</td>
<td>1103</td>
</tr>
<tr>
<td>defun Machine specific float sign</td>
<td>1103</td>
</tr>
<tr>
<td>defun Machine specific float bit length</td>
<td>1103</td>
</tr>
<tr>
<td>defun Decode floating-point values</td>
<td>1103</td>
</tr>
<tr>
<td>defun The cotangent routine</td>
<td>1104</td>
</tr>
</tbody>
</table>
71 OpenMath

71.1 A Technical Overview[4] ................................................. 1109
The OpenMath Architecture .............................................. 1109
OpenMath Encodings ...................................................... 1111
Content Dictionaries ....................................................... 1112
OpenMath in Action ........................................................ 1114

71.2 Technical Details[3] ..................................................... 1115

71.3 The Structure of the API ............................................... 1115

71.4 OpenMath Expressions ................................................. 1116
Expressions ................................................................. 1116
Symbols ..................................................................... 1116
Encoding and Decoding OpenMath Expressions ..................... 1116

71.5 Big Integers ............................................................. 1117

71.6 Functions Dealing with OpenMath Devices ........................ 1117

71.7 Functions to Write OpenMath Expressions to Devices ............ 1118
Beginning and Ending Objects ............................................. 1118
Writing Basic Objects ..................................................... 1119
Writing Structured Objects ............................................... 1119

71.8 Functions to Extract OpenMath Expressions from Devices ....... 1120
Testing the type of the current token .................................. 1120
Extracting the current token ............................................. 1121

71.9 Comments in the SGML/XML Encodings ............................. 1124

71.10 I/O Functions for Devices ............................................ 1125

71.11 Communications ....................................................... 1125
Functions to Initiate an OMconn ....................................... 1126

71.12 Parameters ................................................................ 1127

71.13 Miscellaneous Functions and Variables ............................. 1127

71.14 The OM.h header file ................................................. 1128

71.15 Axiom OpenMath stub functions ..................................... 1137
Axiom specific functions .................................................. 1137
defun om-Read .............................................................. 1137
defun om-listCDs ............................................................. 1138
defun om-listSymbols ....................................................... 1138
defun om-supportsCD ...................................................... 1138
defun om-supportsSymbol ........................................ 1138
Lisp conversion functions ........................................ 1139
defun om-setDevEncoding ......................................... 1139
Device manipulation functions .................................... 1139
defun om-openFileDev ............................................. 1139
defun om-openStringDev ........................................... 1140
defun om-closeDev .................................................. 1140
Connection manipulation functions .............................. 1140
defun om-makeConn .................................................. 1140
defun om-closeConn .................................................. 1140
defun om-getConnInDev ............................................. 1141
defun om-getConnOutDev .......................................... 1141
Client/Server functions .......................................... 1141
defun om-bindTCP .................................................... 1141
defun om-connectTCP ................................................ 1142
Device input/output functions .................................... 1142
defun om-getApp ..................................................... 1143
defun om-getAtp ...................................................... 1144
defun om-getAttr ..................................................... 1144
defun om-getBind ..................................................... 1144
defun om-getBVar ..................................................... 1144
defun om-getByteArray ............................................. 1145
defun om-getEndApp ............................................... 1145
defun om-getEndAtp ................................................. 1145
defun om-getEndAttr ............................................... 1145
defun om-getEndBind ............................................... 1146
defun om-getEndBVar ............................................... 1146
defun om-getEndError ............................................... 1146
defun om-getEndObject ............................................. 1146
defun om-getError ................................................... 1147
defun om-getFloat ................................................... 1147
defun om-getInt ...................................................... 1147
defun om-getObject .................................................. 1147
defun om-getString .................................................. 1148
defun om-getSymbol ................................................ 1148
defun om-getType .................................................... 1148
defun om-getVar ..................................................... 1148
defun om-putApp ...................................................... 1149
defun om-putAtp ...................................................... 1149
defun om-putAttr ..................................................... 1149
defun om-putBind ..................................................... 1149
defun om-putBVar ..................................................... 1150
defun om-putByteArray ............................................ 1150
defun om-putEndApp ................................................ 1150
defun om-putEndAtp ................................................ 1150
defun om-putEndAttr ................................................. 1151
defun om-putEndBind .................................................. 1151
defun om-putEndBVar .................................................. 1151
defun om-putEndError .................................................. 1151
defun om-putEndObject ............................................... 1152
defun om-putError ..................................................... 1152
defun om-putFloat .................................................... 1152
defun om-putInt ....................................................... 1152
defun om-putObject .................................................. 1153
defun om-putString ................................................... 1153
defun om-putSymbol .................................................. 1153
defun om-putVar ....................................................... 1153
defun om-stringToStringPtr ........................................ 1154
defun om-stringPtrToString ......................................... 1154

72 NRLIB code.lisp support code 1155
defun makeByteWordVec2 ............................................... 1155
defmacro spadConstant ............................................... 1155

defvar $*monitor-domains* ......................................... 1163
defvar $*monitor-nrlibs* .......................................... 1163
defvar $*monitor-table* ........................................... 1164
defstruct $monitor-data ........................................... 1164
defstruct $libstream ............................................... 1164
defun Initialize the monitor statistics hashtable ................ 1164
defun End the monitoring process, we cannot restart ............ 1165
defun Return a list of the monitor-data structures ............... 1165
defun Add a function to be monitored ................................ 1166
defun Remove a function being monitored ......................... 1166
defun Enable all (or optionally one) function for monitoring ... 1166
defun Disable all (optionally one) function for monitoring ... 1167
defun Reset the table count for the table (or a function) ..... 1167
defun Incr the count of fn by 1 .................................. 1168

defun Decr the count of fn by 1 .................................. 1168

defun Return the monitor information for a function ............ 1169

defun Hang a monitor call on all of the defuns in a file ......... 1169

defun Return a list of the functions with zero count fields ... 1169

defun Return a list of functions with non-zero counts .......... 1170

defun Write out a list of symbols or structures to a file ....... 1170

defun Save the *monitor-table* in loadable form ................. 1171

defun restore a checkpointed file ................................ 1171

defun Printing help documentation ................................ 1172

Monitoring algebra files .......................................... 1174

defun Monitoring algebra code.lsp files ........................... 1174

defun Monitor autoloaded files .................................... 1174

defun Monitor an nrlib ............................................. 1175
defun Given a monitor-data item, extract the nrlib name .......................... 1175
defun Is this an exposed algebra function? ............................................ 1176
defun Monitor exposed domains ............................................................ 1176
defun Generate a report of the monitored domains .................................... 1177
defun Parse an )abbrev expression for the domain name ............................ 1178
defun Given a spad file, report all nrlibs it creates ................................ 1178
defun Print percent of functions tested ................................................... 1179
defun Find all monitored symbols containing the string ............................. 1179

74 HyperDoc Basic Command support ....................................................... 1181
  defun Basic Command matrix entry ....................................................... 1182
  defun Read Matrix ............................................................................... 1182
  defun Input Matrix By Formula ............................................................. 1183
  defun Basic Command Matrix by Formula generate .................................... 1185
  defun Input Explicit Matrix ................................................................... 1185
  defun Basic Command generate explicit matrix ........................................ 1187
  defun Basic Command generate matrix .................................................... 1188
  defun Basic Command iteration ............................................................. 1189
  defun Indefinite Integration Basic Command ......................................... 1189
  defun bcIndefiniteIntegrateGen .............................................................. 1190
  defun Definite Integration Basic Command ............................................ 1190
  defun bcDefiniteIntegrateGen .................................................................. 1192
  defun Sum Basic Command ...................................................................... 1192
  defun bcSumGen .................................................................................... 1194
  defun Sum Basic Command ...................................................................... 1194
  defun bcProductGen ............................................................................... 1195
  defun Differentiate Basic Command ......................................................... 1195
  defun bcDifferentiateGen ....................................................................... 1196
  defun Draw Basic Command .................................................................... 1197
  defun Draw Basic Command by Function ................................................ 1198
  defun bcDraw2DfunGen .......................................................................... 1199
  defun Draw Basic Command by Parameters ............................................. 1200
  defun bcDraw2DparGen ............................................................................ 1201
  defun Draw Basic Command by Equation Solution .................................... 1202
  defun bcDraw2DSolveGen ........................................................................ 1203
  defun Draw Basic Command by 3D function ............................................. 1204
  defun bcDraw3DfunGen ............................................................................ 1205
  defun Draw Basic Command by 3D parameterized tube .............................. 1206
  defun bcDraw3DparGen ............................................................................ 1207
  defun Draw Basic Command by 3D parameterized function ....................... 1208
  defun bcDraw3Dpar1Gen .......................................................................... 1209
  defun Series Basic Command ................................................................... 1210
  defun Series Basic Command expand around a point ................................ 1211
  defun bcSeriesExpansionGen ................................................................... 1212
  defun Series Basic Command series by formula ...................................... 1212
  defun Taylor Series Basic Command ....................................................... 1213
## CONTENTS

- defun bcSeriesByFormulaGen .................................................... 1215
- defun Laurent Series Basic Command ........................................... 1215
- defun Puiseux Series Basic Command ......................................... 1217
- defun bcTaylorSeriesGen .......................................................... 1219
- defun bcLaurentSeriesGen ......................................................... 1219
- defun bcPuiseuxSeriesGen .......................................................... 1219
- defun bcSeriesGen .................................................................... 1219
- defun Limit Basic Command ......................................................... 1220
- defun Real Limit Basic Command ................................................... 1221
- defun Real Limit Basic Command options ...................................... 1222
- defun bcRealLimitGen1 ............................................................... 1223
- defun Complex Limit Basic Command ............................................. 1223
- defun bcComplexLimitGen ........................................................... 1225
- defvar $systemType ..................................................................... 1225
- defvar $numberOfEquations ......................................................... 1226
- defvar $solutionMethod .............................................................. 1226
- defun Solve Basic Command ........................................................ 1226
- defun Linear Solve Basic Command .............................................. 1227
- defun Linear Solve Equations Basic Command .............................. 1228
- defun bcSystemSolve ................................................................. 1228
- defun bcSolveSingle ................................................................. 1229
- defun bcSystemSolveEqns1 ............................................................ 1229
- defun bcLinearSolveEqns1 ............................................................ 1230
- defun bcInputSolveInfo .............................................................. 1230
- defun bcInputEquations ............................................................. 1231
- defun Create a variable string ...................................................... 1233
- defun bcMakeUnknowns ............................................................... 1233
- defun bcMakeEquations .............................................................. 1234
- defun bcMakeLinearEquations ..................................................... 1234
- defun bcInputEquationsEnd ........................................................ 1235
- defun bcSolveEquationsNumerically ......................................... 1235
- defun bcSolveEquationsNumerically1 .......................................... 1236
- defun bcLinearSolveBasicCommand ............................................ 1236
- defun bcLinearSolveBasicCommand trampoline ............................ 1237
- defun Linear Solve Basic Command options ................................ 1237
- defun bcLinearExtractMatrix ....................................................... 1238
- defun bcLinearExtractMatrixInhomogen ...................................... 1238
- defun bcLinearSolveMatrixInhomogen .......................................... 1239
- defun bcLinearSolveMatrixHomo .................................................. 1240
- defun bcLinearMatrixGen ........................................................... 1240
- defun linearFinalRequest .......................................................... 1241
- defun explainLinear ................................................................. 1241
- defun finalExactRequest ............................................................ 1241
- defun bcLinearSolveEqnsGen ....................................................... 1242
- defun bcGenEquations ............................................................... 1242
- defun Output the final formula .................................................... 1243
<table>
<thead>
<tr>
<th>Function/Namespace</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun convert arguments into function call syntax</td>
<td>1243</td>
</tr>
<tr>
<td>defun bcString2HyString2</td>
<td>1244</td>
</tr>
<tr>
<td>defun bcString2HyString</td>
<td>1244</td>
</tr>
<tr>
<td>defun find a character position in a string</td>
<td>1244</td>
</tr>
<tr>
<td>defun Basic Command result page</td>
<td>1244</td>
</tr>
<tr>
<td>defun Basic Command result page – NAG version</td>
<td>1245</td>
</tr>
<tr>
<td>defun bcOptional</td>
<td>1245</td>
</tr>
<tr>
<td>defun create a vertical space on a page</td>
<td>1246</td>
</tr>
<tr>
<td>defun break a string into words</td>
<td>1246</td>
</tr>
<tr>
<td>defun format words into a string</td>
<td>1246</td>
</tr>
<tr>
<td>defun format a vector</td>
<td>1246</td>
</tr>
<tr>
<td>defun format an error message</td>
<td>1247</td>
</tr>
<tr>
<td>defun format intervals</td>
<td>1247</td>
</tr>
<tr>
<td>defun Basic Command page not ready</td>
<td>1247</td>
</tr>
<tr>
<td>defun pad a string with blanks</td>
<td>1248</td>
</tr>
<tr>
<td>defun construct a name string</td>
<td>1248</td>
</tr>
<tr>
<td>defun construct a name string</td>
<td>1248</td>
</tr>
<tr>
<td>defvar $bcParseOnly</td>
<td>1249</td>
</tr>
<tr>
<td>defvar $htLineList</td>
<td>1249</td>
</tr>
<tr>
<td>defvar $curpage</td>
<td>1249</td>
</tr>
<tr>
<td>defvar $activePageList</td>
<td>1249</td>
</tr>
<tr>
<td>defun htpDestroyPage</td>
<td>1250</td>
</tr>
<tr>
<td>defun htpName</td>
<td>1250</td>
</tr>
<tr>
<td>defun htpSetName</td>
<td>1250</td>
</tr>
<tr>
<td>defun htpDomainConditions</td>
<td>1250</td>
</tr>
<tr>
<td>defun htpSetDomainConditions</td>
<td>1251</td>
</tr>
<tr>
<td>defun htpDomainVariableAlist</td>
<td>1251</td>
</tr>
<tr>
<td>defun htpSetDomainVariableAlist</td>
<td>1251</td>
</tr>
<tr>
<td>defun htpDomainPvarSubstList</td>
<td>1251</td>
</tr>
<tr>
<td>defun htpSetDomainPvarSubstList</td>
<td>1251</td>
</tr>
<tr>
<td>defun htpRadioButtonAlist</td>
<td>1252</td>
</tr>
<tr>
<td>defun htpButtonValue</td>
<td>1252</td>
</tr>
<tr>
<td>defun htpSetRadioButtonAlist</td>
<td>1252</td>
</tr>
<tr>
<td>defun htpInputAreaAlist</td>
<td>1253</td>
</tr>
<tr>
<td>defun htpSetInputAreaAlist</td>
<td>1253</td>
</tr>
<tr>
<td>defun htpAddInputAreaProp</td>
<td>1253</td>
</tr>
<tr>
<td>defun htpPropertyList</td>
<td>1253</td>
</tr>
<tr>
<td>defun htpProperty</td>
<td>1254</td>
</tr>
<tr>
<td>defun htpSetProperty</td>
<td>1254</td>
</tr>
<tr>
<td>defun htpLabelInputString</td>
<td>1254</td>
</tr>
<tr>
<td>defun htpLabelFilteredInputString</td>
<td>1255</td>
</tr>
<tr>
<td>defun replacePercentByDollar_fn</td>
<td>1255</td>
</tr>
<tr>
<td>defun replacePercentByDollar</td>
<td>1256</td>
</tr>
<tr>
<td>defun htpSetLabelInputString</td>
<td>1256</td>
</tr>
<tr>
<td>defun htpLabelSpadValue</td>
<td>1256</td>
</tr>
<tr>
<td>defun htpSetLabelSpadValue</td>
<td>1257</td>
</tr>
</tbody>
</table>
defun htpLabelErrorMsg ........................................... 1257
defun htpSetLabelErrorMsg ......................................... 1257
defun htpLabelType .................................................... 1258
defun htpLabelDefault .................................................. 1258
defun htpLabelSpadType ................................................. 1258
defun htpLabelFilter .................................................... 1259
defun htpPageDescription .............................................. 1259
defun htpSetPageDescription ........................................... 1259
defun htpAddToPageDescription ........................................ 1260
defun issue a single hypertex line or group of lines ............... 1260
defun bcHt ................................................................ 1260
defun bcIssueHt ............................................................. 1261
defun mapStringize ......................................................... 1261
defun basicStringize ....................................................... 1261
defun stringize ............................................................... 1261
defun htInitPage ............................................................. 1262
defun htAddHeading ......................................................... 1262
defun htShowPage ............................................................. 1263
defun show the page which has been computed ....................... 1263
defun make a page given the description in itemList ................ 1263
defun htMakePage1 ........................................................... 1264
defun htMakeErrorPage ...................................................... 1265
defun htQuote ................................................................. 1265
defun htProcessToggleButtons ............................................ 1265
defun htProcessBcButtons .................................................. 1266
defun htProcessBcStrings ................................................ 1268
defun bcSadFaces ............................................................. 1269
defun htLispLinks ............................................................. 1269
defun htLispMemoLinks ..................................................... 1270
defun htBcLinks ............................................................... 1270
defun htBcLispLinks .......................................................... 1271
defun beforeAfter ............................................................ 1272
defun mkCurryFun ............................................................. 1272
defun htRadioButtons ........................................................ 1273
defun htBcRadioButtons ..................................................... 1274
defun setUpDefault .......................................................... 1275
defun buttonNames .......................................................... 1275
defun htInputStrings ........................................................ 1276
defun htProcessDomainConditions ....................................... 1276
defun renamePatternVariables .......................................... 1277
defun renamePatternVariables1 ......................................... 1278
defun substFromAlist ....................................................... 1278
defun computeDomainVariableAlist .................................... 1280
defun pvarCondList .......................................................... 1280
defun pvarCondList1 ......................................................... 1281
defun pvarsOfPattern ....................................................... 1281
defun htMakeTemplates, substLabel  ........................................ 1282
defun htMakeTemplates  .......................................................... 1283
defun templateParts  ........................................................................ 1284
defun htMakeDoneButton ............................................................. 1284
defun htProcessDoneButton ......................................................... 1285
defun htMakeButton ...................................................................... 1285
defun bchtMakeButton .................................................................. 1287
defun htProcessDoitButton ........................................................... 1288
defun htMakeDoitButton .............................................................. 1288
defun doDoitButton ....................................................................... 1289
defun executeInterpreterCommand ................................................. 1289
defun htDoneButton ...................................................................... 1289
defun typeCheckInputAreas .......................................................... 1290
defun checkCondition ................................................................... 1292
defun condErrorMsg ..................................................................... 1293
defun parseAndEval ....................................................................... 1294
defun parseAndEval1 ..................................................................... 1294
defun oldParseString ................................................................... 1295
defun makeSpadCommand ............................................................... 1295
defun htMakeInputList ................................................................. 1296
defun bracketString ..................................................................... 1296
defun quoteString ........................................................................ 1297
defvar $funnyQuote ...................................................................... 1297
defvar $funnyBacks ...................................................................... 1297
defun htEscapeString .................................................................... 1297
defun ltsv .................................................................................... 1298
defun htSetVars ........................................................................... 1298
defun htShowSetTree ..................................................................... 1298
defun htShowCount ....................................................................... 1300
defun htShowSetTreeValue ............................................................ 1301
defun mkSetTitle ........................................................................... 1302
defun listOfStrings2String ............................................................ 1302
defun htShowSetPage ..................................................................... 1302
defun htShowLiteralsPage .............................................................. 1303
defun htSetLiterals ....................................................................... 1303
defun htSetLiteral ........................................................................ 1304
defun htShowIntegerPage ............................................................... 1305
defun htSetInteger ......................................................................... 1306
defun htShowFunctionPage ............................................................ 1306
defun htShowFunctionPageContinued ............................................. 1307
defun htSetvarDoneButton ............................................................. 1308
defun htFunctionSetLiteral ............................................................ 1308
defun htSetFunCommand ............................................................... 1309
defun htSetFunCommandContinue ................................................. 1309
defun htKill .................................................................................. 1310
defun htSetNotAvailable .............................................................. 1310
defun mkUnixPattern ............................... 1341

75 Browser Support Code 1343
  75.1 Pages Initiated from HyperDoc Pages .......................... 1343
defun conPage .................................. 1343
defun gets line quickly for constructor name or abbreviation .... 1344
defun conPageConEntry ............................. 1344
defun kdPageInfo ................................ 1345
defun kArgPage ................................ 1346
defun reportCategory ................................ 1347
defun reportAO .................................. 1348
defun mkDomTypeForm ................................ 1348
defun domainDescendantsOf ......................... 1349

75.2 Branches of Constructor Page ............................. 1350
defun kiPage ...................................... 1350
defun kePage ...................................... 1351
defun kePageOpAlist ................................ 1353
defun kePageDisplay ................................ 1354
defun ksPage ...................................... 1355
defun dbSearchOrder ................................ 1356
defun kcPage ...................................... 1357
defun kcpPage ..................................... 1361
defun reduceAlistForDomain ......................... 1362
defun kcaPage ..................................... 1362
defun kcdoPage .................................... 1362
defun kcPage ...................................... 1363
defun kcaPage1 ................................... 1363
defun kccPage ..................................... 1364
defun augmentHasArgs ............................... 1365
defun kcdePage ................................... 1366
defun getDependentsOfConstructor .................... 1367
defun kcuPage ..................................... 1367
defun getUsersOfConstructor ......................... 1368
defun kcnPage ..................................... 1368
defun koPageInputAreaUnchanged? ..................... 1370
defun kDomainName ................................ 1370
defun kArgumentCheck .............................. 1371
defun dbMkEvalable ................................ 1372
defun topLevelInterpEval ......................... 1372
defun kisValidType ................................ 1372
defun kCheckArgumentNumbers ....................... 1373
defun parseNoMacroFromString ....................... 1373
defun mkConform .................................. 1374

75.3 Operation Page for a Domain Form from Scratch 1375
defun conOpPage .................................. 1375
defun conOpPage1 .................................. 1375
defun libConstructorSig ............................................. 1406

76 The Interpreter ..................................................... 1409

77 The Global Variables ............................................... 1449

77.1 Star Global Variables ............................................. 1449
*eof* ................................................................. 1449
*features* .......................................................... 1449
*package* .......................................................... 1449
*standard-input* .................................................... 1450
*standard-output* ................................................... 1450
*top-level-hook* ..................................................... 1450

77.2 Dollar Global Variables .......................................... 1452
$boot ................................................................. 1453
coerceFailure ....................................................... 1453
$currentLine ........................................................ 1453
$displayStartMsgs ................................................... 1453
$e ................................................................. 1453
$erMsgToss ........................................................ 1453
$fn ................................................................. 1453
$frameRecord ......................................................... 1453
$HiFiAccess ........................................................ 1454
$HistList ........................................................... 1454
$HistListAct ......................................................... 1454
$HistListLen ........................................................ 1454
$HistRecord .......................................................... 1454
$historyFileType ..................................................... 1455
$internalHistoryTable ............................................. 1455
$interpreterFrameName ............................................ 1455
$interpreterFrameRing ............................................ 1455
$InteractiveFrame .................................................. 1455
$intRestart .......................................................... 1455
$intTopLevel ......................................................... 1455
$IOindex ............................................................. 1456
$lastPos ............................................................. 1456
$libQuiet ............................................................. 1456
$msgDatabaseName .................................................. 1456
$ncMsgList .......................................................... 1456
$newcompErrorCount ............................................... 1456
$newspad ............................................................ 1456
$nopos ............................................................... 1456
$oldHistoryFileName ............................................... 1457
$okToExecuteMachineCode ......................................... 1457
$options ............................................................... 1457
$previousBindings .................................................... 1457
$PrintCompilerMessageIfTrue .................................... 1457
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 will 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

The Interpreter

The Axiom interpreter is a large common lisp program. It has several forms of interaction and run from terminal in a standalone fashion, run under the control of a session handler program, run as a web server, or run in a unix pipe.
Chapter 2

The Fundamental Data Structures

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

2.1 The global variables

Credits

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

\[\text{defvar $\text{creditlist}$}\]

\begin{verbatim}
  defvar creditlist '(
  "An alphabetical listing of contributors to AXIOM:"
  "Michael Albaugh  Cyril Alberga  Roy Adler"
  "Christian Aistleitner Richard Anderson George Andrews"
  "S.J. Atkins  Henry Baker  Martin Baker"
  "Stephen Balzac  Yuriy Baransky  David R. Barton"
  "Gerald Baumgartner  Gilbert Baumslag  Michael Becker"
  "Nelson H. F. Beebe  Jay Belanger  David Bindel"
  "Fred Blair  Vladimir Bondarenko  Mark Botch"
  )
\end{verbatim}
CHAPTER 2. THE FUNDAMENTAL DATA STRUCTURES

"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 Ericsson  Richard Fateman"
"Bertfried Fauser  Stuart Feldman  John Fletcher"
"Brian Ford  Albrecht Fortenbacher  George Frances"
"Constantine Frangos  Timothy Freeman  Korrihn Fu"
"Marc Gaetano  Rudiger Gebauer  Van de Geijn"
"Kathy Gerber  Patricia Gianni  Samantha Goldrich"
"Holger Gollan  Teresa Gomez-Diaz  Laureano Gonzalez-Vega"
"Stephen Gorlter  Johannes Grabmeier  Matt Grayson"
"Klaus Ebbe Grue  James Griesmer  Vladimir Gonzalez-Vega"
"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  Kai Kaminski"
"Grant Keedy  Wilfrid Kendall  Tony Kennedy"
"Ted Kosan  Paul Kosinski  Klaus Kusche"
"Bernhard Kutzler  Tim Lahey  Larry Lambe"
"Kaj Laurson  George L. Legendre  Franz Lehner"
"Frederic Lehobey  Michel Levaud  Howard Levy"
"Ren-Cang Li  Rudiger Loos  Michael Lucks"
"Richard Luczak  Camm Maguire  Francois Maltey"
"Alasdair McAndrew  Bob McElrath  Michael McGettrick"
"Edi Meier  Ian Meikle  David Mentre"
"Victor S. Miller  Gerard Milmeister  Mohammed Mobarak"
"H. Michael Moeller  Michael Monagan  Marc Moreno-Maza"
"Scott Morrison  Joel Moses  Mark Murray"
"William Naylor  Patrice Naudin  C. Andrew Neff"
"John Nelder  Godfrey Nolan  Arthur Norman"
"Jinzhong Niu  Michael O’Connor  Summat Oemrawsingh"
2.1. THE GLOBAL VARIABLES

"Kostas Oikonomou Humberto Ortiz-Zuazaga Julian A. Padget"
"Bill Page David Parnas Susan Pelzel"
"Michel Petitot Didier Pinchon Ayal Pinkus"
"Frederick H. Pitts Jose Alfredo Portes Gregorio Quintana-Orti"
"Claude Quitte Arthur C. Rals Norman Ramsey"
"Anatoly Raportirenko Albert D. Rich Michael Richardson"
"Guilherme Reis Huan Ren Renaud Rioboo"
"Jean Rivlin Nicolas Robidoux Simon Robinson"
"Raymond Rogers Michael Rothstein Martin Rubey"
"Philip Santas Alfred Scheerhorn William Schelter"
"Gerhard Schneider Martin Schoenert Marshall Schor"
"Frithjof Schulze Fritz Schwarz Steven Segletes"
"V. Sima Nick Simicich William Sit"
"Elena Smirnova Jonathan Steinbach Fabio Stumbo"
"Christine Sundaresan Robert Sutor Moss E. Sweedler"
"Eugene Surowitz Max Tegmark T. Doug Telford"
"James Thatcher Balbir Thomas Mike Thomas"
"Dylan Thurston Steve Toleque Barry Trager"
"Themos T. Tsikas Gregory Vanuxem Bernhard Wall"
"Stephen Watt Jaap Weel Juergen Weiss"
"M. Weller Mark Wegman James Wen"
"Thorsten Werther Michael Wester R. Clint Whaley"
"John M. Wiley Berhard Will Clifton J. Williamson"
"Stephen Wilson Shmuel Winograd Robert Wisbauer"
"Sandra Wityak Waldemar Wivianka Knut Wolf"
"Liu Xiaojun Clifford Yapp David Yun"
"Vadim Zhytnikov Richard Zippel Evelyn Zoernack"
"Bruno Zuercher Dan Zwillinger"

The $current-directory$ variable is set to the current directory at startup. This is used by the $cd$ function and some of the compile routines. This is the result of the (p34) get-current-directory function. This variable is used to set *default-pathname-defaults*. The (p38) reroot function resets it to $spadroot$.

An example of a runtime value is:

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

\begin{verbatim}
(defvar $current-directory nil)
\end{verbatim}
The $defaultMsgDatabaseName variable contains the location of the international message database. This can be changed to use a translated version of the messages. It defaults to the United States English version. The relative pathname used as the default is hardcoded in the (p38) reroot function. This value is prefixed with the $spadroot to make the path absolute.

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

An example of a runtime value is:

```lisp
(defun |$defaultMsgDatabaseName| ()
  (concatenate 'string #p"/research/test/mnt/ubuntu/doc/msgs/s2-us.msgs")
)
```

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

The $directory-list is a runtime list of absolute pathnames. This list is generated by (p38) reroot from the list of relative paths held in the variable $relative-directory-list. Each entry will be prefixed by $spadroot.

An example of a runtime value is:

```lisp
(defun $directory-list ()
  (list
   (concatenate 'string "/research/test/mnt/ubuntu/../../../src/input/"
   (concatenate 'string "/research/test/mnt/ubuntu/../../../src/msg/"
   (concatenate 'string "/research/test/mnt/ubuntu/../../../src/algebra/"
   (concatenate 'string "/research/test/mnt/ubuntu/../../../src/interp/"
   (concatenate 'string "/research/test/mnt/ubuntu/doc/spadhelp/"))))
)
```

```
(defvar $directory-list nil)
```
2.1. THE GLOBAL VARIABLES

The $InitialModemapFrame is used as the initial value. See the function “makeInitialModemapFrame” (4.3 p 35).
An example of a runtime value is:

$InitialModemapFrame = '((nil))

defvar $InitialModemapFrame

   — initvars —

(defvar $InitialModemapFrame '((nil)))

The $library-directory-list variable is the system-wide search path for library files. (p38)
reroot prepends the $spadroot variable to the $relative-library-directory-list variable.
An example of a runtime value is:

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

defvar $library-directory-list

   — initvars —

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

The $msgDatabaseName is a locally shared variable among the message database routines.
An example of a runtime value is:

|$msgDatabaseName| = nil

defvar $msgDatabaseName

   — initvars —
The \$openServerIfTrue variable appears to control whether the interpreter will be used as an open server, probably for OpenMath use.
If an open server is not requested then this variable to NIL.
See the function ‘openserver’ (63 p 987).
An example of a runtime value is:

\$openServerIfTrue = nil

defvar \$openServerIfTrue

—— initvars ——

(defun \$openServerIfTrue nil)

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

This list is used by the (p38) reroot function to generate the absolute list of paths held in the variable \$directory-list. Each entry will be prefixed by \$spadroot.
An example of a runtime value is:

\$relative-directory-list =

("/.../src/input/
"/doc/msgn/
"/.../src/algebra/
"/.../src/interp/
"/doc/spadhelp")

defvar \$relative-directory-list

—— initvars ——

(defun \$relative-directory-list nil)
The $relative-library-directory-list is a hand-generated list of directories containing algebra. The (p38) reroot function will prefix every path in this list with the value of the $spadroot variable to construct the $library-directory-list variable.

An example of a runtime value is:

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

defvar $relative-library-directory-list

— initvars —

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

The $spadroot variable is the internal name for the AXIOM shell variable. It is set in reroot to the value of the argument. The value is expected to be a directory name. The (p33) initroot function uses this variable if the AXIOM shell variable is not set. The (p35) make-absolute-filename function uses this path as a prefix to all of the relative filenames to make them absolute.

An example of a runtime value is:

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

defvar $spadroot

— initvars —

(defvar $spadroot nil)

The $SpadServer determines whether Axiom acts as a remote server.
See the function “openserver” (63 p 987).
An example of a runtime value is:

$SpadServer = nil

defvar $SpadServer

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

The $SpadServerName defines the name of the spad server socket. In unix these exist in the tmp directory as names.
See the function “openserver” (63 p 987).
An example of a runtime value is:

$SpadServerName = "/tmp/.d"

defvar $SpadServerName

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

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

$IOindex = 1

defvar $IOindex

| initvars |

(defvar $IOindex 1 "The current Axiom prompt number")
Chapter 3
Starting Axiom

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

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

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

### 3.1 Variables Used

### 3.2 Data Structures

### 3.3 Functions

Set the restart hook

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

— defun set-restart-hook 0 —

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

restart function (The restart function)

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

The `compiler::*compile-verbose*` flag has been set to nil globally. We do not want to know about the microsteps of GCL’s compile facility.

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

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

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

The “initroot” (4.3 p 33) function sets global variables that depend on the AXIOM shell variable. These are needed to find basic files like `s2-us.msgs`, which contains the error message text.

The “openserver” (63 p 987) function tried to set up the socket connection used for things like hyperdoc. The `$openServerIfTrue` variable starts `true`, which implies trying to start a server.

Axiom has multiple frames that contain independent information about a computation. There can be several frames at any one time and you can shift back and forth between the frames. By default, the system starts in “frame0” (try the `frame names` command). See the Frame Mechanism chapter (32.3 page 552).

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

The “printLoadMsgs” (45.29 p 736) variable controls whether load messages will be output as library routines are loaded. We disnable this by default. It can be changed by using `set message autoload`.

The “current-directory” (2.1 p 5) variable is set to the current directory. This is used by the `cd` function and some of the compile routines.

The “statisticsInitialization” (67 p 1035) function initializes variables used to collect statistics. Currently, only the garbage collector information is initialized.
"--- defun restart ---"

(defun restart ()
  (declare (special $openServerIfTrue $SpadServerName |$SpadServer| |
  $IOindex| |$InteractiveFrame| |$printLoadMsgs| $current-directory |
  $displayStartMsgs| |$currentLine|))
  +#+:akcl
  (init-memory-config :cons 1024 :fixnum 200 :symbol 500 :package 8 |
  +#+:akcl (setq compiler::*compile-verbose* nil)
  +#+:akcl (setq compiler::*suppress-compiler-warnings* t)
  +#+:akcl (setq compiler::*suppress-compiler-notes* t)
  +#+:akcl (setq si::*system-directory* "")
  (in-package "BOOT")
  (initroot)
  +#+:akcl
  (when (and $openServerIfTrue (zerop (openserver $SpadServerName))))
    (setq $openServerIfTrue nil)
    (setq $SpadServer| t))
  (setq $IOindex| 1)
  (setq $InteractiveFrame| (makeInitialModemapFrame))
  (setq $printLoadMsgs| nil)
  (setq $current-directory (get-current-directory))
  (setq *default-pathname-defaults* (pathname $current-directory))
  (statisticsInitialization)
  (initHist)
  (initializeInterpreterFrameRing)
  (when $displayStartMsgs| (ispadStartUpMsgs))
  (setq $currentLine| nil)
  (restart0)
  (readSpadProfileIfThere)
  (ispad))

---

defun Non-interactive restarts

[interpopen p??]
[operationopen p??]
[categoryopen p??]
3.3. FUNCTIONS

[browseopen p??]
[getEnv p??]

— defun restart0 —

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

defun The startup banner messages

[fillerSpaces p18]
[specialChar p980]
[sayKeyedMsg p329]
[sayMSG p331]
[$msgAlist p326]
[$opSysName p??]
[$linelength p774]
[*yearweek* p??]
[*build-version* p??]

— defun spadStartUpMsgs —

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

(ifcar p??)

— defun fillerSpaces —

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

Starts the interpreter but do not read in profiles

(setOutputAlgebra p763)
(runspad p19)
[$PrintCompilerMessageIfTrue p??]

— defun spad —

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

________

defvar $quitTag

— initvars —

(defvar $quitTag system::*quit-tag*)
defun runspad

[quitTag p18]
[coerceFailure p??]
[top-level p??]
[seq p??]
[exit p??]
[resetStackLimits p19]
[ncTopLevel p23]
[$quitTag p18]

— defun runspad —

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

________

defun Reset the stack limits

[reset-stack-limits p??]

— defun resetStackLimits 0 —

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

________
Chapter 4

Handling Terminal Input

4.1 Streams

defvar $curinstream

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

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

defvar $curoutstream

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

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

defvar $errorinstream

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

defvar $erroroutstream

— initvars —

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

defvar $*eof*

— initvars —

(defvar *eof* nil)

defvar $*whitespace*

— initvars —

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

defvar $InteractiveMode

— initvars —

(defvar $InteractiveMode t)
4.1. STREAMS

```lisp
(defvar $boot
  ;; initvars
  (defvar $boot nil))
```

---

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

Top-level read-parse-eval-print loop for the interpreter. Uses the Bill Burge's parser. [ncInt-Loop p23]

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

---

**defun ncIntLoop**

[ntloop p24]
[curinstream p21]
(defun |ncIntLoop| ()
  (let ((curinstream *standard-output*)
       (curoutstream *standard-input*))
    (declare (special curinstream curoutstream))
    (|intloop|)))

defvar $intTopLevel

— initvars —

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

defvar $intRestart

— initvars —

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

defun intloop

Note that the SpadInterpretStream function uses a list of three strings as an argument. The
values in the list seem to have no use and can eventually be removed. [intTopLevel p24] [SpadInterpretStream p25]
[resetStackLimits p19]
[$intTopLevel p24]
[$intRestart p24]

— defun intloop —

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

defvar $intTopLevel

— initvars —

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

defvar $intRestart

— initvars —

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

defun intloop

Note that the SpadInterpretStream function uses a list of three strings as an argument. The
values in the list seem to have no use and can eventually be removed. [intTopLevel p24] [SpadInterpretStream p25]
[resetStackLimits p19]
[$intTopLevel p24]
[$intRestart p24]

— defun intloop —

(defun |intloop| ()
  (defun |intloop| ()
    (let ((curinstream *standard-output*)
      (curoutstream *standard-input*))
      (declare (special curinstream curoutstream))
      (|intloop|)))
4.1. STREAMS

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

---

defvar $ncMsgList

— initvars —

(defvar |$ncMsgList| nil)

---

defun SpadInterpretStream

The SpadInterpretStream function takes three arguments

  str This is passed as an argument to intloopReadConsole

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

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

System commands are handled by the function in the “hook” variable $systemCommandFunction
which has the default function InterpExecuteSpadSystemCommand. Thus, when a system com-
mand is entered this function is called.

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

— initvars —

(defvar $promptMsg 'S2CTP023)

defun GCL cmpnote function

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

— defun cmpnote —

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

defvar $newcompErrorCount

— initvars —

(defvar $newcompErrorCount 0)

defvar $nopos

— initvars —

(defvar $nopos (list '|noposition|))
4.1. STREAMS

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

|— defun SpadInterpretStream —|
4.2 The Read-Eval-Print Loop

defun intloopReadConsole

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

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

We read a line from standard input.

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

Notice that all but two paths (a null input or a “)fi” or a “)fin”) will end up as a recursive
call to ourselves. [top-level p??]

[serverReadLine p42]
[leaveScratchpad p639]
[mkprompt p40]
[intloopReadConsole p28]
[intloopPrefix? p34]
[intnplisp p34]
[setCurrentLine p40]
[ncloopCommand p478]
[concat p1047]
[ncloopEscaped p35]
[nintloopProcessString p36]
[$dalymode p663]
4.3 Helper Functions

Get the value of an environment variable

```
(getenv p??)
```

--- defun getenviron ---

(defun getenviron (var)
  "Get the value of an environment variable"
  #+allegro (sys::getenv (string var))
  #+clisp (ext:getenv (string var))
  #+(or cmu scl)
  (cdr)
CHAPTER 4. HANDLING TERMINAL INPUT

```lisp
(assoc (string var) ext:*environment-list* :test #'equalp :key #'string))
#+(or kcl akcl gcl) (si::getenv (string var))
#+lispworks (lw:environment-variable (string var))
#+lucid (lcl:environment-variable (string var))
#+mcl (ccl::getenv var)
#+sbcl (sb-ext:posix-getenv var)
)

defvar $intCoerceFailure

— initvars —
(defvar |$intCoerceFailure| 'coerceFailure)

defvar $intSpadReader

— initvars —
(defvar |$intSpadReader| 'SPAD_READER)

defun InterpExecuteSpadSystemCommand

[interCoerceFailure p30]
[interSpadReader p30]
[ExecuteInterpSystemCommand p31]
|$intSpadReader p30|
|$intCoerceFailure p30|

— defun InterpExecuteSpadSystemCommand —

(defun |InterpExecuteSpadSystemCommand| (string)
  (declare (special |$intSpadReader| |$intCoerceFailure|))
  (catch |$intCoerceFailure|
    (catch |$intSpadReader|
      (|ExecuteInterpSystemCommand| string))))
```
4.3. HELPER FUNCTIONS

---

defun ExecuteInterpSystemCommand
[intProcessSynonyms p31]
[substring p??]
[doSystemCommand p446]
[$currentLine p??]

---

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

---

defun Handle Synonyms
[processSynonyms p31]
[line p??]

---

(defun intProcessSynonyms
  (str)
  (let ((line str))
    (declare (special line))
    ([processSynonyms]
      line))

---

defun Synonym File Reader
[strpos p1045]
[substring p??]
[string2id-n p??]
[lassoc p??]
[strconc p??]
defun processSynonyms ()
(let (fill p aline synstr syn to opt fun cl chr)
  (declare (special |$CommandSynonymAlist| line))
  (setq p (strpos ")" line 0 nil))
  (setq fill "")
  (cond
    (p
      (setq aline (substring line p nil))
      (when (> p 0) (setq fill (substring line 0 p))))
    (t
      (setq p 0)
      (setq aline line)))
  (setq to (strpos " " aline 1 nil))
  (cond (to (setq to (1- to))))
  (setq synstr (substring aline 1 to))
  (setq syn (string2id-n synstr 1))
  (when (setq fun (lassoc syn |$CommandSynonymAlist|))
    (setq to (strpos ")" fun 1 nil))
    (cond
      ((and to (not (eql to (1- (size fun)))))
        (setq opt (strconc " " (substring fun to nil)))
        (setq fun (substring fun 0 (1- to ))))
      (t (setq opt " ")))
    (when (> (size synstr) (size fun))
      (do ((G167173 (size synstr)) (i (size fun) (1+ i)))
        ((> i G167173) nil)
        (setq fun (conc fun " "))))
    (setq cl (strconc fill (rplacstr aline 1 (size synstr) fun) opt))
    (setq line cl)
    (setq chr (elt line (1+ p)))
    (processSynonyms)))

defun init-memory-config

Austin-Kyoto Common Lisp (AKCL), now known as Gnu Common Lisp (GCL) requires
some changes to the default memory setup to run Axiom efficiently. This function performs
4.3. HELPER FUNCTIONS

those setup commands. [allocate p??]
[allocate-contiguous-pages p??]
[allocate-relocatable-pages p??]
[set-hole-size p??]

— defun init-memory-config 0 —

(defun init-memory-config (&key
  (cons 500)
  (fixnum 200)
  (symbol 500)
  (package 8)
  (array 400)
  (string 500)
  (cfun 100)
  (cpages 3000)
  (rpages 1000)
  (hole 2000))

;; initialize AKCL memory allocation parameters
#+:AKCL
(progn
  (system:allocate 'cons cons)
  (system:allocate 'fixnum fixnum)
  (system:allocate 'symbol symbol)
  (system:allocate 'package package)
  (system:allocate 'array array)
  (system:allocate 'string string)
  (system:allocate 'cfun cfun)
  (system:allocate-contiguous-pages cpages)
  (system:allocate-relocatable-pages rpages)
  (system:set-hole-size hole))
#-:AKCL
nil)

Set spadroot to be the AXIOM shell variable

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

[reroot p38]
[getenviron p29]
[$spadroot p9]

— defun initroot —

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

Does the string start with this prefix?

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

— defun intloopPrefix? 0 —

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

defun Interpret a line of lisp code

This is used to handle )lisp top level commands [nplisp p472]
[$currentLine p72]

— defun intnplisp —

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

Get the current directory

— defun get-current-directory 0 —

(defun get-current-directory ()
  "Get the current directory"
4.3. HELPER FUNCTIONS

(namestring (truename ""))

Prepend the absolute path to a filename

Prefix a filename with the AXIOM shell variable. \(\$\text{spadroot} p9\)

— defun make-absolute-filename 0 —

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

Make the initial modemap frame

[copy p?]
[\$InitialModemapFrame p7]

— defun makeInitialModemapFrame 0 —

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

defun ncloopEscaped

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

— defun ncloopEscaped 0 —

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

(setCurrentLine p40)
(intloopProcess p62)
(next p36)
(incString p37)

— defun intloopProcessString —

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

defun ncloopParse

(ncloopDQlines p70)
(npParse p141)
(dqToList p344)

— defun ncloopParse —

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

—

defun next

(Delay p102)
(next1 p37)

— defun next —
4.3. HELPER FUNCTIONS

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

---

defun next1

(StreamNull p333)
[incAppend p85]
[next p36]

— defun next1 —

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

---

defun incString

[incRenumber p72]
[incLude p75]
[Top p75]

— defun incString —

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

—

Call the garbage collector

Call the garbage collector on various platforms.

— defun reclaim 0 —
defun reroot

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

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

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

For the example call:

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

the variables are set as:

```
$spadroot = "/research/test/mnt/ubuntu"
```
4.3. HELPER FUNCTIONS

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

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

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

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

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

|$msgDatabaseName| = nil

$current-directory = "/research/test/

(make-absolute-filename p35)

$spadroot p9

$directory-list p6

$relative-directory-list p8

$library-directory-list p7

$relative-library-directory-list p9

|$defaultMsgDatabaseName| p6

|$msgDatabaseName| p7

$current-directory p5

--- defun reroot ---

(defun reroot (dir)
  (declare (special $spadroot $directory-list $relative-directory-list
                   $library-directory-list $relative-library-directory-list
                   |$defaultMsgDatabaseName| |$msgDatabaseName| $current-directory))

  (setq $spadroot dir)
  (setq $directory-list)
  (mapcar #'make-absolute-filename $relative-directory-list)
  (setq $library-directory-list)
  (mapcar #'make-absolute-filename $relative-library-directory-list)
  (setq |$defaultMsgDatabaseName|)
    (pathname (make-absolute-filename "/msgs/s2-us.msgs"))
  (setq |$msgDatabaseName| ()


(setq $current-directory $spadroot))

---

defun setCurrentLine

Remember the current line. The cases are:

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

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

—— defun setCurrentLine 0 ——

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

---

Show the Axiom prompt

[concat p1047]
[substring p??]
[currenttime p??]
[InputPromptType p746]
[IOindex p10]
[InterpreterFrameName p??]

—— defun mkprompt ——
4.3. HELPER FUNCTIONS

(require 'cl)

(defun mkprompt ()
  "Show the Axiom prompt"
  (declare (special |$inputPromptType| |$IOindex| |$interpreterFrameName|))
  (case |$inputPromptType|
    (|none| "")
    (|plain| "-> ")
    (|step| (concat "(" (princ-to-string |$IOindex|) ") -> ")
    (|frame| (concat (princ-to-string |$interpreterFrameName|) " "
                 (princ-to-string |$IOindex|) ") -> ")
    (t (concat (princ-to-string |$interpreterFrameName|) " [" 
         (substring (currenttime) 8 nil) "] [" 
         (princ-to-string |$IOindex|) "] -> "))))

----------

defvar $frameAlist

— initvars —

(defvar |$frameAlist| nil)

----------

defvar $frameNumber

— initvars —

(defvar |$frameNumber| 0)

----------

defvar $currentFrameNum

— initvars —

(defvar |$currentFrameNum| 0)
defvar $EndServerSession

    — initvars —

    (defvar $EndServerSession nil)

defvar $NeedToSignalSessionManager

    — initvars —

    (defvar $NeedToSignalSessionManager nil)

defvar $sockBufferLength

    — initvars —

    (defvar $sockBufferLength 9217)

READ-LINE in an Axiom server system

[coerceFailure p??]
[top-level p??]
[spad-reader p??]
[read-line p??]
[addNewInterpreterFrame p561]
[sockSendInt p??]
[sockSendString p??]
[mkprompt p40]
[sockGetInt p??]
[lassoc p??]
[changeToNamedInterpreterFrame p560]
[sockGetString p??]
[unescapeStringsInForm p61]
4.3. HELPER FUNCTIONS

```
[protectedEVAL p45]
[executeQuietCommand p45]
[parseAndInterpret p46]
[serverReadLine is-console (vol9)]
[serverSwitch p??]
[$KillLispSystem p??]
[$NonSmanSession p??]
[$SpadCommand p??]
[$QuietSpadCommand p??]
[$MenuServer p??]
[$sockBufferLength p42]
[$LispCommand p??]
[$EndServerSession p42]
[$EndSession p??]
[$SwitchFrames p??]
[$CreateFrameAnswer p??]
[$currentFrameNum p41]
[$frameNumber p41]
[$frameAlist p41]
[$CreateFrame p??]
[$CallInterp p??]
[$EndOfOutput p??]
[$SessionManager p??]
[$NeedToSignalSessionManager p42]
[$EndServerSession p42]
[$SpadServer p10]
[*eof* p22]
in-stream p961]
```

```lisp
(defun serverReadLine |serverReadLine| (stream)
  "used in place of READ-LINE in a Axiom server system."
  (let (in-stream *eof* 1 framename currentframe form stringbuf line action)
    (declare (special in-stream *eof* |$SpadServer| |$EndServerSession|
       |$NeedToSignalSessionManager| |$SessionManager| |$EndOfOutput|
       |$CallInterp| |$CreateFrame| |$frameAlist| |$frameNumber|
       |$currentFrameNum| |$CreateFrameAnswer| |$SwitchFrames| |$EndSession|
       |$EndServerSession| |$LispCommand| |$sockBufferLength| |$MenuServer|
       |$QuietSpadCommand| |$SpadCommand| |$NonSmanSession| |$KillLispSystem|)
    (force-output)
    (if (or (null |$SpadServer|) (null (is-console stream)))
      ([read-line] stream)
    (progn
      (setq in-stream stream)
      (setq *eof* nil)
      (setq line
```
CHAPTER 4. HANDLING TERMINAL INPUT

(do ()
  (null (and (null |$EndServerSession|) (null *eof*))) nil)
  (when |$NeedToSignalSessionManager| |
  (sockSendInt| |$SessionManager| |$EndOfOutput|))
  (setq |$NeedToSignalSessionManager| nil)
  (setq action (serverSwitch))
  (cond
    ((= action |$CallInterp|)
      (setq l (read-line stream))
      (setq |$NeedToSignalSessionManager| t)
      (return l))
    ((= action |$CreateFrame|)
      (setq framename (gentemp "frame")
        (addNewInterpreterFrame framename)
        (setq |$frameAlist|
          (cons (cons |$frameNumber| framename) |$frameAlist|))
      (sockSendInt |$SessionManager| |$CreateFrameAnswer|)
      (sockSendInt |$SessionManager| |$frameNumber|)
      (setq |$frameNumber| (1+ |$frameNumber|))
      (sockSendString |$SessionManager| (mkprompt)))
    ((= action |$SwitchFrames|)
      (setq |$frameAlist|
        (sockGetInt |$SessionManager|)
        (assoc |$currentFrameNum| |$frameAlist|)
        (changeToNamedInterpreterFrame currentframe))
    ((= action |$EndSession|)
      (setq |$EndServerSession| t))
    ((= action |$LispCommand|)
      (setq |$NeedToSignalSessionManager| t)
      (sockGetString |$MenuServer| stringbuf |$sockBufferLength|)
      (parseAndInterpret stringbuf))
    ((= action |$QuietSpadCommand|)
      (executeQuietCommand))
    ((= action |$SpadCommand|)
      (sockGetString |$MenuServer| stringbuf 512)
      (catch 'spad_reader
        (parseAndInterpret stringbuf)))
    ((= action |$NonSmanSession|) (setq |$SpadServer| nil))
    ((= action |$KillLispSystem|) (bye))
    (t nil)))
  (cond
4.3. HELPER FUNCTIONS

(defun protectedEVAL
  (resetStackLimits p19)
  (sendHTErrorSignal p??)

  — defun protectedEVAL —

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

— initvars —

(defun executeQuietCommand
  (When $QuietCommand is true Spad will not produce any output from a top level command)

  [spad-reader p??]
  [coerceFailure p??]
  [toplevel p??]
  [spadreader p??]
  [make-string p??]
defun executeQuietCommand

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

defun parseAndInterpret

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

defun parseFromString

(defun parseFromString (str)
  (let (#$InteractiveMode $boot $spad $e)
    (declare (special #$InteractiveMode $boot $spad $e)
      #$InteractiveFrame))
    (setq #$InteractiveMode t)
    (setq $boot nil)
    (setq $spad t)
    (setq $e $InteractiveFrame)
    (|processInteractive| (|parseFromString| str) nil)))
4.3. HELPER FUNCTIONS

(defun parseFromString (s)
  (setq s (next (ncloopParse (next (lineoftoks (incString s))))))
  (unless (StreamNull s) (pf2Sex (macroExpanded (cadar s)))))

——

defvar $interpOnly

  — initvars —

  (defvar $interpOnly nil)

——

defvar $minivectorNames

  — initvars —

  (defvar $minivectorNames nil)

——

defvar $domPvar

  — initvars —

  (defvar $domPvar nil)

——
defun processInteractive

Parser Output -->> Interpreter

Top-level dispatcher for the interpreter. It sets local variables and then calls processInteractive1 to do most of the work. This function receives the output from the parser. [initializeTimedNames p??]
[setq p??]
[processInteractive1 p50]
[reportInstantiations p742]
[clrhash p??]
[writeHistModesAndValues p603]
[updateHist p589]
[$op p??]
[$Coerce p??]
[$compErrorMessageStack p??]
[$freeVars p??]
[$mapList p??]
[$compilingMap p??]
[$compilingLoop p??]
[$interpOnly p47]
[$whereCacheList p??]
[$timeGlobalName p??]
[$StreamFrame p??]
[$declaredMode p??]
[$localVars p??]
[$analyzingMapList p??]
[$lastLineInSEQ p??]
[$instantCoerceCount p??]
[$instantCanCoerceCount p??]
[$instantMmCondCount p??]
[$fortVar p??]
[$minivector p??]
[$minivectorCode p??]
[$minivectorNames p47]
[$domPvar p47]
[$inRetract p??]
[$instantRecord p??]
[$reportInstantiations p742]
[$ProcessInteractiveValue p50]
[$defaultFortVar p??]
[$interpreterTimedNames p??]
[$interpreterTimedClasses p??]

— defun processInteractive —

(defun |processInteractive| (form posnForm)
4.3. HELPER FUNCTIONS

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

(declare (special |$op| |$Coerce| |$compErrorMessageStack|
    |$freeVars| |$mapList| |$compilingMap|
    |$compilingLoop| |$interpOnly| |$whereCacheList|
    |$timeGlobalName| |$StreamFrame| |$declaredMode|
    |$localVars| |$analyzingMapList| |$lastLineInSEQ|
    |$instantCoerceCount| |$instantCanCoerceCount|
    |$instantMmCondCount| |$fortVar| |$minivector|
    |$minivectorCode| |$minivectorNames| |$domPvar|
    |$inRetract| |$instantRecord| |$reportInstantiations|
    |$ProcessInteractiveValue| |$defaultFortVar|
    |$interpreterTimedNames| |$interpreterTimedClasses|))

(|initializeTimedNames| |$interpreterTimedNames| |$interpreterTimedClasses|)

(if (consp form) ; compute name of operator
    (setq |$op| (qcar form))
    (setq |$op| form))

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

-----

defvar $ProcessInteractiveValue

— initvars —

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

-----

defvar $HTCompanionWindowID

— initvars —

(defvar |$HTCompanionWindowID| nil)

-----

defun processInteractive1

This calls the analysis and output printing routines [recordFrame p923]
[startTimingProcess p??]
[interpretTopLevel p51]
[stopTimingProcess p??]
[recordAndPrint p54]
[objValUnwrap p??]
[objMode p??]
[&c p??]
[$ProcessInteractiveValue p50]
[$InteractiveFrame p??]

— defun processInteractive1 —
4.3. HELPER FUNCTIONS

(defun processInteractive1 (form posnForm)
  (let ([e object])
    (declare (special e ProcessInteractiveValue InteractiveFrame))
    (setq e InteractiveFrame)
    (recordFrame 'system)
    (startTimingProcess 'analysis)
    (setq object (interpretTopLevel form posnForm))
    (stopTimingProcess 'analysis)
    (startTimingProcess 'print)
    (unless ProcessInteractiveValue
      (recordAndPrint (objValUnwrap object) (objMode object)))
    (recordFrame 'normal)
    (stopTimingProcess 'print)
    object))

---

defun interpretTopLevel

[interpreter p51]
[interpret p52]
[stopTimingProcess p51]
[peekTimedName p51]
[interpretTopLevel p51]
[timedNameStack p51]

  defun interpretTopLevel —

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

---

defvar $genValue

If the $genValue variable is true then evaluate generated code, otherwise leave code unevaluated. If $genValue is false then we are compiling. This variable is only defined and used
CHAPTER 4. HANDLING TERMINAL INPUT

locally.

— initvars —

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

defun Type analyzes and evaluates expression x, returns object

[interpret1 p52]

[$env p??]

[$eval p??]

[$genValue p51]

— defun interpret —

(defun interpret | (&&rest arg &aux restargs x)

(let ((|$env| |$eval| |$genValue| posnForm))

(declare (special |$env| |$eval| |$genValue|)))

(setq x (car arg))

(setq restargs (cdr arg))

(if (consp restargs)

(setq posnForm (car restargs))

(setq posnForm restargs))

(setq |$env| (list (list nil)))

(setq |$eval| t) ; generate code -- don't just type analyze

(setq |$genValue| t) ; evaluate all generated code

(|interpret1| x nil posnForm)))

defun Dispatcher for the type analysis routines

This is the dispatcher for the type analysis routines. It type analyzes and evaluates the expression x in the rootMode (if non-nil) which may be $EmptyMode. It returns an object if evaluating, and a modeset otherwise. It creates the attributed tree. [mkAtreeWithSrcPos p??]

[putTarget p??]

[bottomUp p??]

[getArgValue p??]

[objNew p??]

[getValue p??]

[interpret2 p53]

[keyedSystemError p??]
4.3. HELPER FUNCTIONS

---

**defun interpret1**

(defun interpret1 (x rootMode posnForm)
  (let (node modeSet newRootMode argVal val)
    (declare (special $genValue |$eval|))
    (setq node (mkAAtreeWithSrcPos x posnForm))
    (when rootMode (putTarget node rootMode))
    (setq modeSet (bottomUp node))
    (if (null |$eval|)
      (progn
        (if (null rootMode)
          (setq newRootMode (car modeSet))
          (setq newRootMode rootMode))
        (setq argVal (getArgValue node newRootMode))
        (cond
          ((and argVal (null |$genValue|))
            (objNew argVal newRootMode))
          ((and argVal (setq val (getValue node)))
            (interpret2 val newRootMode posnForm))
          (t
            (keyedSystemError 'S2IS0053 (list x))))))))

---

**defun interpret2**

This is the late interpretCoerce. I removed the call to coerceInteractive, so it only does the JENKS cases ALBI [objVal p??]

[object p??]

[member p1048]

[objNew p??]

[systemErrorHere p??]

[coerceInteractive p??]

[throwKeyedMsgCannotCoerceWithValue p??]

[$EmptyMode p??]

[$ThrowAwayMode p??]

---

**defun interpret2**

(defun interpret2 (object m1 posnForm)
  (declare (ignore posnForm))
  (let (x m op ans)
(declare (special |$EmptyMode| |$ThrowAwayMode|))
(cond ((equal m1 |$ThrowAwayMode|) object)
  (t
   (setq x (|objVal| object))
   (setq m (|objMode| object))
   (cond ((equal m |$EmptyMode|)
           (cond ((and (consp x)
                      (progn (setq op (qcar x)) t)
                      (|member| op '(map stream)))
                  (|objNew| x m1))
               ((equal m1 |$EmptyMode|)
                (|objNew| x m))
               (t
                (|systemErrorHere| "interpret2")))))
  (m1
   (if (setq ans (|coerceInteractive| object m1))
       ans
       (|throwKeyedMsgCannotCoerceWithValue| x m m1)))
  (t object)))))

---

defun Result Output Printing

Prints out the value x which is of type m, and records the changes in environment $e$ into
$InteractiveFrame$ $printAnyIfTrue$ is documented in setvart.boot. It is controlled with the
$\texttt{se me any}$ command. [\texttt{output p??}]

[\texttt{putHist p590}]
[\texttt{objNewWrap p??}]
[\texttt{printTypeAndTime p56}]
[\texttt{printStorage p56}]
[\texttt{printStatisticsSummary p56}]
[\texttt{mkCompanionPage p??}]
[\texttt{recordAndPrintTest p??}]
[\texttt{$outputMode p??}]
[\texttt{$mkTestOutputType p??}]
[\texttt{$runTestFlag p??}]
[\texttt{$e p??}]
[\texttt{$mkTestFlag p??}]
[\texttt{$HTCompanionWindowID p50}]
[\texttt{$QuietCommand p45}]
[\texttt{$printStatisticsSummaryIfTrue p749}]
[\texttt{$printTypeIfTrue p752}]


4.3. HELPER FUNCTIONS

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

— defun recordAndPrint —
defun printStatisticsSummary

[sayKeyedMsg p329]
[statisticsSummary p??]
[$collectOutput p??]

— defun printStatisticsSummary —

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

—

defun printStorage

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

— defun printStorage —

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

—

defun printTypeAndTime

[printTypeAndTimeSaturn p58]
[printTypeAndTimeNormal p57]
[$saturn p??]

— defun printTypeAndTime —
4.3. HELPER FUNCTIONS

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

(defun printTypeAndTimeNormal\(x \ m\))
  (let (xp mp timeString result)
    (declare (special \$outputLines\) \$collectOutput\) \$printTypeIfTrue\)
    (cond ((and (consp m) (eq (qcar m) '|Union|))
      (setq xp (retract (objNewWrap x m)))
      (setq mp (objMode xp))
      (setq m
        (cons '|Union|
          (append
            (dolist (arg (qcdr m) (nreverse result))
              (when (sameUnionBranch arg mp) (push arg result)))))
            (list "...")))))))
    (when \$printTimeIfTrue\)
(setq timeString
     (|makeLongTimeString|
      |$interpreterTimedNames|
      |$interpreterTimedClasses|)))
(cond
  ((and |$printTimeIfTrue| |$printTypeIfTrue|
       (if |$collectOutput|
           (push (|msgText| 'S2GL0012 (list m)) |$outputLines|)
           (|sayKeyedMsg| 'S2GL0014 (list m timeString ))))
   ($printTimeIfTrue
    (unless |$collectOutput| (|sayKeyedMsg| 'S2GL0013 (list timeString))))
   ($printTypeIfTrue
    (if |$collectOutput|
        (push (|justifyMyType| (|msgText| 'S2GL0012 (list m))) |$outputLines|)
        (|sayKeyedMsg| 'S2GL0012 (list m))))))

defun printTypeAndTimeSaturn

[makeLongTimeString p??]
[form2StringAsTeX p??]
devaluate p??
[printAsTeX p59]
|$printTimeIfTrue p751|
|$printTypeIfTrue p752|
|$interpreterTimedClasses p??|
|$interpreterTimedNames p??|

— defun printTypeAndTimeSaturn —

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

(when |$printTimeIfTrue|
   (|printAsTeX| "\axPrintTime{"
   (|printAsTeX| timeString)
   (|printAsTeX| "}"))))

---------

defun printAsTeX

[$texOutputStream p??]

   — defun printAsTeX 0 —

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

---------

defun sameUnionBranch

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

   — defun sameUnionBranch 0 —

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

---------

(equal t3 m))
(t (equal uArg m)))))

---

defun msgText

[segmentKeyedMsg p330]
[getKeyedMsg p329]
[substituteSegmentedMsg p??]
[flowSegmentedMsg p??]
[$linelength p774]
[$margin p774]

— defun msgText —

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

---

defun Right-justify the Type output

[fillerSpaces p18]
[$linelength p774]

— defun justifyMyType —

(defun justifyMyType (arg)
  (let (len)
    (declare (special $linelength))
    (setq len (#| arg))
    (if (> len $linelength)
      arg
      (concat (#fillerSpaces (~- $linelength len)) arg))))
4.3. HELPER FUNCTIONS

defun Destructively fix quotes in strings

[unescapeStringsInForm p61]
[$funnyBacks p1297]
[$funnyQuote p1297]

— defun unescapeStringsInForm —

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

————

Include a file into the stream

[ST p??]
[intloopInclude0 p61]

— defun intloopInclude —

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

————

defun intloopInclude0

[incStream p71]
[intloopProcess p62]
[next p36]
[intloopEchoParse p67]
[insertpile p335]
[lineoftoks p111]
CHAPTER 4. HANDLING TERMINAL INPUT

---

(defun intloopInclude0
  (st name n)
  (let (lines)
    (declare (special lines))
    (setq lines (incStream st name))
    (intloopProcess n NIL
      (next #'intloopEchoParse
        (next #'insertpile
          (next #'lineoftoks
            lines)))))

---

defun intloopProcess

(StreamNull p333)
(pfAbSynOp? p412)
(setCurrentLine p40)
tokPart p413
(intloopProcess p62)
(intloopSpadProcess p63)
$systemCommandFunction p??
$systemCommandFunction p??

---

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

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

---

defun intloopSpadProcess, interp

(ncConversationPhase p66)
(ncEltQ p416)
(ncError p67)

---

defun intloopSpadProcess, interp —

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

---

defun phParse

TPDHERE: The pform function has a leading percent sign

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

(ncPutQ p416)

---

defun phParse —

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

---

defun phIntReportMsgs

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

(ncEltQ p416)
(ncPutQ p416)
4.3. HELPER FUNCTIONS

— defun phIntReportMsgs —

(defun phIntReportMsgs (carrier interactive?)
  (declare (ignore interactive?))
  (let (nerr msgs lines)
    (declare (special $erMsgToss))
    (cond
     (|$erMsgToss| 'ok)
     (t
      (setq lines (|ncEltQ| carrier '|lines|))
      (setq msgs (|ncEltQ| carrier '|messages|))
      (setq nerr (length msgs))
      (|ncPutQ| carrier '|ok?| (eql nerr 0))
      (cond
       ((eql nerr 0) 'ok)
       (t
        (|processMsgList| msgs lines)
        (|sayKeyedMsg| 'S2CTP010 (list nerr))
        'ok)))))

---

defun phInterpret

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

---

defun intInterpretPform

(defun intInterpretPform)

\begin{verbatim}
(defun intInterpretPform (pf)
  (processInteractive (zeroOneTran (pf2Sex pf)) pf))

(defun zeroOneTran (sex)
  (nsubst ’$EmptyMode ’? sex))

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

(defun ncConversationPhase,wrapup)
\end{verbatim}
4.3. HELPER FUNCTIONS

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

------

defun ncError
[SpadCompileItem p??]

— defun ncError 0 —

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

------

defun intloopEchoParse
[ncloopDQlines p70]
[setCurrentLine p40]
[mkLineList p68]
[ncloopPrintLines p68]
[npParse p141]
[dqToList p344]
[$EchoLines p??]
[$lines p??]

— defun intloopEchoParse —

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

---

(defun ncloopPrintLines
  ;ncloopPrintLines lines ==
  ; for line in lines repeat WRITE_-LINE CDR line
  ; WRITE_-LINE " "
  (defun |ncloopPrintLines| (lines)
    (loop
      (cond
        ((or (atom Var4) (progn (setq line (car Var4)) nil))
          (return nil))
        (t (write-line (cdr line))))
      (setq Var4 (cdr Var4)))
    lines nil)
    (write-line " "))

---

defun mkLineList
  ;mkLineList lines ==
  ; l := [CDR line for line in lines | nonBlank CDR line]
  ; #l = 1 => CAR l
  ; l
  (defun |mkLineList| (lines)
    (let (l)
      (let (l)
        (setq l
4.3. HELPER FUNCTIONS

((lambda (Var2 Var1 line)
  (loop
    (cond
      ((or (atom Var1) (progn (setq line (car Var1)) nil))
       (return (nreverse Var2)))
      (t
       (and (nonBlank (cdr line))
        (setq Var2 (cons (cdr line) Var2))))))
    (setq Var1 (cdr Var1))))
  nil lines nil)
  (cond
    ((eq (length l) 1) (car l))
    (t 1)))))

defun nonBlank

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

— defun nonBlank 0 —

(defun |nonBlank| (str)
  (let (value)
    ((lambda (Var3 i)
      (loop
        (cond
          ((> i Var3) (return nil))
          (t
           (cond
            ((not (equal (elt str i) #\Space))
             (identity (progn (setq value t) (return value)))))
            (setq i (+ i 1))))
          (maxindex str) 0)
        value)))
  ———
defun ncloopDQlines

(let (b a)
  (StreamNull stream)
  (setq a (poGlobalLinePosn (tokPosn (cadr dq))))
  (setq b (poGlobalLinePosn (caar stream)))
  (streamChop (+ (- a b) 1) stream))

defun poGlobalLinePosn

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

defun streamChop

Note that changing the name “lyne” to “line” will break the system. I do not know why. The symptom shows up when there is a file with a large contiguous comment spanning enough lines to overflow the stack.
(let (d c lyne b a tmp1)
  (cond
    (((StreamNull) s) (list nil nil))
    ((eql n 0) (list nil s))
    (t
      (setq tmp1 (streamChop (- n 1) (cdr s)))
      (setq a (car tmp1))
      (setq b (cadr tmp1))
      (setq lyne (car s))
      (setq c (ncloopPrefix? "command" (cdr lyne)))
      (setq d (cons (car lyne) (cond (c c) (t (cdr lyne))))))
    (list (cons d a) b))))

defun ncloopInclude0

[incStream p71]
[ncloopProcess p ??]
[next p36]
[ncloopEchoParse p ??]
[insertpile p335]
[lineoftoks p111]
[$lines p ??]

— defun ncloopInclude0 —

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

——

defun incStream

[incRenumber p72]
[incLude p75]
[incRgen p101]
[Top p75]
--- defun incStream ---

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

---

defun incRenumber

[incZip p72]
[incIgen p73]

--- defun incRenumber ---

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

---

defun incZip

[Delay p102]
[incZip1 p72]

--- defun incZip ---

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

---

defun incZip1

[StreamNull p333]
[incZip p72]

--- defun incZip1 ---

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

(defun incIgen
) [Delay p102]
[incIgen1 p73]

--- defun incIgen ---

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

---

(defun incIgen1
) [incIgen p73]

--- defun incIgen1 ---

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

---

(defun incRenumberLine
) [incRenumberItem p74]
[incHandleMessage p74]
--- defun incRenumberLine ---

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

---

defun incRenumberItem

[lnSetGlobalNum p346]

--- defun incRenumberItem ---

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

---

defun incHandleMessage

[ncSoftError p351]
[ncBug p368]

--- defun incHandleMessage 0 ---

(defun incHandleMessage (x)
  "Message handling for the source includer"
  (let ((msgtype (elt (elt x 1) 1))
        (pos (car (elt x 0)))
        (key (car (elt (elt x 1) 0)))
        (args (cadr (elt (elt x 1) 0))))
    (cond ((eq msgtype 'none) 0)
          ((eq msgtype 'error) (ncSoftError pos key args))
          ((eq msgtype 'warning) (ncSoftError pos key args))
          ((eq msgtype 'say) (ncSoftError pos key args))
          (t (ncBug key args))))
defun incLude

[Delay p102]
[incLude1 p79]

— defun incLude —

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

defmacro Rest

— defmacro Rest —

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

defvar $Top

— initvars —

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

— defmacro Rest —

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

defvar $IfSkipToEnd

— initvars —

(defvar IfSkipToEnd 10 "used in incLude1 for parsing")
defvar $IfKeepPart

    — initvars —

    (defvar $IfKeepPart 11 "used in include1 for parsing")

    — —

defvar $IfSkipPart

    — initvars —

    (defvar $IfSkipPart 12 "used in include1 for parsing")

    — —

defvar $ElseifSkipToEnd

    — initvars —

    (defvar $ElseifSkipToEnd 20 "used in include1 for parsing")

    — —

defvar $ElseifKeepPart

    — initvars —

    (defvar $ElseifKeepPart 21 "used in include1 for parsing")

    — —

defvar $ElseifSkipPart

    — initvars —
4.3. HELPER FUNCTIONS

(defvar |ElseifSkipPart| 22 "used in include1 for parsing")

---

defvar $ElseSkipToEnd

— initvars —

(defvar |ElseSkipToEnd| 30 "used in include1 for parsing")

---

defvar $ElseKeepPart

— initvars —

(defvar |ElseKeepPart| 31 "used in include1 for parsing")

---

defvar $Top?

[quotient p??]

— defun Top? 0 —

(defun |Top?| (|st|)
 "used in include1 for parsing"
 (eql (quotient |st| 10) 0))

---

defvar $If?

[quotient p??]

— defun If? —
(defun |If?| (|st|)
  "used in include1 for parsing"
  (eql (quotient |st| 10) 1))

-----

defvar $Elseif?
[QUOTIENT p??]

    — defun Elseif? —

(defun |Elseif?| (|st|)
  "used in include1 for parsing"
  (eql (quotient |st| 10) 2))

-----

defvar $Else?
[QUOTIENT p??]

    — defun Else? —

(defun |Else?| (|st|)
  "used in include1 for parsing"
  (eql (quotient |st| 10) 3))

-----

defvar $SkipEnd?
[remainder p??]

    — defun SkipEnd? —

(defun |SkipEnd?| (|st|)
  "used in include1 for parsing"
  (eql (remainder |st| 10) 0))

-----
defvar $KeepPart?
[remainder p??]

— defun KeepPart? —

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

———

defvar $SkipPart?
[remainder p??]

— defun SkipPart? —

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

———

defvar $Skipping?
[KeepPart? p79]

— defun Skipping? —

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

———

defun incLude1

[StreamNull p333]
[Top? p77]
xlPrematureEOF p84]
[Skipping? p79]
(defun |incLude1| (&rest z)
  (let (pred s1 n tail head includee fn1 info str state lno states ufos ln ss eb)
    (setq eb (car z))
    (setq ss (cadr . (z)))
    (setq ln (caddr . (z)))
    (setq ufos (cadddr . (z)))
    (setq states (car (cddddr . (z))))
    (setq lno (+ ln 1))
    (setq state (elt states 0)))
  — defun incLude1 —

4.3. HELPER FUNCTIONS

(cond
  ((|StreamNull| ss)
    (cond
      ((null (|Top?| state))
        (cons (|xlPrematureEOF| eb "--premature end" lno ufos)
          |StreamNil|))
      (t |StreamNil|))
    (t
      (progn
        (setq str (expand-tabs (car ss)))
        (setq info (|incClassify| str))
        (cond
          ((null (elt info 0))
            (cond
              ((|Skipping?| state)
                (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|))
              (t
                (cons (|xlOK| eb str lno (elt ufos 0)) (|Rest|))))
            ((equal (elt info 2) "other")
              (cond
                ((|Skipping?| state)
                  (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|))
                (t
                  (cons
                    (|xlOK1| eb str (concat "command" str) lno (elt ufos 0))
                    (|Rest|)))))
            ((equal (elt info 2) "say")
              (cond
                ((|Skipping?| state)
                  (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|))
                (t
                  (progn
                    (setq str (|incCommandTail| str info))
                    (cons (|xlSay| eb str lno ufos str)
                      (cons (|xlOK| eb str lno (ELT ufos 0)) (|Rest|))))))
            ((equal (elt info 2) "include")
              (cond
                ((|Skipping?| state)
                  (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|))
                (t
                  (progn
                    (setq fn1 (|inclFname| str info))
                    (cond
                      ((null fn1)
                        (cons (|xlNoSuchFile| eb str lno ufos fn1) (|Rest|))
                      ((null (probe-file fn1))
                        (cons (|xlCannotRead| eb str lno ufos fn1) (|Rest|))
                      ((|incActive?| fn1 ufos)
                        (cons (|xlFileCycle| eb str lno ufos fn1) (|Rest|))
                      (t
                        (cons (|xlNoSuchFile| eb str lno ufos fn1) (|Rest|)))))))))
            )))
  )))
)
(progn
  (setq includee
    (|include| (+ eb (elt info 1))
      (|incFileInput| fn1)
    0
    (cons fn1 ufos)
    (cons |Top| states)))
  (cons (|xlOK| eb str lno (elt ufos 0))
    (|incAppend| includee (|Rest|)))))))
((equal (elt info 2) "console")
 (cond
   (||Skipping?| state)
   (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
   (t
    (progn
      (setq head
        (|include| (+ eb (elt info 1))
          (|incConsoleInput|)
        0
        (cons "console" ufos)
        (cons |Top| states)))
      (setq tail (|Rest|))
      (setq n (|incNConsoles| ufos))
      (cond
       ((< 0 n)
        (setq head
          (cons (|xlConActive| eb str lno ufos n) head))
        (setq tail
          (cons (|xlConStill| eb str lno ufos n) tail))))
      (setq head (cons (|xlConsole| eb str lno ufos) head))
      (cons (|xlOK| eb str lno (elt ufos 0))
        (|incAppend| head tail)))))))
((equal (elt info 2) "fin")
 (cond
   (||Skipping?| state)
   (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
   (null (|Top?| state))
   (cons (|xlPrematureFin| eb str lno ufos) |StreamNil|)
   (t
    (cons (|xlOK| eb str lno (elt ufos 0)) |StreamNil|)))
((equal (elt info 2) "assert")
 (cond
   (||Skipping?| state)
   (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
   (t
    (progn
      (|assertCond| str info)
      (cons (|xlOK| eb str lno (elt ufos 0))
        (|incAppend| includee (|Rest|)))))))
((equal (elt info 2) "if")

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

defun xlPrematureEOF

[xlMsg p84]
[inclmsgPrematureEOF p86]

— defun xlPrematureEOF —

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

defun xlMsg

[inclLine p86]

— defun xlMsg —

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

defun xlOK

[lxOK1 p??]

— defun xlOK —

(defun |xlOK| (extrablanks string localnum fileobj)
  (|lxOK1| extrablanks string string localnum fileobj))
defun xlOK1

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

----------

defun incAppend

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

----------

defun incAppend1

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

[incLine1 p86]

— defun incLine —

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

defun incLine1

[lnCreate p345]

— defun incLine1 —

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

defun inclmsgPrematureEOF

[origin p??]

— defun inclmsgPrematureEOF 0 —

(defun inclmsgPrematureEOF (ufo)
  (list 'S2C10002 (list (theorigin ufo))))

defun theorigin

— defun theorigin 0 —
4.3. HELPER FUNCTIONS

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

---

defun porigin

[stringp p??]

---

defun ifCond

[MakeSymbol p??] [incCommandTail p99] [$inclAssertions p??]

---

defun xlSkip

[incLine p86] [CONCAT p??]
(let ((string (concat "-- Omitting:" str)) (globalnum -1))
  (list
    (incLine extrablanks string globalnum localnum fileobj)
    (list nil 'none))))

---

(defun xlSay
  [xlMsg p84]
  [inclmsgSay p88]

  — defun xlSay —

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

---

(defun inclmsgSay
  [id p??]

  — defun inclmsgSay —

  (defun |inclmsgSay| (str)
    (list 'S2Cl0001 (list (|theid| str)))))

---

(defun theid

  — defun theid 0 —

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

---
defun xlNoSuchFile

[xlMsg p84]
[inclmsgNoSuchFile p89]

—— defun xlNoSuchFile ——

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

——

defun inclmsgNoSuchFile

[thefname p89]

—— defun inclmsgNoSuchFile ——

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

——

defun thefname

[pfname p89]

—— defun thefname 0 ——

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

——

defun pfname

[PathnameString p??]

—— defun pfname ——

(defun pfname (x) (listPathnameString x))

——
defun xlCannotRead

[xlMsg p84]
[inclmsgCannotRead p90]

— defun xlCannotRead —

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

-----

defun inclmsgCannotRead

[thefname p89]

— defun inclmsgCannotRead —

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

-----

defun xlFileCycle

[xlMsg p84]
[inclmsgFileCycle p90]

— defun xlFileCycle —

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

-----

defun inclmsgFileCycle

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

— defun inclmsgFileCycle —

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

---

defun xlConActive

[xMsg p84]
[inclmsgConActive p92]

— defun xlConActive —

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

---
defun inclmsgConActive

[id p??]

— defun inclmsgConActive —

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

———

defun xlConStill

[xlMsg p84]
[inclmsgConStill p92]

— defun xlConStill —

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

———

defun inclmsgConStill

[id p??]

— defun inclmsgConStill —

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

———

defun xlConsole

[xlMsg p84]
[inclmsgConsole p93]

— defun xlConsole —

(defun xlConsole (eb str lno ufos)
  (xlMsg eb str lno (elt ufos 0) (list (inclmsgConsole (|inclmsgConsole|) '|say|))))
4.3. HELPER FUNCTIONS

---

defun inclmsgConsole

— defun inclmsgConsole 0 —

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

---

defun xlSkippingFin

[xlMsg p84]
inclmsgFinSkipped p93

— defun xlSkippingFin —

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

---

defun inclmsgFinSkipped

— defun inclmsgFinSkipped 0 —

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

---

defun xlPrematureFin

[xlMsg p84]
inclmsgPrematureFin p94

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

defun inclmsgPrematureFin

[origin p??]
  ---
  defun inclmsgPrematureFin ---

  (defun inclmsgPrematureFin (ufo)
    (list 'S2C10003 (list (theorigin ufo))))

---

defun assertCond

[MakeSymbol p??]
  [incCommandTail p99]
  [$inclAssertions p??]
  [*whitespace* p22]
  ---
  defun assertCond ---

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

---

defun xlIfSyntax

[Top? p77]
  [Else? p78]
  [xlMsg p84]
  [inclmsgIfSyntax p95]
4.3. HELPER FUNCTIONS

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

———-

defun inclmsgIfSyntax
[concat p1047]
[id p??]
[origin p??]
— defun inclmsgIfSyntax —
(defun |inclmsgIfSyntax| (ufo found context)
(setq found (concat ")" found))
(list ’S2CI0009 (list (|theid| found)
(|theid| context)
(|theorigin| ufo))))

———-

defun xlIfBug
[xlMsg p84]
[inclmsgIfBug p96]
— defun xlIfBug —
(defun |xlIfBug| (eb str lno ufos)
(|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgIfBug|) ’|bug|)))

———-

95


defun inclmsgIfBug
   — defun inclmsgIfBug 0 —
   (defun inclmsgIfBug ()
      (list 'S2CB0002 nil))

defun xlCmdBug

   [xlMsg p84]
   [inclmsgCmdBug p96]
   — defun xlCmdBug —
   (defun xlCmdBug (eb str lno ufos)
      (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgCmdBug|) '|bug|)))

defun inclmsgCmdBug
   — defun inclmsgCmdBug 0 —
   (defun inclmsgCmdBug ()
      (list 'S2CB0003 nil))

defvar $incCommands
   This is a list of commands that can be in an include file
   — postvars —
   (eval-when (eval load)
      (setq incCommands (list "say" "include" "console" "fin" "assert" "if" "elseif" "else" "endif")))
4.3. HELPER FUNCTIONS

defvar $pfMacros

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

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

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

-- initvars --

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

---

defun incClassify

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

[incCommand? p98]
[incCommands p96]

-- defun incClassify --

```
(defun |incClassify| (s)
(let (p1 bad eb n i)
  (declare (special |incCommands|)))
```
(if (null (incCommand? s))
  (list nil 0 "")
  (progn
    (setq i 1)
    (setq n (length s))
    ((lambda ()
       (loop
        (cond
          ((not (and (< i n) (char= (elt s i) #\space)))
           (return nil))
          (t (setq i (1+ i)))))))
    (cond
      ((not (< i n)) (list t 0 "other"))
      (t
       (if (= i 1)
        (setq eb 0)
        (setq eb i))
       (setq bad t)
       ((lambda (tmp1 p)
          (loop
           (cond
             ((or (atom tmp1)
                (progn (setq p (car tmp1)) nil)
                (not bad))
              (return nil))
             (t
              (cond
               (incPrefix? p i s)
               (identity
                (progn
                 (setq bad nil)
                 (setq p1 p)))))))
        (setq tmp1 (cdr tmp1))))
      (incCommands nil)
      (if bad
       (list t 0 "other")
       (list t eb p1)))))

---

(defun incCommand? [char p??]
  — defun incCommand? 0 —

(defun [incCommand?] (s)
  "does this start with a close paren?"
4.3. HELPER FUNCTIONS

(and (< 0 (length s)) (equal (elt s 0) \\))

---

defun incPrefix?

;incPrefix? (prefix, start, whole) ==
; #prefix > #whole-start => false
; good:=true
; for i in 0..#prefix-1 for j in start.. while good repeat
; good:= prefix.i = whole.j
; good

— defun incPrefix? 0 —

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

---

defun incCommandTail

[incDrop p100]

— defun incCommandTail —

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

———
defun incDrop

[substring p??]

— defun incDrop 0 —

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

———

defun inclFname

[incFileName p622] [incCommandTail p99]

— defun inclFname —

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

———

defun incFileInput

[incRgen p101] [make-instream p981]

— defun incFileInput —

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

———

defun incConsoleInput

[incRgen p101] [make-instream p981]

— defun incConsoleInput —
4.3. HELPER FUNCTIONS

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

defun incNConsoles

[incNConsoles p101]

— defun incNConsoles —

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

defun incActive?

— defun incActive? 0 —

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

defun incRgen

Note that incRgen1 recursively calls this function. [Delay p102]
[incRgen1 p102]

— defun incRgen —

(defun |incRgen| (s)
  (|Delay| #'|incRgen1| (list s)))
defun Delay
  — defun Delay 0 —
  (defun |Delay| (f x)
    (cons '|nonnullstream| (cons f x)))

defvar $StreamNil
  — initvars —
  (defvar |StreamNil| (list '|nullstream|))

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

defun incRgen1
  This function reads a line from the stream and then conses it up with a recursive call to incRgen. Note that incRgen recursively wraps this function in a delay list. [incRgen p101] [StreamNil p102]
  — defun incRgen1 —
  (defun |incRgen1| (&rest z)
    (let (a s)
      (declare (special |StreamNil|))
      (setq s (car z))
      (setq a (read-line s nil nil))
      (if (null a)
        (progn
          (close s)
          |StreamNil|)))
(cons a (incRgen s)))
Chapter 5

The Token Scanner

defvar $SPACE

    — postvars —

    (eval-when (eval load)
        (defvar SPACE (qenum " " 0)))

———

defvar $ESCAPE

    — postvars —

    (eval-when (eval load)
        (defvar ESCAPE (qenum "_ " 0)))

———

defvar $STRINGCHAR

    — postvars —

    (eval-when (eval load)
        (defvar STRINGCHAR (qenum "\ " 0)))
defvar $PLUSCOMMENT

— postvars —

(eval-when (eval load)
  (defvar PLUSCOMMENT (qenum "+" 0)))

defvar $MINUSCOMMENT

— postvars —

(eval-when (eval load)
  (defvar MINUSCOMMENT (qenum "-" 0)))

defvar $RADIXCHAR

— postvars —

(eval-when (eval load)
  (defvar RADIXCHAR (qenum "r" 0)))

defvar $DOT

— postvars —

(eval-when (eval load)
  (defvar DOT (qenum "." 0)))
defvar $EXPONENT1

— postvars —

(eval-when (eval load)
  (defvar EXPONENT1 (qenum "E" "0")))

-----

defvar $EXPONENT2

— postvars —

(eval-when (eval load)
  (defvar EXPONENT2 (qenum "e" "0")))

-----

defvar $CLOSEPAREN

— postvars —

(eval-when (eval load)
  (defvar CLOSEPAREN (qenum "" "0")))

-----

defvar $QUESTION

— postvars —

(eval-when (eval load)
  (defvar QUESTION (qenum "?" "0")))

-----
defvar $scanKeyWords

— postvars —

(eval-when (eval load)
(defvar |scanKeyWords|
(list
  (list "add" 'add)
  (list "and" 'and)
  (list "break" 'break)
  (list "by" 'by)
  (list "case" 'case)
  (list "default" 'default)
  (list "define" 'defn)
  (list "do" 'do)
  (list "else" 'else)
  (list "exit" 'exit)
  (list "export" 'export)
  (list "for" 'for)
  (list "free" 'free)
  (list "from" 'from)
  (list "has" 'has)
  (list "if" 'if)
  (list "import" 'import)
  (list "in" 'in)
  (list "inline" 'inline)
  (list "is" 'is)
  (list "isnt" 'isnt)
  (list "iterate" 'iterate)
  (list "local" '|local|)
  (list "macro" 'macro)
  (list "mod" 'mod)
  (list "or" 'or)
  (list "pretend" 'pretend)
  (list "quo" 'quo)
  (list "rem" 'rem)
  (list "repeat" 'repeat)
  (list "return" 'return)
  (list "rule" 'rule)
  (list "then" 'then)
  (list "where" 'where)
  (list "while" 'while)
  (list "with" 'with)
  (list "|" 'bar)
  (list "." 'dot)
  (list ":::" 'coerce)
  (list ":" 'colon)
  (list ":-" 'colondash)
  (list "@" 'at)
defvar $infgeneric

— postvars —

(eval-when (eval load)
  (prog ()
    (return
      ((lambda (var value)
         (loop
           (cond
             ((or (atom var) (progn (setq value (car var)) nil))
              (return nil))
             (t
              (setf (get (car value) 'infgeneric) (cadr value))))
           (setq var (cdr var))))
       (list 'equal '=)
       (list 'times '*)
       (list 'has '|has!)
       (list 'case '|case!)
       (list 'rem '|rem!)
       (list 'mod '|mod!)
       (list 'quo '|quo!)
       (list 'slash '/)
       (list 'backslash '\\)
       (list 'slashslash //'\)
       (list 'backslashbackslash '\\\)
       (list 'slashbackslash '//\)
       (list 'power '**)
       (list 'carat '^)
       (list 'plus '+)
       (list 'minus '-)
       (list 'lt '<)
       (list 'gt '>)
       (list 'oangle '<<)
       (list 'cangle '>>)
       (list 'le '<=)
       (list 'ge '>=)
       (list 'notequal '~=)
       (list 'by '|by!)
       (list 'arrow '->))
defun lineoftoks

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

;lineoftoks(s)==
 ; $f: local:=nil
 ; $r:local :=nil
 ; $ln:local :=nil
 ; $linepos:local:=nil
 ; $n:local :=nil
 ; $sz:local := nil
 ; $floatok:local:=true
 ; if not nextline s
 ; then CONS(nil,nil)
 ; else
 ; if null scanIgnoreLine($ln,$n) -- line of spaces or starts ) or >
 ; then cons(nil,$r)
 ; else
 ; toks:=[]
 ; a:= incPrefix?("command",1,$ln)
 ; a =>
 ; $ln:=SUBSTRING($ln,8,nil)
 ; $b:= dqUnit constoken($ln,$linepos, ["command",$ln],0)
 ; cons([ [b,s] ],$r)
 ;
 ; while $n<$sz repeat toks:=dqAppend(toks,scanToken())
 ; if null toks
 ; then cons([],$r)
 ; else cons([ [toks,s] ],$r)
defun lineoftoks

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

defun nextline

[npNull p333]
[strposl p1046]
[sz p??]
[sz p??]
defun nextline
((defun nextline (s)
  (declare (special $sz $n $linepos $ln $r $f))
  (cond
   ((null s) nil)
   (t
    (setq f (car s))
    (setq r (cdr s))
    (setq ln (cdr f))
    (setq linepos (caar f))
    (setq n (strposl " " ln 0 t)); spaces at beginning
    (setq sz (length ln))
    t))))

defun scanIgnoreLine
(defun scanIgnoreLine (ln n)
  (cond
   ((null n) n)
   (t
    (cond
     ((equal (qenum ln 0) CLOSEPAREN)
      (cond
       (((incPrefix? "command" 1 ln) t)
        (t nil)))
      (t nil))))
    (t n)))))

defun constoken
(defun constoken (ln n)
--- defun constoken ---

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

---

defun scanToken

(qenum p1046)
 [startsComment? p115]
 [scanComment p116]
 [startsNegComment? p117]
 [scanNegComment p117]
 [fid p115]
 [punctuation? p118]
 [scanPunct p118]
 [startsId? p1044]
 [scanWord p126]
 [scanSpace p129]
 [scanString p130]
 [digit? p122]
 [scanNumber p132]
 [scanEscape p135]
 [scanError p135]
 [dqUnit p343]
 [constoken p113]
 [lnExtraBlanks p345]
 [$linepos p??]
 [$n p??]
 [$ln p??]

--- defun scanToken ---

(defun |scanToken| ()
 (let (b ch n linepos c ln)
   (declare (special |$linepos| |$n| |$ln|))
   (setq ln |$ln|)
   (setq c (qenum |$ln| |$n|))
   (setq linepos |$linepos|)
   (setq n |$n|)
   (setq ch (elt |$ln| |$n|)))
(setq b
  (cond
    ((|startsComment?|) (|scanComment|) nil)
    ((|startsNegComment?|) (|scanNegComment|) nil)
    ((equal c QUESTION)
      (setq |$n| (+ |$n| 1))
      (|lfid| "?"))
    ((!punctuation?| c) (|scanPunct|))
    ((|startsId?| ch) (|scanWord| nil))
    ((equal c SPACE) (|scanSpace|) nil)
    ((equal c STRINGCHAR) (|scanString|))
    ((|digit?| ch) (|scanNumber|))
    ((equal c ESCAPE) (|scanEscape|))
    (t (|scanError|)))
  (cond
    ((null b) nil)
    (t
      (|dqUnit|
       (|constoken| ln linepos b (+ n (|lnExtraBlanks| linepos))))))))

defun lfid

To pair badge and badgee

— defun lfid 0 —

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

—

defun startsComment?

[qenum p1046]

$ln p??
|$sz p??
|$n p???

— defun startsComment? —

(defun |startsComment?| ()
  (let (www)
    (declare (special |$ln| |$sz| |$n|))
    (cond
CHAPTER 5. THE TOKEN SCANNER

((< |$n| |$sz|)  
 (cond  
  ((equal (qenum |$ln| |$n|) PLUSCOMMENT)  
    (setq www (+ |$n| 1)))  
    (cond  
      ((not (< www |$sz|)) nil)  
      (t (equal (qenum |$ln| www) PLUSCOMMENT)))))  
  (t nil)))  
  (t nil))))

defun scanComment

[lfcomment p116]  
[substring p??]  
|$ln p??]  
|$sz p??]  
|$n p??]

— defun scanComment —

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

— defun lfcomment —

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

— defun lfcomment 0 —
defun startsNegComment?

(defun |startsNegComment?| ()
  (let (www)
    (declare (special |$ln| |$sz| |$n|))
    (cond
     ((< |$n| |$sz|)
      (cond
       ((equal (qenum |$ln| |$n|) MINUSCOMMENT)
        (setq www (+ |$n| 1)))
       (cond
        ((not (< www |$sz|)) nil)
        (t (equal (qenum |$ln| www) MINUSCOMMENT)))))
     (t nil))))

defun scanNegComment

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

— defun lfnegcomment 0 —

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

———

defun punctuation?

— defun punctuation? —

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

———

defun scanPunct

[subMatch p119]
[scanError p135]
[scanKeyTr p120]
[$n p??]
[$ln p??]

— defun scanPunct —

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

———
defun subMatch

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

defun substringMatch

(defun |substringMatch| (l dict i)
  (let (equl ls s s1 done ll u h)
    (setq h (qenum l i))
    (setq u (elt dict h))
    (setq ll (size l))
    (setq s1 "")
    (setq s1
      (for j in 0.. SIZE u - 1 while not done repeat
        s:=ELT(u,j)
        ls:=SIZE s
        done:=if ls+i > ll
          then false
        else
          eql:= true
          for k in 1..ls-1 while eql repeat
            eql:= EQL(QENUM(s,k),QENUM(l,k+i))
            if eql
              then
                s1:=s
              true
            else false
      s1))

[qenum p1046]
[size p1045]
(loop
  (cond
    ((or (> j Var4) done) (return nil))
    t
    (setq s (elt u j))
    (setq ls (size s))
    (setq done
      (cond
        ((< ll (+ ls i)) nil)
        t
        (setq equl t)
        ((lambda (Var5 k)
          (loop
            (cond
              ((or (> k Var5) (not equl)) (return nil))
              t
              (setq equl (eql (qenum s k) (qenum l (+ k i))))
              (setq k (+ k 1))))
            (- ls 1) 1)
            (cond (equl (setq s1 s) t) (t nil)))))
            (- (size u) 1) 0)
            s1))

---

defun scanKeyTr

[keyword p121]
[scanPossFloat p121]
[lfkey p122]
[scanCloser? p125]
[$floatok p?]

--- defun scanKeyTr ---

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

---
defun keyword
[hget p1044]

— defun keyword 0 —

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

———

defun keyword?
[hget p1044]

— defun keyword? 0 —

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

———

defun scanPossFloat
[digit? p122]
[lfkey p122]
spleI p122
[scanExponent p126]
-ln p??
$sz p??
$n p??

— defun scanPossFloat —

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

———
CHAPTER 5. THE TOKEN SCANNER

defun digit?
[digit p1045]

— defun digit? —

(defun digit? (x)
  (digitp x))

———

defun lfkey
[keyword p121]

— defun lfkey —

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

———

defun spleI
[spleI p122]

— defun spleI —

(defun spleI (dig)
  (spleI1 dig nil))

———

defun spleI1
[qenum p1046]
[substring p222]
[scanEsc p123]
[spleI p122]
[concat p1047]
[$ln p222]
[$sz p222]
--- defun spleI1 ---

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

---

defun scanEsc

;scanEsc()==
; if $n>=$sz
; then if nextline($r)
; then
; while null $n repeat nextline($r)
; scanEsc()
; false
; else false
; else
; n1:=STRPOSL('" ',|ln|,|$n|,true)
; if null n1
; then if nextline($r)
; then
; while null $n repeat nextline($r)
; scanEsc()
(defun scanEsc ()
  (let (n1)
    (declare (special $ln $r $sz $n))
    (cond
      ((not (< $n $sz))
       (cond
         ((nextline $r)
          (loop
            (cond
              ($n (return nil))
              (t (nextline $r)))))
          (scanEsc) nil)
        (t nil))
      (t
       (setq n1 (strpos $ln $n t))
       (cond
        (false
         else false)
        else
        (if ($n=n1)
          then true
          else if QENUM($ln,n1)=ESCAPE
          then
            ($n:=n1+1
            scanEsc()
            false
            else
            ($n:=n1
            startsNegComment? or startsComment?
            =>
            nextline($r)
            scanEsc()
            false
            false)
            nextline p112
            scanEsc p123
            strposl p1046
            qenum p1046
            [startsNegComment? p117]
            [startsComment? p115]
            [$ln ??]
            [$r ??]
            [$sz ??]
            [$n ??]

-- defun scanEsc --
[((null n1)
  (cond
    (((nextline |$r|))
      (lambda ()
        (loop
          (cond
            (|$n| (return nil))
            (t (|nextline| |$r|))))))
      (|scanEsc|)
      nil)
    (t nil)))))
  ((equal |$n| n1) t)
  ((equal (genum |$ln| n1) ESCAPE)
    (setq |$n| (+ n1 1))
    (|scanEsc|)
    nil)
  (t (setq |$n| n1)
    (cond
      ((or (|startsNegComment?|) (|startsComment?|))
        (progn
          (|nextline| |$r|)
          (|scanEsc|)
          nil))
      (t nil))))))))

——

defvar $scanCloser

— postvars —

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

——

defun scanCloser?

[keyword p121]
[scanCloser p125]

— defun scanCloser? 0 —

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

---

defun scanWord

(scanW p128)
(lfid p115)
(keyword? p121)
(lfkey p122)
($floatok p??)

---
defun scanWord ---

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

---

defun scanExponent

(lffloat p127)
(qenum p1046)
(digit? p122)
(spleI p122)
(concat p1047)
($ln p??)
($sz p??)
($n p??)

---
defun scanExponent ---
(defun |scanExponent| (a w)
  (let (c1 e c n)
    (declare (special |$ln| |$sz| |$n|))
    (cond
      ((not (< |$n| |$sz|)) (|lffloat| a w "0"))
      (t
       (setq n |$n|)
       (setq c (qenum |$ln| |$n|))
       (cond
        ((or (equal c EXPONENT1) (equal c EXPONENT2))
         (setq |$n| (+ |$n| 1))
         (cond
          ((not (< |$n| |$sz|))
           (setq |$n| n)
           (|lffloat| a w "0"))
          (|digit?| (elt |$ln| |$n|))
          (setq e (|spleI| #'|digit?|))
          (|lffloat| a w e))
        (t
         (setq c1 (qenum |$ln| |$n|))
         (cond
          ((or (equal c1 PLUSCOMMENT) (equal c1 MINUSCOMMENT))
           (setq |$n| (+ |$n| 1))
           (cond
            ((not (< |$n| |$sz|))
             (setq |$n| n)
             (|lffloat| a w "0"))
            (|digit?| (elt |$ln| |$n|))
            (setq e (|spleI| #'|digit?|))
            (|lffloat| a w)
            (cond
             ((equal c1 MINUSCOMMENT)
              (concat "-" e))
             (t e))))
          (t
           (setq |$n| n)
           (|lffloat| a w "0")))
        (t (|lffloat| a w "0"))))
    (t (|lffloat| a w "0"))))

---

defun lffloat

[concat p1047]

— defun lffloat 0 —

(defun |lffloat| (a w e)
defmacro idChar?

---
defmacro idChar? 0 ---

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

---
defun scanW

---
defun scanW ---

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

defun posend

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

NOTE: do not replace "lyne" with "line"

— defun posend —

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

—

defun scanSpace

[strposl p1046]
[lfspaces p130]
 [$floatok p??]
[$ln p??]
[$n p??]

— defun scanSpace —

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

——

defun lfspaces

--- defun lfspaces 0 ---

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

——

defun scanString

|lfstring p130|
|scanS p131|
|$floatok p?|
|$n p?|

--- defun scanString ---

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

——

defun lfstring

--- defun lfstring 0 ---

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

——
131

defun scanS
[ncSoftError p351]
[lnExtraBlanks p345]
[strpos p1045]
[substring p??]
[scanEsc p123]
[concat p1047]
[scanTransform p132]
[scanS p131]
[$ln p??]
[$linepos p??]
[$sz p??]
[$n p??]
— defun scanS —
(defun |scanS| ()
(let (b a str mn escsym strsym n)
(declare (special |$ln| |$linepos| |$sz| |$n|))
(cond
((not (< |$n| |$sz|))
(|ncSoftError|
(cons |$linepos| (+ (|lnExtraBlanks| |$linepos|) |$n|)) ’S2CN0001 nil) "")
(t
(setq n |$n|)
(setq strsym (or (strpos "\"" |$ln| |$n| nil) |$sz|))
(setq escsym (or (strpos "_" |$ln| |$n| nil) |$sz|))
(setq mn (min strsym escsym))
(cond
((equal mn |$sz|)
(setq |$n| |$sz|)
(|ncSoftError|
(cons |$linepos| (+ (|lnExtraBlanks| |$linepos|) |$n|)) ’S2CN0001 nil)
(substring |$ln| n nil))
((equal mn strsym)
(setq |$n| (+ mn 1))
(substring |$ln| n (- mn n)))
(t
(setq str (substring |$ln| n (- mn n)))
(setq |$n| (+ mn 1))
(setq a (|scanEsc|))
(setq b
(cond
(a
(setq str (concat str (|scanTransform| (elt |$ln| |$n|))))
(setq |$n| (+ |$n| 1)) (|scanS|))
(t (|scanS|))))
(concat str b)))))))


ch. 5.

---

`defun scanTransform`

---

```lisp
(defun scanTransform (x) x)
```

---

`defun scanNumber`

```lisp
(defun scanNumber ()
  (let (v w n a)
    (declare (special $floatok $ln $sz $n))
    (setq a (spleI #'digit?))
    (cond
      ((not (< $n $sz))
       (lfinteger a))
      ((not (equal (qenum $ln $n) RADIXCHAR))
       (cond
         ((and $floatok (equal (qenum $ln $n) DOT))
          (setq n $n))
         (setq $n (+ $n 1)))
       (cond
         ((and (< $n $sz) (equal (qenum $ln $n) DOT)))
         (setq $n n)))
    (cond
      ((not (equal (qenum $ln $n) DOT))
       (setq n $n))
      (setq $n (+ $n 1)))
    (cond
      ((and (< $n $sz) (equal (qenum $ln $n) DOT)))
      (setq $n n))
```
(defun rdigit? (x)
  (strpos x "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" nil))

---

defun lfinteger

---

defun lfinteger 0 ---

(defun lfinteger (x)
  (list [integer| x]))
defun lfrinteger
[concat p1047]

— defun lfrinteger 0 —

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

defun scanCheckRadix

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

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

— defun scanCheckRadix —

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

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

defun scanError

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

defun lferror

— defun lferror 0 —
(defun |lferror| (x)
  (list '|error| x))

---

defvar $scanKeyTable

—— postvars ——

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

---

defun scanKeyTableCons

This function is used to build the scanKeyTable

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

—— defun scanKeyTableCons ——

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

——
defvar $scanDict

— postvars —

eval-when (eval load)
(.defvar |scanDict| ([scanDictCons]))

——

defun scanDictCons

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

[hkeys p1044]

— defun scanDictCons —

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

\[
((\text{or (atom Var7)} \ (\text{progn \ (setq s (car Var7)) \ nil})) \\
\ (\text{return nil})) \\
\ (t \ (\text{scanInsert} \ s \ d))) \\
\ (\text{setq Var7 (cdr Var7)))) \\
\ l \ nil) \\
\ d))
\]

---

defun scanInsert

; scanInsert(s, d) ==
;  l := #s
;  h := QENUM(s, 0)
;  u := ELT(d, h)
;  n := #u
;  k := 0
;  while l <= #(ELT(u, k)) repeat
;    k := k + 1
;    v := MAKE_VEC(n + 1)
;    for i in 0..k-1 repeat VEC_SETELT(v, i, ELT(u, i))
;    VEC_SETELT(v, k, s)
;    for i in k..n-1 repeat VEC_SETELT(v, i + 1, ELT(u, i))
;    VEC_SETELT(d, h, v)
;  s

[qenum p1046]

defun scanInsert

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

-----

defvar $scanPun

— postvars —

eval-when (eval load)
  (defvar |scanPun| ([iscanPunCons|)))

-----

defun scanPunCons

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

[hkeys p1044]

— defun scanPunCons —

(defun |scanPunCons| ()
 (let (a listing)
    (setq listing (hkeys |scanKeyTable|))
    (setq a (make-array (list 256) :element-type 'bit :initial-element 0)))
((lambda (i)
  (loop
    (cond
      ((> i 255) (return nil))
      (t (setf (sbit a i) 0)))
    (setq i (+ i 1))))
  0)
((lambda (Var8 k)
  (loop
    (cond
      ((or (atom Var8) (progn (setq k (car Var8)) nil))
       (return nil))
      (t
       (cond
        ((null (startsId? (elt k 0)))
         (setq (sbit a (qenum k 0)) 1)))
        (setq Var8 (cdr Var8))))
  listing nil)
  a))}
Chapter 6

Input Stream Parser

defun Input Stream Parser

(defun npParse
  (defun npParse (stream)
    (let (($ttok| |$stok| |$stack| |$inputStream| found)
      (declare (special |$ttok| |$stack| |$inputStream| |$stok|))
      (setq |$inputStream| stream)
      (setq |$stack| nil)
      (setq |$stok| nil)
      (setq |$ttok| nil)
      (npFirstTok)
      (setq found (catch 'trappoint (npItem)))
      (cond
        ((eq found 'trapped)
         (ncSoftError (tokPosn |$stok|) 's2cy0006 nil)
         (pfWrong (pfDocument "top level syntax error") (pfListOf nil)))
        ((null (null |$inputStream|)) 141
          (setq |$inputStream| (catch 'trappoint (npItem)))))
    )))

141
(\|ncSoftError\| (\|tokPosn\| \$stok\|) 's2cy0002 nil)
(\|pfWrong\|
 (\|pfDocument\| (list "input stream not exhausted"))
 (\|pfList0f\| nil)))
((null \$stack\|)
(\|ncSoftError\| (\|tokPosn\| \$stok\|) 's2cy0009 nil)
(\|pfWrong\| ((\|pfDocument\| (list "stack empty")) (\|pfList0f\| nil)))
(t (car \$stack\|))))

defun npItem

[npQualDef p145]
[npEqKey p145]
[npItem1 p142]
[npPop1 p144]
[pfEnSequence p263]
[npPush p143]
[pfNovalue p279]

— defun npItem —

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

—

defun npItem1

[npQualDef p145]
[npEqKey p145]
[npItem1 p142]
[npPop1 p144]
— defun npItem1 —

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

——

defun npFirstTok

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

  [tokPosn p413]
  [tokPart p413]
  [$ttok p??]
  [$stok p??]
  [$inputStream p??]

— defun npFirstTok —

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

——

defun Push one item onto $stack

[$stack p??]

— defun npPush 0 —
(defun npPush (x)
  (declare (special \$stack))
  (push x \$stack))

---

defun Pop one item off \$stack

[\$stack p??]

    — defun npPop1 0 —

(defun npPop1 ()
  (declare (special \$stack))
  (pop \$stack))

---

defun Pop the second item off \$stack

[\$stack p??]

    — defun npPop2 0 —

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

---

defun Pop the third item off \$stack

[\$stack p??]

    — defun npPop3 0 —

(defun npPop3 ()
  (let (a)
    (declare (special \$stack))
    (setq a (caddr \$stack)))
(rplacd (cdr |$stack|) (cdddr |$stack|)) a))

---

defun npQualDef

[npComma p146]  
[npPush p143]  
[npPop1 p144]

---

---

defun Advance over a keyword

Test for the keyword, if found advance the token stream [npNext p145]

---

defun npEqKey

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

---

defun Advance the input stream

This advances the input stream. The call to npFirstTok picks off the next token in the input stream and updates the current leaf ($stok) and the current token ($ttok) [npFirstTok p143] [inputStream p??]
(defun \npNext\ ()
  (declare (special \$inputStream\))
  (setq \$inputStream\ (cdr \$inputStream\))
  (\npFirstTok\))

—

defun npComma

[npTuple p146]
[npQualifiedDefinition p147]

— defun npComma —

(defun \npComma\ ()
  (\npTuple\ #\npQualifiedDefinition\))

—

defun npTuple

[npListofFun p221]
[npCommaBackSet p146]
[pfTupleListOf p292]

— defun npTuple —

(defun \npTuple\ (\p\)
  (\npListofFun\ \p\ #\npCommaBackSet\ #\pfTupleListOf))

—

defun npCommaBackSet

[npEqKey p145]

— defun npCommaBackSet —

(defun \npCommaBackSet\ ()
  (and
    (\npEqKey\ 'comma)
    (or (\npEqKey\ 'backset) t))))
defun npQualifiedDefinition

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

defun npQualified

(defun npQualified (f)
  (if (funcall f)
      (progn
        (do () ; while ... do
          (not (and (npEqKey 'where) (or (npDefinition) (npTrap))))))
      (npLetQualified f)))

defun npDefinitionOrStatement

(defun npDefinitionOrStatement ()
  (npBackTrack)
  (npGives)
  (npDef))

— defun npQualifiedDefinition —

— defun npQualified —

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

---

defun npBackTrack

[npState p212]
[npEqPeek p152]
[npRestore p152]
[upTrap p212]

— defun npBackTrack —

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

---

defun npGives

[npBackTrack p148]
[npExit p215]
[npLambda p148]

— defun npGives —

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

---

defun npLambda

[npVariable p213]
[npLambda p148]
--- defun npLambda ---

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

---

defun npType

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

[npLeftAssoc p206]
[npSuch p150]

— defun npMatch —

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

——

defun npSuch

[npLeftAssoc p206]
[npLogical p197]

— defun npSuch —

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

——

defun npWith

[npEqKey p145]
[npState p212]
[npCategoryL p152]
[npTrap p212]
[npEqPeek p152]
[npRestore p152]
[npVariable p213]
[npCompMissing p151]
[npPush p143]
[pfWith p296]
[npPop2 p144]
[npPop1 p144]
[pfNothing p245]

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

defun npCompMissing

[npEqKey p145]
[npMissing p151]

  — defun npCompMissing —

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

defun npMissing

[trappoint p??]
[ncSoftError p351]
tokPosn p413
[pname p1045]
[$stok p??]

  — defun npMissing —

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

[npFirstTok p143]
[$stack p??]
[$inputStream p??]

— defun npRestore —

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

——

defun Peek for keyword s, no advance of token stream

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

— defun npEqPeek 0 —

(defun npEqPeek (s)
  (declare (special $ttok $stok))
  (and (eq (caar $stok) ’key) (eq s $ttok)))

——

defun npCategoryL

[npCategory p153]
[npPush p143]
[npUnSequence p293]
[npPop1 p144]

— defun npCategoryL —
(defun npCategoryL ()
  (and
    (npCategory)
    (npPush (list (npUnSequence (npPop1)))))
)

defun npCategory

[npPP p209]
[npSCategory p153]

— defun npCategory —

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

— defun npSCategory —

(defun npSCategory ()
  (let (a)
    (cond
      ((npWConditional #'npCategoryL) (npPush (list (npPop1))))
      ((npDefaultValue) t)
      (t
        (setq a (npState))))
  ))
(cond
  ((|npPrimary|)
    (cond
      (((|npEqPeek| 'colon) (|npRestore| a) (|npSignature|))
        (t
          (|npRestore| a)
          (or
            (and (|npApplication|) (|npPush| (list (|pfAttribute| (|npPop1|))))
            (|npTrap|))))))
      (t nil)))))

---

defun npSignature

[|npSigItemlist p154|]
[|npPush p143|]
[|pfWDec p293|]
[|pfNothing p245|]
[|npPop1 p144|]

  — defun npSignature —

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

---

defun npSigItemlist

[|npListing p155|]
[|npSigItem p156|]
[|npPush p143|]
[|pfListOf p245|]
[|pfAppend p255|]
[|pfParts p249|]
[|npPop1 p144|]

  — defun npSigItemlist —

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

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

— defun npListing —

defun Always produces a list, fn is applied to it

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

— defun npList —
defun npSigItem

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

defun npTypeVariable

(defun npTypeVariable ()
(or
 (npParenthesized #'npTypeVariablelist)
 (and (npSignatureDefinee) (npPush (npList0f (list (npPop1)))))))

defun npSignatureDefinee

(defun npSignatureDefinee ()
(or
 (npName #'npTypeVariablelist)
 (npInfixOperator)
 (npPrefixColon)))
(defun npSignatureDefinee ()
  (or (npName) (npInfixOperator) (npPrefixColon)))

---

defun npTypeVariablelist

[npListing p155]
[npSignatureDefinee p156]

---

defun npSigDecl

[npEqKey p145]
[npType p149]
[npTrap p212]
[npPush p143]
[pfSpread p239]
[pfParts p249]
[npPop2 p144]
[npPop1 p144]

---

defun npPrimary

[npPrimary1 p164]
[npPrimary2 p158]
— defun npPrimary —

(defun npPrimary ()
  (or (npPrimary1) (npPrimary2))
)

— defun npPrimary2 —

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

— defun npADD —

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

TPDHERE: Note that there is also an npAdd function

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

TPDHERE: Note that there is also an npADD function [npEqKey p145]
[npState p212]
[npDefinitionOrStatement p147]
[npTrap p212]
[npEqPeek p152]
[npRestore p152]
[npVariable p213]
[npCompMissing p151]
[npDefinitionOrStatement p147]
[npPush p143]
[pfAdd p252]
[npPop2 p144]
[npPop1 p144]
[pfNothing p245

— defun npAdd —

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

defun npAtom2

[npInfixOperator p160]
[npAmpersand p204]
CHAPTER 6. INPUT STREAM PARSER

--- defun npAtom2 ---

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

--- defun npInfixOperator ---

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

[npPushId p209]
[$ttok p??]
[$stok p??]

— defun npInfixOp —

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

— defun npPrefixColon —

(defun npPrefixColon

[npEqPeek p152]
[npPush p143]
[tokConstruct p411]
[tokPosn p413]
[npNext p145]
[$stok p??]

|npPrefixColon|

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

———
defun npApplication

[npDotted p162]
[npPrimary p157]
[npApplication2 p163]
[npPush p143]
[pfApplication p253]
[npPop2 p144]
[npPop1 p144]

— defun npApplication —

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

———

defun npDotted

[ p??]

— defun npDotted —

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

———

defun npAnyNo

fn must transform the head of the stack

— defun npAnyNo 0 —

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

———
defun npSelector
[ npEqKey p145 ]
[ npPrimary p157 ]
[ npTrap p212 ]
[ npPush p143 ]
[ pfApplication p253 ]
[ npPop2 p144 ]
[ npPop1 p144 ]

— defun npSelector —

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

defun npApplication2
[ npDotted p162 ]
[ npPrimary1 p164 ]
[ npApplication2 p163 ]
[ npPush p143 ]
[ pfApplication p253 ]
[ npPop2 p144 ]
[ npPop1 p144 ]

— defun npApplication2 —

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

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

— defun npPrimary1 —

defun npMacro

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

— defun npMacro —

defun npMdef

TPDHERE: Beware that this function occurs with uppercase also [npQuiver p198]

[pfCheckMacroOut p240]

[npPop1 p144]
--- defun npMdef ---

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

---

defun npMDEF

TPDHERE: Beware that this function occurs with lowercase also [npBackTrack
p148]
[npStatement p170]
[npMDEFinition p165]

--- defun npMDEF ---

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

---

defun npMDEFinition

[npPP p209]
[npMdef p164]

--- defun npMDEFinition ---
(defun npMDefinition () ()
  (npPP '#\npMdef)))

---

defun npFix

[npEqKey p145]
[npDef p187]
[npPush p143]
[pfFix p265]
[npPop1 p144]

---

defun npLet

[npLetQualified p166]
[npDefinitionOrStatement p147]

---

defun npLetQualified

[npEqKey p145]
[npDefinition p167]
[npTrap p212]
[npCompMissing p151]
[npPush p143]
— defun npLetQualified —

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

— defun npDefinition —

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

— defun npDefinitionItem —

(defun npDefinitionItem
  (npTyping
   npImport
   npState
   npStatement
   npEqPeek
   npRestore
   npDef
   npMacro
   npDefn)
CHAPTER 6. INPUT STREAM PARSER

[npTrap p212]

--- defun npDefinitionItem ---

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

---

defun npTyping

[npEqKey p145]
[npDefaultItemlist p168]
[npTrap p212]
[npPush p143]
[pfTyping p291]
[npPop1 p144]

--- defun npTyping ---

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

---

defun npDefaultItemlist

[npPC p??]
[npSDefaultItem p169]
— defun npDefaultItemlist —

(defun |npDefaultItemlist| ()
(and
  (npPC |npSDefaultItem|)
  (npPush ((pfUnSequence (npPop1)))))

———

defun npSDefaultItem

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

———

defun npDefaultItem

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

[npEqKey p145]
[npType p149]
[npTrap p212]
[npPush p143]
[pfSpread p239]
[pfParts p249]
[npPop2 p144]
[npPop1 p144]

---

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

---

defun npStatement

[npExpress p179]
[npLoop p175]
[npIterate p174]
[npReturn p178]
[npBreak p174]
[npFree p173]
[npImport p180]
[npInline p174]
[npLocal p173]
[npExport p171]
[npTyping p168]
[npVoid p179]

---

(defun npStatement ()
(or
  (npExpress)
  (npLoop)
  (npIterate)
  (npReturn)
  (npBreak)
  (npFree))
defun npExport

(npEqKey 145)
(npLocalItemlist 171)
(npTrap 212)
(npPush 143)
(pfExport 264)
(npPop1 144)

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

defun npLocalItemlist

(npPC ??)
(npSLocalItem 172)
(npPush 143)
(pfUnSequence 293)
(npPop1 144)

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

———
defun npSLocalItem

[npListing p155]
[npLocalItem p172]
[npPush p143]
[pfAppend p255]
[pfParts p249]
[npPop1 p144]

— defun npSLocalItem —

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

———

defun npLocalItem

[npTypeVariable p156]
[npLocalDecl p172]

— defun npLocalItem —

(defun npLocalItem ()
    (and
        (npTypeVariable)
        (npLocalDecl))))

———

defun npLocalDecl

[npEqKey p145]
[npType p149]
[npTrap p212]
[npPush p143]
[pfSpread p239]
[pfParts p249]
[npPop2 p144]
[npPop1 p144]
[pfNothing p245]
defun npLocalDecl

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

defun npLocal

[npEqKey p145]
[npLocalItemlist p171]
[npTrap p212]
[npPush p143]
[pfLocal p274]
[npPop1 p144]

— defun npLocal —

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

———

defun npFree

[npEqKey p145]
[npLocalItemlist p171]
[npTrap p212]
[npPush p143]
[pfFree p265]
[npPop1 p144]

— defun npFree —

(defun npFree ()
)
(and
  (npEqKey 'free)
  (or (npLocalItemlist) (npTrap)))
  (npPush (pfFree (npPop1))))

---

defun npInline

[npAndOr p181]
[npQualTypelist p180]
[pfInline p272]

— defun npInline —

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

---

defun npIterate

[npEqKey p145]
[npPush p143]
[pfIterate p271]
[pfNothing p245]

— defun npIterate —

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

---

defun npBreak

[npEqKey p145]
[npPush p143]
[pfBreak p258]
[pfNothing p245]

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

defun npLoop

[npIterators p175]
[npCompMissing p151]
[npAssign p216]
[npTrap p212]
[npPush p143]
[pfLp p276]
[npPop2 p144]
[npPop1 p144]
[npEqKey p145]
[pfLoop1 p275]

— defun npLoop —

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

———

defun npIterators

[npForIn p177]
[npZeroOrMore p177]
[npIterator p176]
[npPush p143]
[npPop2 p144]
[npPop1 p144]
[npWhile p177]
[npIterators p175]
--- defun npIterators ---

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

---

defun npIterator

[npForIn p177]
[npSuchThat p176]
[npWhile p177]

--- defun npIterator ---

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

---

defun npSuchThat

[npAndOr p181]
[npLogical p197]
[pfSuchthat p288]

--- defun npSuchThat ---

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

---
defun Apply argument 0 or more times

(npPush p143)
(npPop2 p144)
(npPop1 p144)
($stack p??)

— defun npZeroOrMore —

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

defun npWhile

(npAndOr p181)
(npLogical p197)
(pfWhile p295)

— defun npWhile —

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

defun npForIn

(npEqKey p145)
(npVariable p213)
(npTrap p212)
(npCompMissing p151)
(npBy p199)
(npPush p143)
--- defun npForIn ---

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

---

defun npReturn

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

---
defun npVoid

[npAndOr p181]
[npStatement p170]
[pfNoValue p279]

— defun npVoid —

(defun |npVoid| ()
  (|npAndOr| 'do #'|npStatement| #'|pfNoValue|))

------

defun npExpress

[npExpress1 p179]
[npIterators p175]
[npPush p143]
[pfCollect p260]
[npPop2 p144]
[pfListOf p245]
[npPop1 p144]

— defun npExpress —

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

------

defun npExpress1

[npConditionalStatement p180]
[npADD p158]

— defun npExpress1 —

(defun |npExpress1| ()


(or (npConditionalStatement) (npADD)))

——

defun npConditionalStatement

[npConditional p195]
[npQualifiedDefinition p147]

—— defun npConditionalStatement ——

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

——

defun npImport

[npAndOr p181]
[npQualTypelist p180]
[pfImport p271]

—— defun npImport ——

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

——

defun npQualTypelist

[npPC p??]
[npSQualTypelist p181]
[upPush p143]
[pfUnSequence p293]
[npPop1 p144]

—— defun npQualTypelist ——

(defun npQualTypelist ()
  (and
   (npPC #\'npSQualTypelist)))
defun npSQualTypelist

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

---

defun npQualType

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

---

defun npAndOr

(defun npAndOr ()
  (npEqKey))
--- defun npAndOr ---

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

---

defun npEncAp

(defun npEncAp (f)
  (and (apply f nil) (npAnyNo #'npEncl) (npFromdom)))

---

defun npEncl

(defun npEncl ()
  (and
   (npBDefinition)
   (npPush (pfApplication (npPop2) (npPop1)))))

defun npAtom1

(defun |npAtom1| ()
  (or
    (|npPDefinition|)
    (and
      (or (|npName|) (|npConstTok|) (|npDollar|) (|npBDefinition|))
      (|npFromdom|))))

defun npPDefinition

(defun |npPDefinition| ()
  (and
    (|npParenthesized| #'|npDefinitionlist|)
    (|npPush| (|pfEnSequence| (|npPop1|))))

defun npDollar

(defun |npDollar| ()
  (and
    (|npEqPeek| #'|npDefinitionlist|)
    (|npPush| (|pfEnSequence| (|npPop1|))))
defun npDollar

(defun npDollar ()
  (declare (special $stok))
  (and (npEqPeek '$)
    (progn
      (npPush (tokConstruct 'id ' (npPosn $stok)))
      (npNext)))))

defun npConstTok

(defun npConstTok ()
  (let (b a)
    (declare (special $stok))
    (cond
      ((member (tokType $stok) '(integer string char float command))
        (npPush $stok)
        (npNext))
      ((npEqPeek |)
        (setq a $stok)
        (setq b (npState))
        (npNext)
        (cond
          ((and (npPrimary1))
            (npPush (pfSymb (npPop1) (npPosn a))))
          t)
        (t (npRestore b) nil)))
    (t nil))))
defun npBDefinition

(defun npBDefinition () (or (npPDefinition) (npBracketed #'npDefinitionlist)))

defun npBracketed

(defun npBracketed (f) (or (npParened f) (npBracked f) (npBraced f) (npAngleBared f)))

defun npParened

(defun npParened (f) (f))
(or (npEnclosed '('|' | ')') #|pfParen| f)
  (npEnclosed ''|'|' |')') #|pfParen| f)))

——

defun npBracked

[ npEnclosed p211 ]
[ pfBracket p257 ]
[ pfBracketBar p257 ]

— defun npBracked —

(defun npBracked (f)
  (or (npEnclosed '[' ']' '#|pfBracket| f)
       (npEnclosed '{' '{' '#|pfBraceBar| f))
)

——

defun npBraced

[ npEnclosed p211 ]
[ pfBrace p257 ]
[ pfBraceBar p257 ]

— defun npBraced —

(defun npBraced (f)
  (or (npEnclosed '{' '{' '#|pfBrace| f)
       (npEnclosed '{' '{' '#|pfBraceBar| f))
)

——

defun npAngleBared

[ npEnclosed p211 ]
[ pfHide p269 ]

— defun npAngleBared —

(defun npAngleBared (f)
  (npEnclosed '<'|'|' >') #|pfHide| f))
defun npDefn
[npEqKey p145]
[npPP p209]
[npDef p187]

---

(defun npDefn ()
  (and
    (npEqKey 'defn)
    (npPP #'npDef)))

---

defun npDef
[npMatch p150]
[npCheckItOut p239]
[npPop1 p144]
[npDefTail p194]
[npTrap p212]
[npPop1 p144]
[npPush p143]
[pfDefinition p261]
[pfPushBody p249]

---

(defun npDef ()
  (let (body rt arg op tmp1)
    (when (npMatch)
      ; [op,arg,rt]:= pfCheckItOut(npPop1())
      (setq tmp1 (pfCheckItOut (npPop1)))
      (setq op (car tmp1))
      (setq arg (cadr tmp1))
      (setq rt (caddr tmp1))
      (or (npDefTail) (npTrap))
      (setq body (npPop1))
      (if (null arg)
        (npPush (pfDefinition op body))
        (npPush (pfDefinition op (pfPushBody rt arg body)))))))
defun npBPileDefinition

(defun npBPileDefinition ()
  (and
   (npPileBracketed #'npPileDefinitionlist)
   (npPush (pfSequence (pfListOf (npPop1))))))

----

defun npPileBracketed

(defun npPileBracketed (f)
  (cond
   ((npEqKey 'settab)
    (cond
     ((npEqKey 'backtab) (npPush (pfNothing))) ; never happens
     ((and (apply f nil)
           (or (npEqKey 'backtab) (npMissing 'backtab)))
      (npPush (pfPile (npPop1)))
      (t nil)))
    (t nil)))

----
defun npPileDefinitionlist

[npListAndRecover p189]
[npDefinitionlist p193]
[npPush p143]
[npAppend p255]
[npPop1 p144]

— defun npPileDefinitionlist —

(defun npPileDefinitionlist ()
 (and
  (npListAndRecover #'npDefinitionlist)
  (npPush (npAppend (npPop1)))))

defun npListAndRecover

[trappoint p??]
[npRecoverTrap p190]
[syGeneralErrorHere p192]
[npEqKey p145]
[npEqPeek p152]
[npNext p145]
[npPop1 p144]
[npPush p143]
[$inputStream p??]
[$stack p??]

— defun npListAndRecover —

(defun npListAndRecover (f)
 (let (found c done b savestack)
   (declare (special |$inputStream| |$stack|))
   (setq savestack |$stack|)
   (setq |$stack| nil)
   (setq c |$inputStream|)
   (do ()
     (cond
      ((eq found 'trapped)
       (setq |$inputStream| c)
       (npRecoverTrap)))
      ((null found)
       (done)
       (setq found (catch 'trappoint (apply f nil))))
   ))
(setq $inputStream| c)
  (syGeneralErrorHere|) (npRecoverTrap|))
  (cond
    ((npEqKey| 'backset) (setq c $inputStream|))
    ((npEqPeek| 'backtab) (setq done t))
    (t
      (setq $inputStream| c)
      (syGeneralErrorHere|)
      (npRecoverTrap|)
      (cond
        ((npEqPeek| 'backtab) (setq done t))
        (t
          (npNext|)
          (setq c $inputStream|))))))
  (setq b (cons (npPop1|) b)))
  (setq $stack| savestack)
  (npPush| (nreverse b))))

---

defun npRecoverTrap

[npFirstTok p143]
[tokPosn p413]
[npMoveTo p191]
[syIgnoredFromTo p191]
[npPush p143]
[pfWrong p296]
[pfDocument p246]
[pfListOf p245]
[$stok p??]

--- defun npRecoverTrap ---

(defun npRecoverTrap| ()
  (let (pos2 posi)
    (declare (special "$stok|))
      (npFirstTok|))
    (setq posi (|tokPosn| "$stok|))
      (npMoveTo| 0)
    (setq pos2 (|tokPosn| "$stok|))
      (syIgnoredFromTo| posi pos2)
    (npPush| (list (pfWrong| (pfDocument| (list "pile syntax error"))
                     (pfList0f| nil)))))
defun npMoveTo

(npEqPeek p152)
(npNext p145)
(npMoveTo p191)
(npEqKey p145)
($inputStream p??]

— defun npMoveTo —

(defun |npMoveTo| |n|)
(declare (special |$inputStream|))
(cond
  ((null |$inputStream|) t)
  ((|npEqPeek| 'backtab)
   (cond
    ((eql |n| 0) t)
    (t (|npNext|) (|npMoveTo| (1- |n|)))))
  ((|npEqPeek| 'backset)
   (cond
    ((eql |n| 0) t)
    (t (|npNext|) (|npMoveTo| |n|))))
  ((|npEqKey| 'settab) (|npMoveTo| (+ |n| 1)))
  (t (|npNext|) (|npMoveTo| |n|))))

defun syIgnoredFromTo

[pfGlobalLinePosn p235]
[ncSoftError p351]
[FromTo p380]
[From p380]
[To p380]

— defun syIgnoredFromTo —

(defun |syIgnoredFromTo| |pos1 pos2|
(cond
  ((equal (|pfGlobalLinePosn| |pos1|) (|pfGlobalLinePosn| |pos2|))
   (|incSoftError| (|FromTo| |pos1 pos2| 'S2CY0005 nil))
   (t
    (|incSoftError| (|From| |pos1|) 'S2CY0003 nil)
    (|incSoftError| (|To| |pos2|) 'S2CY0004 nil))))
defun syGeneralErrorHere

[sySpecificErrorHere p192]

— defun syGeneralErrorHere —

(defun |syGeneralErrorHere| ()
  (|sySpecificErrorHere| 'S2CY0002 nil))

defun sySpecificErrorHere

[sySpecificErrorAtToken p192]
[$stok p??]

— defun sySpecificErrorHere —

(defun |sySpecificErrorHere| (key args)
  (declare (special |$stok|))
  (|sySpecificErrorAtToken| |$stok| key args))

defun sySpecificErrorAtToken

[ncSoftError p351]
[tokPosn p413]

— defun sySpecificErrorAtToken —

(defun |sySpecificErrorAtToken| (tok key args)
  (|ncSoftError| (|tokPosn| tok) key args))
defun npDefinitionlist

[npSemiListing p193]
[npQualDef p145]

— defun npDefinitionlist —

(defun |npDefinitionlist| ()
  (|npSemiListing| #'|npQualDef|))

-----

defun npSemiListing

[npListofFun p221]
[npSemiBackSet p193]
[pfAppend p255]

— defun npSemiListing —

(defun |npSemiListing| (p)
  (|npListofFun| p #'|npSemiBackSet| #'|pfAppend|))

-----

defun npSemiBackSet

[npEqKey p145]

— defun npSemiBackSet —

(defun |npSemiBackSet| ()
  (and (|npEqKey| 'semicolon) (or (|npEqKey| 'backset) t)))

-----

defun npRule

[npEqKey p145]
[npPP p209]
[npSingleRule p194]

— defun npRule —
(defun npRule ()
  (and
   (npEqKey 'rule)
   (npPP #\'npSingleRule)))

---

defun npSingleRule

[npQuiver p198]
[npDefTail p194]
[npTrap p212]
[npPush p143]
[pfRule p285]
[npPop2 p144]
[npPop1 p144]

---

defun npSingleRule —

(defun npSingleRule ()
  (when (npQuiver)
    (or (npDefTail) (npTrap))
    (npPush (pfRule (npPop2) (npPop1))))))

---

defun npDefTail

[npEqKey p145]
[npDefinitionOrStatement p147]

---

defun npDefTail —

(defun npDefTail ()
  (and
   (or (npEqKey 'def) (npEqKey 'mdef)
    (npDefinitionOrStatement))))

---

defun npDefaultValue

[npEqKey p145]
[npDefinitionOrStatement p147]
— defun npDefaultValue —

(defun npDefaultValue ()
  (and
    (npEqKey 'default)
    (or (npDefinitionOrStatement) (npTrap))
    (npPush (list (pfAdd (pfNothing) (npPop1) (pfNothing))))))

— defun npWConditional —

(defun npWConditional (f)
  (when (npConditional f) (npPush (pfTweakIf (npPop1)))))

— defun npConditional —

(defun npConditional (f)
  (cond
    ((and (npEqKey 'IF)
      (or (npLogical) (npTrap)))
      (npConditional f) (npPush (pfTweakIf (npPop1))))))
(or (|npEqKey| 'backset) t))
(cond
 ((|npEqKey| 'settab)
  (cond
   ((|npEqKey| 'then)
    (and (or (apply f nil) (|npTrap|))
      (|npElse| f)
      (|npEqKey| 'backtab)))
    (t (|npMissing| '|then|))))
   ((|npEqKey| 'then)
    (and (or (apply f nil) (|npTrap|)) (|npElse| f)))
    (t (|npMissing| '|then|))))
 (t nil)))

---

**defun npElse**

|npState p212|
|npBacksetElse p197|
|npTrap p212|
|npPush p143|
|pfIf p269|
|npPop3 p144|
|npPop2 p144|
|npPop1 p144|
|upRestore p152|
|pfIfThenOnly p270|

---

---

(defun |npElse| (f)
  (let (a)
    (setq a (|npState|))
    (cond
     ((|npBacksetElse|)
      (and
       (or (apply f nil) (|npTrap|))
       (|npPush| (|pfIf| (|npPop3|) (|npPop2|) (|npPop1|))))
      (t
       (|npRestore| a)
       (|npPush| (|pfIfThenOnly| (|npPop2|) (|npPop1|))))))))
defun npBacksetElse

TPDHERE: Well this makes no sense. [npEqKey p145]

— defun npBacksetElse —

(defun npBacksetElse ()
  (if (npEqKey 'backset)
      (npEqKey 'else)
      (npEqKey 'else)))

defun npLogical

[npLeftAssoc p206]
[npDisjand p197]

— defun npLogical —

(defun npLogical ()
  (npLeftAssoc '(or) #'npDisjand))

defun npDisjand

[npLeftAssoc p206]
[npDiscrim p197]

— defun npDisjand —

(defun npDisjand ()
  (npLeftAssoc '(and) #'npDiscrim))

defun npDiscrim

[npLeftAssoc p206]
[npQuiver p198]

— defun npDiscrim —
(defun npDiscrim ()
  (|npLeftAssoc| '(case has) #'|npQuiver|)))

------

defun npQuiver

[npRightAssoc p206]
[npRelation p198]

  — defun npQuiver —

(defun npQuiver ()
  (|npRightAssoc| '(arrow larrow) #'|npRelation|))

------

defun npRelation

[npLeftAssoc p206]
[npSynthetic p198]

  — defun npRelation —

(defun npRelation ()
  (|npLeftAssoc| '(equal notequal lt le gt ge oangle cangle) #'|npSynthetic|)))

------

defun npSynthetic

[npBy p199]
[npAmpersandFrom p202]
[npPush p143]
[pfApplication p253]
[npPop2 p144]
[npPop1 p144]
[pfInfApplication p271]

  — defun npSynthetic —

(defun npSynthetic ()
  ()

  — defun npSynthetic —

(defun npSynthetic ()
  ()
(cond
  ((|npBy|)
   ((|lambda| ()
     (loop
       (cond
         ((not (and (|npAmpersandFrom|)
                      (or (|npBy|)
                           (progn
                            (|npPush| (|pfApplication| (|npPop2|) (|npPop1|)) nil)))
           (return nil))
         t
          (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|))))))))
  (t nil)))

——

defun npBy
[|npLeftAssoc p206|
[|npInterval p199|

   — defun npBy —

(defun |npBy| ()
  (|npLeftAssoc| '(by) #'|npInterval|))

——

defun
[|npArith p200|
[|npSegment p200|
[|npEqPeek p152|
[|npPush p143|
[|pfApplication p253|
[|npPop1 p144|
[|pfInfApplication p271|
[|npPop2 p144|

   — defun npInterval —

(defun |npInterval| ()
(and
  (npArith))
(or
  (and
    (npSegment))
  (or
    (and
      (npEqPeek 'bar)
      (npPush ((pfApplication (npPop1) (npPop1)))
      (and
        (npArith)
        (npPush ((pfInfApplication (npPop2) (npPop2) (npPop1)))
        (npPush ((pfApplication (npPop1) (npPop1))))
      t))))

---

defun npSegment

[npEqPeek p152]
[npPushId p209]
[npFromdom p202]

   — defun npSegment —

(defun npSegment ()
  (and (npEqPeek 'seg) (npPushId) (npFromdom)))

---

defun npArith

[npLeftAssoc p206]
[npSum p201]

   — defun npArith —

(defun npArith ()
  (npLeftAssoc '(mod) #'+npSum))

---
defun npSum
    [npLeftAssoc p206]
    [npTerm p201]

    — defun npSum —

    (defun |npSum| ()
      (|npLeftAssoc| '(plus minus) #'|npTerm|)))

    — defun npTerm —

    (defun |npTerm| ()
      (or
        (and
          (|npInfGeneric| '(minus plus))
          (or
            (and (|npRemainder|) (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))))
            t))
          (|npRemainder|))))

    — defun npRemainder —

    (defun |npRemainder| ()
      (|npLeftAssoc| '(rem quo) #'|npProduct|)))
defun npProduct

[npLeftAssoc p206]
[npPower p202]

— defun npProduct —

(defun npProduct ()
  (npLeftAssoc
   '(times slash backslash slashslash backslashbackslash
     slashbackslash backslashslash)
   #'|npPower|)))

——

defun npPower

[npRightAssoc p206]
[npColon p217]

— defun npPower —

(defun npPower ()
  (npRightAssoc '(power carat) #'npColon)))

——

defun npAmpersandFrom

[npAmpersand p204]
[npFromdom p202]

— defun npAmpersandFrom —

(defun npAmpersandFrom ()
  (and (npAmpersand) (npFromdom)))

——

defun npFromdom

[npEqKey p145]
[npApplication p162]
(defun npFromdom1 (c)
  (or
    (and
      (npEqKey '$_$)
      (or (npApplication) (npTrap))
      (npFromdom1 (npPop1))
      (npPush (pfFromDom (npPop1) (npPop1))))
    (npPush c)))
defun npAmpersand

(npEqKey p145)
(npName p204)
(npTrap p212)

— defun npAmpersand —

(defun npAmpersand ()
  (and
   (npEqKey 'ampersand)
   (or (npName) (npTrap)))))

defun npName

(npId p204)
(npSymbolVariable p205)

— defun npName —

(defun npName ()
  (or (npId) (npSymbolVariable)))))

defvar $npTokToNames

— initvars —

(defvar $npTokToNames (list '~ '¬ '# '□ '‘ '­ '[] '{} '\{}\{}))

defun npId

(npPush p143)
(npNext p145)
(tokConstruct p411)
(tokPosn p413)
(defun npId ()
  (declare (special |npTokToNames| |ttok| |stok|))
  (cond
   ((eq (caar |stok|) '|id|)
    (|npPush| |stok|)
    (|npNext|))
   ((and (eq (caar |stok|) '|key|) (member |ttok| |npTokToNames|))
    (|npPush| (|tokConstruct| '|id| |ttok| (|tokPosn| |stok|)))
    (|npNext|))
   (t nil)))

(defun npSymbolVariable ()
  (let (a)
    (setq a (|npState|))
    (cond
     ((and (|npEqKey| 'backquote) (|npId|))
      (setq a (|npPop1|))
      (|npPush| (|tokConstruct| '|idsy| (|tokPart| a) (|tokPosn| a))))
     (t (|npRestore| a) nil))))
defun npRightAssoc

(defvar npState 212)
(defvar npInfGeneric 207)
(defvar npRightAssoc 206)
(defvar npPush 143)
(defvar pfApplication 253)
(defvar npPop2 144)
(defvar npPop1 144)
(defvar pfInfApplication 271)
(defvar npRestore 152)

(defun npRightAssoc (o p)
  (let (a)
    (setq a (npState))
    (cond
      ((apply p nil)
       (lambda ()
        (loop
         (cond
           (not
            (and
             (npInfGeneric o)
             (or
              (npRightAssoc o p)
              (progn (npPush (pfApplication (npPop2) (npPop1))) nil)))
            (return nil))
           (t
            (npPush (pfInfApplication (npPop2) (npPop2) (npPop1))))))
        t)
      (t
       (npRestore a)
       nil))))

(defun p o p o p o p = (((p o p) o p) o p)
  p o p o p o p = (((p o p) o p) o p)
  p o p o = (p o p) o
  ;npLeftAssoc(operations,parser)==
  ; if APPLY(parser,nil)
  ; then
  ;   while npInfGeneric(operations)
  ;     and (APPLY(parser,nil) or
;         (npPush pfApplication(npPop2(),npPop1());false))
;     repeat
;       npPush pfInfApplication(npPop2(),npPop2(),npPop1())
;    true
;  else false

[npInfGeneric p207]
[npPush p143]
[pfApplication p253]
[npPop2 p144]
[npPop1 p144]
[pfInfApplication p271]

— defun npLeftAssoc —

(defun |npLeftAssoc| (operations parser)
  (when (apply parser nil)
    ((lambda nil
       (loop
        (cond
         ((not
            (and
             (npInfGeneric| operations)
             (or
              (apply parser nil)
              (progn (npPush| (|pfApplication| (|npPop2|) (|npPop1|)) nil)))
             (return nil))
             (t
              (npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|))))))))
             t))

——

defun npInfGeneric

[npDDInfKey p208]
[npEqKey p145]

— defun npInfGeneric —

(defun |npInfGeneric| (s)
  (and
   (npDDInfKey| s)
   (or (npEqKey| 'backset) t)))

——
defun npDDInfKey

(defvar $stok)

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

(defun npInfKey

(defvar $stok)

(defvar $ttok)

(defun npInfKey (s)
  (npPushId s)
  (npInfKey s)
  (npRestore)
  nil))

—— defun npDDInfKey ——

—— defun npInfKey ——
(defun |npInfKey| (s)
  (declare (special |$ttok| |$stok|))
  (and (eq (caar |$stok|) '|key|) (member |$ttok| s) (npPushId)))

-----

defun npPushId

tokConstruct p411
 tokPosn p413
 npNext p145
 $stack p??
 $stok p??
 $ttok p??

— defun npPushId —

(defun |npPushId| ()
  (let (a)
    (declare (special |$stack| |$stok| |$ttok|))
    (setq a (get |$ttok| 'infgeneric))
    (when a (setq |$ttok| a))
    (setq |$stack|
      (cons (|tokConstruct| '|id| |$ttok| (|tokPosn| |$stok|)) |$stack|)
      (npNext))))

-----

defvar $npPParg

— initvars —

(defvar *npPParg* nil "rewrite npPP without flets, using global scoping")

-----

defun npPP

This was rewritten by NAG to remove flet. npParened p185
 npPff p211
 npPileBracketed p188
--- defun npPP ---

(defun npPP (f)
  (declare (special *npPParg*))
  (setq *npPParg* f)
  (or
   (npParencd '#npPPf)
   (and (npFileBracketed '#npPPg) (npPush (pfEnSequence (npPop1)))
        (funcall f)))

---

defun npPPff

defun npPPg

--- defun npPPg ---

(defun npPPg ()
  (and (npListAndRecover '#npPPf)
       (npPush (pfAppend (npPop1))))

---
defun npPPf

(defun npPPf ()
  (|npSemiListing| #'npPPff))

---

defun npEnclosed

(defun |npEnclosed| (open close fn f)
  (let (a)
    (declare (special |$stok|))
    (setq a |$stok|)
    (when (|npEqKey| open)
      (cond
        ((|npEqKey| close)
         (|npPush| (funcall fn a (|pfTuple| (|pfListOf| NIL))))))
        ((and (apply f nil)
          (or (|npEqKey| close)
              (|npMissingMate| close a)))
         (|npPush| (funcall fn a (|pfEnSequence| (|npPop1|))))
         ('t nil))))

---
defun npState

[$stack p??]
[$inputStream p??]

— defun npState —

(defun npState ()
  (declare (special $stack $inputStream))
  (cons $inputStream $stack))

——

defun npTrap

[trappoint p??]
[tokPosn p413]
[ncSoftError p351]
[$stok p??]

— defun npTrap —

(defun npTrap ()
  (declare (special $stok))
  (ncSoftError (tokPosn $stok) 'S2CY0002 nil)
  (throw 'trappoint 'trapped))

——

defun npTrapForm

[trappoint p??]
[pfSourceStok p243]
[syGeneralErrorHere p192]
[ncSoftError p351]
[tokPosn p413]

— defun npTrapForm —

(defun npTrapForm (x)
  (let (a)
    (setq a (pfSourceStok x))
    (cond
      ((eq a '|NoToken|)
(syGeneralErrorHere)
(throw 'trappoint 'trapped))
(t
  (incSoftError! (tokPosn a) 'S2CY0002 nil)
  (throw 'trappoint 'trapped))))

---

defun npVariable

[npParenthesized p214]
[npVariablelist p213]
[npVariableName p213]
[npPush p143]
[pfListOf p245]
[npPop1 p144]

  — defun npVariable —

(defun npVariable ()
  (or
   (npParenthesized #'npVariablelist)
   (and (npVariableName) (npPush (pfListOf (list (npPop1)))))
   ))

---

defun npVariablelist

[npListing p155]
[npVariableName p213]

  — defun npVariablelist —

(defun npVariablelist ()
  (npListing #'npVariableName))

---

defun npVariableName

[npName p204]
[npDecl p214]
--- defun npVariableName ---

```
(defun npVariableName ()
  (and
   (npName)
   (or (npDecl) (npPush (pfTyped (npPop1) (pfNothing))))))
```

--- defun npDecl ---

```
(defun npDecl ()
  (and
   (npEqKey :colon)
   (or (npType) (npTrap))
   (npPush (pfTyped (npPop2) (npPop1)))))
```

--- defun npParenthesized ---

```
(defun npParenthesized (f)
  (or (npParenthesize '(|' ')| f) (npParenthesize '(|'| '||'| f)))
```
defun npParenthesize

(defvar $stok nil)

(defun npParenthesize (open close f)
  (let (a)
    (declare (special $stok))
    (setq a $stok)
    (cond
      ((npEqKey open)
        (cond
          ((npMissingMate close)
            (npPush nil))
          (t nil))
      (t nil))))

— defun npParenthesize —

defun npMissingMate

(defun npMissingMate (close open)
  (ncSoftError (tokPosn open) 'S2CY0008 nil)
  (npMissing close))

— defun npMissingMate —

defun npExit

(defvar $npBackTrack nil)

(defun npExit (open close)
  (npAssign (npBackTrack) open))

— defun npExit —
defun npPileExit

(defun npPileExit ()
  (and
   (npAssign)
   (or (npEqKey 'exit) (npTrap))
   (or (npStatement) (npTrap))
   (npPush (pfExit (npPop2) (npPop1))))
)

defun npAssign

(defun npAssign ()
  (npBackTrack #'npMDEF 'becomes #'npAssignment))

defun npAssignment

(defun npAssignment ()
  (and
   (npAssignVariable)
   (or (npEqKey 'becomes) (npTrap))
   (or (npGives) (npTrap))
   (npPush (pfAssign (npPop2) (npPop1)))))

---

defun npAssignVariable

(defun npAssignVariable ()
  (and (npColon) (npPush (pfListOf (list (npPop1))))))

---

defun npColon

(defun npColon ()
  (and (npTypified) (npPush (pfListOf (list (npPop1))))))

---
(defun npColon() ()
  (and (npTypified) (npAnyNo #'npTagged)))

--------------

defun npTagged

[npTypedForm1 p218]
[pfTagged p288]

— defun npTagged —

(defun npTagged () ()
  (npTypified (colon #'pfTagged)))

--------------

defun npTypedForm1

[npEqKey p145]
[npType p149]
[npTrap p212]
[npPush p143]
[npPop2 p144]
[npPop1 p144]

— defun npTypedForm1 —

(defun npTypedForm1 (sy fn) ()
  (and
   (npEqKey sy)
   (or (npType) (npTrap))
   (or (npTypified) (npAnyNo) (npTypeStyle))))

--------------

defun npTypified

[npApplication p162]
[npAnyNo p162]
[npTypeStyle p219]

— defun npTypified —
(defun npTypified ()
  (and (npApplication) (npAnyNo #\'npTypeStyle)))

---

defun npTypeStyle

[npCoerceTo p220]
[npRestrict p220]
[npPretend p219]
[npColonQuery p219]

    — defun npTypeStyle —

(defun npTypeStyle ()
  (or (npCoerceTo) (npRestrict) (npPretend) (npColonQuery)))

---

defun npPretend

[npTypedForm p220]
[pfPretend p281]

    — defun npPretend —

(defun npPretend ()
  (npTypedForm 'pretend #\'pfPretend))

---

defun npColonQuery

[npTypedForm p220]
[pfRetractTo p284]

    — defun npColonQuery —

(defun npColonQuery ()
  (npTypedForm 'atat #\'pfRetractTo))

---
defun npCoerceTo
[upTypedForm p220]
[pfCoerceto p259]

--- defun npCoerceTo ---
(defun |npCoerceTo| ()
  (|npTypedForm| 'coerce #'|pfCoerceto|))

---

defun npTypedForm
[upEqKey p145]
[upApplication p162]
[upTrap p212]
[upPush p143]
[upPop2 p144]
[upPop1 p144]

--- defun npTypedForm ---
(defun |npTypedForm| (sy fn)
  (and
    (|npEqKey| sy)
    (or (|npApplication|) (|npTrap|))
    (|npPush| (funcall fn (|npPop2|) (|npPop1|)))))

---

defun npRestrict
[upTypedForm p220]
[pfRestrict p283]

--- defun npRestrict ---
(defun |npRestrict| ()
  (|npTypedForm| 'at #'|pfRestrict|))

---
6.1 macro handling

defun npListoffun

(defun npListoffun (f h g)
  (let (a)
    (declare (special $stack))
    (cond
      ((apply f nil)
       (cond
        ((and (apply h nil) (or (apply f nil) (npTrap)))
         (setq a $stack)
         (setq $stack nil)
         (do ()
          ((not (and (apply h nil)
                       (or (apply f nil) (npTrap))))))
         (setq $stack (cons (nreverse $stack) a))
         (npPush (funcall g (cons (npPop3) (cons (npPop2) (npPop1))))))
       (t t)))
      (t nil))))

6.1 Macro handling

defun phMacro

(defun phMacro (carrier)
  (ncEltQ carrier ptree,...)
  (ncPutQ carrier (ptree, ptreePremacro,...))
  (pform p??)

  — defun phMacro —
(let (ptree)
  (setq ptree (|ncEltQ| carrier '|ptree|))
  (|ncPutQ| carrier '|ptreePremacro| ptree)
  (setq ptree (|macroExpanded| ptree))
  (|ncPutQ| carrier '|ptree| ptree)
  'ok))

---

defun macroExpanded

$macActive is a list of the bodies being expanded. $posActive is a list of the parse forms where the bodies came from. [macExpand p222]

[$posActive p??]
[$macActive p??]

---

defun macroExpanded

(defun |macroExpanded| (pf)
  (let (|$posActive| |$macActive|)
    (declare (special |$posActive| |$macActive|))
    (setq |$macActive| nil)
    (setq |$posActive| nil)
    (|macExpand| pf)))

---

defun macExpand

[pfWhere? p294]
[macWhere p228]
[pfLambda? p273]
[macLambda p228]
[pfMacro? p277]
[macMacro p229]
[pfId? p246]
[macId p227]
[pfApplication? p255]
[macApplication p223]
[pfMapParts p236]
[macExpand p222]

---

defun macExpand
(defun |macExpand| (pf)
  (cond
   ((|pfWhere?| pf)   (|macWhere| pf))
   ((|pfLambda?| pf) (|macLambda| pf))
   ((|pfMacro?| pf)  (|macMacro| pf))
   ((|pfId?| pf)      (|macId| pf))
   ((|pfApplication?| pf) (|macApplication| pf))
   (t (|pfMapParts| #'|macExpand| pf))))

defun macApplication

[pfMapParts p236]
[macExpand p222]
[pfApplicationOp p254]
[pfMLambda? p278]
[pf0ApplicationArgs p237]
[mac0MLambdaApply p223]
[$pfMacros p97]

— defun macApplication —

(defun |macApplication| (pf)
  (let (args op)
    (declare (special |$pfMacros|))
    (setq pf (|pfMapParts| #'|macExpand| pf))
    (setq op (|pfApplicationOp| pf))
    (cond
     ((null (|pfMLambda?| op)) pf)
     (t
      (setq args (|pf0ApplicationArgs| pf))
      (|mac0MLambdaApply| op args pf |$pfMacros|))))

—

defun mac0MLambdaApply

TPDHERE: The pform function has a leading percent sign. fix this [pf0MLambdaArgs p278]
[pfMLambdaBody p279]
[pfSourcePosition p238]
[ncHardError p352]
[pfId? p246]
defun mac0MLambdaApply

(defun mac0MLambdaApply (mlambda args opf |$pfMacros|)
  (declare (special |$pfMacros|))
  (let (pos body params)
    (declare (special |$posActive| |$macActive|))
    (setq params (|pf0MLambdaArgs| mlambda))
    (setq body (|pfMLambdaBody| mlambda))
    (cond
      ((not (eql (length args) (length params)))
        (setq pos (|pfSourcePosition| opf))
        (|ncHardError| pos 'S2CM0003 (list (length params) (length args))))
      (t
        (lambda (parms p arrgs a) ; for p in params for a in args repeat
          (loop
            (cond
              ((or (atom parms)
                  (progn (setq p (car parms)) nil)
                (atom arrgs)
                (progn (setq a (car arrgs)) nil))
              (return nil))
              (t
                (cond
                  (null (|pfId?| p))
                  (setq pos (|pfSourcePosition| opf))
                  (|ncHardError| pos 'S2CM0004 (list (||pform| p))))
                  (t
                    (|mac0Define| (|pfIdSymbol| p) 'mparam a))))))
          params nil args nil)
          (|mac0ExpandBody| body opf |$macActive| |$posActive|))))

---

defun mac0ExpandBody

(defun mac0ExpandBody
  (pfSourcePosition p238)
  (mac0InfiniteExpansion p225)
  (macExpand p222)
6.1. MACRO HANDLING

(defun |mac0ExpandBody| (body opf |$macActive| |$posActive|)
  (declare (special |$macActive| |$posActive|))
  (let (posn pf)
    (cond
      ((member body |$macActive|
        (setq pf (cadr |$posActive|)
           (setq posn (|pfSourcePosition| pf))
           (|mac0InfiniteExpansion| posn body |$macActive|))
      (t
        (setq |$macActive| (cons body |$macActive|))
        (setq |$posActive| (cons opf |$posActive|))
        (|macExpand| body))))

---

defun mac0InfiniteExpansion

TPDHERE: The pform function has a leading percent sign. fix this |mac0InfiniteExpansion,name p226|
[ncSoftError p351]
[pform p??]

---

defun mac0InfiniteExpansion

(declare (special |$macActive| |$posActive|))
  (let (rnames fname tmp1 blist result)
    (setq blist (cons body active))
    (setq tmp1 (mapcar #'|mac0InfiniteExpansion,name| blist))
    (setq fname (car tmp1));[fname, :rnames] := [name b for b in blist]
    (setq rnames (cdr tmp1))
    (|ncSoftError| posn 'S2CM0005
      (list
        (dolist (n (reverse rnames) (nreverse result))
          (setq result (append (reverse (list n "==>")) result)))
        fname ([%pform| body]))
      body)))

---
defun mac0InfiniteExpansion,name

— defun mac0InfiniteExpansion,name 0 —

(defun |mac0InfiniteExpansion,name| (b)
  (let (st sy got)
    (setq got (|mac0GetName| b))
    (cond
      ((null got) "???")
      (t
       (setq sy (car got))
       (setq st (cadr got))
       (if (eq st '|mlambda|)
           (concat (pname sy) "(...)"
                   (pname sy)))))

——

defun mac0GetName

Returns [state, body] or NIL. Returns [sy, state] or NIL. [pfMLambdaBody p279]
[$pfMacros p97]

— defun mac0GetName —

(defun |mac0GetName| (body)
  (let (bd tmp1 st tmp2 sy name)
    (declare (special |$pfMacros|))
    ; for [sy, st, bd] in $pfMacros while not name repeat
    ((lambda (macros tmplist)
      (loop
        (cond
          ((or (atom macros)
            (progn (setq tmplist (car macros)) nil)
               name)
            (return nil))
          (t
           (and (consp tmplist)
            (progn
              (setq sy (car tmplist))
              (setq tmp2 (cadr tmplist))
              (and (consp tmp2)
               (progn
                (setq st (car tmp2))
                ...))))))
      (return (list name body))))
  (return (list state body))))
6.1. MACRO HANDLING

(defun macId
  (pf)
  (let (body state got sy)
    (declare (special |$posActive| |$macActive|)))
    (setq sy (|pfIdSymbol| pf))
    (cond
      ((null (setq got (|mac0Get| sy))) pf)
      (t
       (setq state (car got))
       (setq body (cadr got))
       (cond
        ((eq state '|mparam|) body)
        ((eq state '|mlambda|) (|pfCopyWithPos| body (|pfSourcePosition| pf)))
        (t
         (|pfCopyWithPos|
          (|mac0ExpandBody| body |$macActive| |$posActive|)
          (|pfSourcePosition| pf)))))))
defun mac0Get

[ifcdr p??]
[$pfMacros p97]

— defun mac0Get —

(defun |mac0Get| (sy)
  (declare (special |$pfMacros|))
  (ifcdr (assoc sy |$pfMacros|)))

defun macWhere

[macWhere,mac p228]
[$pfMacros p97]

— defun macWhere —

(defun |macWhere| (pf)
  (declare (special |$pfMacros|))
  (|macWhere,mac| pf |$pfMacros|))

defun macWhere,mac

[pfMapParts p236]
[macExpand p222]
[$pfMacros p97]

— defun macWhere,mac —

(defun |macWhere,mac| (pf |$pfMacros|)
  (declare (special |$pfMacros|))
  (|pfMapParts| #'|macExpand| pf))

defun macLambda

[macLambda,mac p229]
[$pfMacros p97]
6.1. MACRO HANDLING

---

(defun macLambda

)(defun macLambda (pf)
(declare (special $pfMacros))
(|macLambda,mac| pf $pfMacros))

---

defun macLambda,mac

[pfMapParts p236]
[macExpand p222]
[$pfMacros p97]

---

(defun macLambda,mac

)(defun macLambda,mac (pf $pfMacros))
(declare (special $pfMacros))
(|pfMapParts| #'|macExpand| pf))

---

defun Add appropriate definition the a Macro pform

This function adds the definition and returns the original Macro pform. **TPDHERE:** The pform function has a leading percent sign. fix this [pfMacroLhs p277]

[pfMacroRhs p277]
[pfId? p246]
[ncSoftError p351]
[pfSourcePosition p238]
[pfIdSymbol p247]
[mac0Define p230]
[pform p??]
[pfMLambda? p278]
[macSubstituteOuter p230]
[pfNothing? p245]
[pfMacro p277]
[pfNothing p245]

---

(defun macMacro

)(defun macMacro (pf)
(let (sy rhs lhs)
(setq lhs (|pfMacroLhs| pf))
(setq rhs (|pfMacroRhs| pf))
(cond
  ((null (|pfId?| lhs))
   (|ncSoftError| (|pfSourcePosition| lhs) 'S2CM0001 (list (|%pform| lhs)))
    pf)
  (t
   (setq sy (|pfIdSymbol| lhs))
   (|mac0Define| sy
    (cond
     (((|pfMLambda?| rhs) '|mlambda|)
      (t '|mbody|))
     (|macSubstituteOuter| rhs))
    (cond
     (((|pfNothing?| rhs) pf)
      (t (|pfMacro| lhs (|pfNothing|))))))))

---

(defun Add a macro to the global pfMacros list

[|pfMacros p97|]

— defun mac0Define 0 —

(defun |mac0Define| (sy state body)
  (declare (special |$pfMacros|))
  (setq |$pfMacros| (cons (list sy state body) |$pfMacros|)))

---

(defun macSubstituteOuter

[|mac0SubstituteOuter p231|]
[|macLambdaParameterHandling p231|]

— defun macSubstituteOuter —

(defun |macSubstituteOuter| (pform)
  (|mac0SubstituteOuter| (|macLambdaParameterHandling| nil pform) pform))

---
defun mac0SubstituteOuter

(let (tmplist)
  (cond
    ((pfId? pform) (macSubstituteId replist pform))
    ((pfLeaf? pform) pform)
    ((pfLambda? pform)
      (setq tmplist (macLambdaParameterHandling replist pform))
      (dolist (p (pfParts pform)) (mac0SubstituteOuter tmplist p))
      pform)
    (t
      (dolist (p (pfParts pform)) (mac0SubstituteOuter replist p))
      pform)))))

---

defun macLambdaParameterHandling

(let (parlist symlist result)
(cond
  ((|pfLeaf?| pform) nil)
  ((|pfLambda?| pform) ; remove (identifier . replacement) from assoclist
   (setq parlist (mapcar #'|pfTypedId| (|pf0LambdaArgs| pform)))
   (setq symlist (mapcar #'|pfIdSymbol| parlist))
   (dolist (par symlist)
     (setq replist
       (let ((pr (assoc par replist :test #'equal)))
         (when pr (remove par replist :test #'equal))))
     replist)
  ((|pfMLambda?| pform) ; construct assoclist (identifier . replacement)
   (setq parlist (|pf0MLambdaArgs| pform)) ; extract parameter list
   (dolist (par parlist (nreverse result))
     (push
      (cons (|pfIdSymbol| par)
        (|pfLeaf| (|pfAbSynOp| par) (gensym) (|pfLeafPosition| par)))
        result)))
  (t
   (dolist (p (|pfParts| pform))
     (|macLambdaParameterHandling| replist p))))

(defun macSubstituteId
  [pfIdSymbol p247]
  — defun macSubstituteId —
  (defun |macSubstituteId| (replist pform)
    (let (ex)
      (setq ex (assoc (|pfIdSymbol| pform) replist :test #'eq))
      (cond
        (ex
          (rplaca pform (cadr ex))
          (rplacd pform (caddr ex))
          pform)
        (t pform)))))

—
Chapter 7

Pftrees

7.1 Abstract Syntax Trees Overview

The functions create and examine abstract syntax trees. These are called pforms, for short.

The pform data structure

- Leaves: [hd, tok, pos] where pos is optional
- Trees: [hd, tree, tree, ...]
- hd is either an id or (id . alist)

The leaves are:

- char := (char expr position)
- Document := (Document expr position)
- error := (error expr position)
- expression := (expression expr position)
- float := (float expr position)
- id := (id expr position)
- idsy := (idsy expr position)
- integer := (integer expr position)
- string := (string expr position)
- symbol := (symbol expr position)

The special nodes:

- ListOf := (listOf items)
- Nothing := (nothing)
- SemiColon := (SemiColon (Body: Expr))

The expression nodes:
CHAPTER 7. PFTrees

Add := ('Add (Base: [Typed], Addin: Expr))
And := ('And left right)
Application := ('Application (Op: Expr, Arg: Expr))
Assign := ('Assign (LhsItems: [AssLhs], Rhs: Expr))
Attribute := ('Attribute (Expr: Primary))
Break := ('Break (From: ? Id))
Coerceto := ('Coerceto (Expr: Expr, Type: Type))
Collect := ('Collect (Body: Expr, Iterators: [Iterator]))
ComDefinition := ('ComDefinition (Doc: Document, Def: Definition))
DeclPart
Definition := ('Definition (LhsItems: [Typed], Rhs: Expr))
DefinitionSequence := (Args: [DeclPart])
Do := ('Do (Body: Expr))
Document := ('Document strings)
DWhere := ('DWhere (Context: [DeclPart], Expr: [DeclPart]))
EnSequence :=
Export := ('Export (Items: [Typed]))
Forin := ('Forin (Lhs: [AssLhs], Whole: Expr))
Free := ('Free (Items: [Typed]))
Fromdom := ('Fromdom (What: Id, Domain: Type))
Hide := ('hide, arg)
If := ('If (Cond: Expr, Then: Expr, Else: ? Expr))
Import := ('Import (Items: [QualType]))
Inline := ('Inline (Items: [QualType]))
Iterate := ('Iterate (From: ? Id))
Lambda := ('Lambda (Args: [Typed], Rets: ReturnedTyped, Body: Expr))
Literal
Local := ('Local (Items: [Typed]))
Loop := ('Loop (Iterators: [Iterator]))
Macro := ('Macro (Lhs: Id, Rhs: ExprorNot))
MLambda := ('MLambda (Args: [Id], Body: Expr))
Not := ('Not arg)
Novalue := ('Novalue (Expr: Expr))
Or := ('Or left right)
Pretend := ('Pretend (Expr: Expr, Type: Type))
QualType := ('QualType (Type: Type, Qual: ? Type))
Restrict := ('Restrict (Expr: Expr, Type: Type))
Retract := ('RetractTo (Expr: Expr, Type: Type))
ReturnTyped := ('returntyuped (type body))
Rule := ('Rule (lhsitems, rhsitems))
Sequence := ('Sequence (Args: [Expr]))
Suchthat := ('Suchthat (Cond: Expr))
Symb := if leaf then symbol else expression
Tagged := ('Tagged (Tag: Expr, Expr: Expr))
TLambda := ('TLambda (Args: [Typed],
Rets: ReturnedTyped Type, Body: Expr))
Tuple := ('Tuple (Parts: [Expr]))
Typed := ('Typed (Id: Id, Type: ? Type))
Typing := ('Typing (Items: [Typed]))
Until := ('Until (Cond: Expr) NOT USED
WDeclare := ('WDeclare (Signature: Typed, Doc: ? Document))
Where := ('Where (Context: [DeclPart], Expr: Expr))
While := ('While (Cond: Expr))
With := ('With (Base: [Typed], Within: [WithPart]))
Special cases of expression nodes are:

- **Application.** The Op parameter is one of `and`, `or`, `Y`, `1`, `{}`, `[]`, `{{{}}`, `[][]`
- **DeclPart.** The comment is attached to all signatutres in Typing, Import, Definition, Sequence, DWhere, Macro nodes
- **EnSequence.** This is either a Tuple or Sequence depending on the argument
- **Literal.** One of integer symbol expression one zero char string float of the form `'(expression expr position)`

### 7.2 Structure handlers

#### defun pfGlobalLinePosn

[pfGlobalLinePosn p70]

```lisp
(defun pfGlobalLinePosn (posn)
  (poGlobalLinePosn posn))
```

#### defun pfCharPosn

[pfCharPosn p377]

```lisp
(defun pfCharPosn (posn)
  (poCharPosn posn))
```

#### defun pfLinePosn

[pfLinePosn p361]

```lisp
(defun pfLinePosn (posn)
  (poLinePosn posn))
```
(defun |pfLinePosn| (posn) (|poLinePosn| posn))

---

defun pfFileName

[poFileName p360]

— defun pfFileName —

(defun |pfFileName| (posn) (|poFileName| posn))

---

defun pfCopyWithPos

[pfLeaf? p247] [pfAbSynOp p412] [tokPart p413] [pfTree p252] [pfParts p249] [pfCopyWithPos p236]

— defun pfCopyWithPos —

(defun |pfCopyWithPos| (pform pos) (if (? (|pfLeaf?| pform) (|pfLeaf| (|pfAbSynOp| pform) (|tokPart| pform) pos) (|pfTree| (|pfAbSynOp| pform) (loop for p in (|pfParts| pform) collect (|pfCopyWithPos| p pos))))))

---

defun pfMapParts

[pfLeaf? p247] [pfParts p249] [pfTree p252]
7.2. STRUCTURE HANDLERS

---

(defun pfMapParts (f pform)
  (let (parts1 parts0)
    (if (pfLeaf? pform)
      pform
      (progn
        (setq parts0 (pfParts pform))
        (setq parts1 (loop for p in parts0 collect (funcall f p)))
        (if (reduce #'(lambda (u v) (and u v)) (mapcar #'eq parts0 parts1))
          pform
          (pfTree (pfAbSynOp pform) parts1))))

---

defun pf0ApplicationArgs

(defun pf0ApplicationArgs (pform)
  (pf0FlattenSyntacticTuple (pfApplicationArg pform))

---

(defun pf0FlattenSyntacticTuple

(defun pf0FlattenSyntacticTuple (pform)
  (if (null (pfTuple? pform))
    (list pform)
    (cond
      ((or (atom arg1) (progn (setq p (car arg1)) nil))
       ; [:pf0FlattenSyntacticTuple p for p in pf0TupleParts pform]
       (loop (lambda (arg0 arg1 p)
               (cond
                 ((or (atom arg1) (progn (setq p (car arg1)) nil)))
               (pf0FlattenSyntacticTuple pform)
               (progn (setq p (car arg1)) nil))))
  )

---
(return (reverse arg0)))
(t
  (setq arg0 (append (reverse (pfFlattenSyntacticTuple p)) arg0)))
  (setq arg1 (cdr arg1)))
nil ((pfTupleParts pform) nil)))

---

defun pfSourcePosition

[pfLeaf? p247]
[pfLeafPosition p248]
poNoPosition? p413
[pfSourcePosition p238]
pfParts p238
$napos p26

---

defun pfSourcePosition (form)
(let (pos)
  (declare (special $nopos))
  (cond
    ((pfLeaf? form) (pfLeafPosition form))
    (t
      (setq pos $nopos)
      ((lambda (theparts p) ; for p in parts while poNoPosition? pos repeat
          (loop
            (cond
              ((or (atom theparts)
                  (progn (setq p (car theparts)) nil)
                (not ($poNoPosition? pos)))
              (return nil))
              (t (setq pos (pfSourcePosition p)))
              (setq theparts (cdr theparts))))
            (pfParts form) nil)
      pos))))

---

defun Convert a Sequence node to a list

[pfSequence? p287]
pfSequenceArgs p287
[pfListof p245]
7.2. STRUCTURE HANDLERS

--- defun pfSequenceToList ---

(defun |pfSequenceToList| (x)
  (if (|pfSequence?| x)
      (|pfSequenceArgs| x)
      (|pfListOf| (list x))))

---

defun pfSpread

[pfTyped p290]

--- defun pfSpread ---

(defun |pfSpread| (arg1 arg2)
  (mapcar #'(lambda (i) (|pfTyped| i arg2)) arg1))

---

defun Deconstruct nodes to lists

[pfTagged? p289]
[pfTaggedExpr p289]
[pfNothing p245]
[pfTaggedTag p289]
[pfId? p246]
[pfListOf p245]
[pfTyped p290]
[pfCollect1? p242]
[pfCollectVariable1 p242]
[pfTuple? p292]
[pf0TupleParts p293]
[pfTaggedToTyped p289]
[pfDefinition? p261]
[pfApplication? p255]
[pfFlattenApp p241]
[pfTaggedToTyped1 p244]
[pfTransformArg p244]
[npTrapForm p212]

--- defun pfCheckItOut ---
(defun pfCheckItOut (x)
  (let (args op ls form rt result)
    (if (pfTagged? x)
      (setq rt (pfTaggedExpr x))
      (setq rt (pfNothing())))
    (if (pfTagged? x)
      (setq form (pfTaggedTag x))
      (setq form x))
    (cond
      ((pfId? form)
        (list (pfListOf (list (pfTyped form rt))) nil rt))
      ((pfCollect1? form)
        (list (pfListOf (list (pfCollectVariable1 form))) nil rt))
      ((pfTuple? form)
        (list (pfListOf
          (dolist (part (pf0TupleParts form) (nreverse result))
            (push (pfTaggedToTyped part) result)))
          nil rt))
      ((pfDefinition? form)
        (list (pfListOf (list (pfTyped form (pfNothing())))) nil rt))
      ((pfApplication? form)
        (setq ls (pfFlattenApp form))
        (setq op (pfTaggedToTyped1 (car ls)))
        (setq args
          (dolist (part (cdr ls) (nreverse result))
            (push (pfTransformArg part) result)))
          (list (pfListOf (list op)) args rt))
      (t (npTrapForm form))))

-----

defun pfCheckMacroOut

[pfId? p246]
[pfApplication? p255]
[pfFlattenApp p241]
[pfCheckId p241]
[pfCheckArg p241]
[npTrapForm p212]

— defun pfCheckMacroOut —

(defun pfCheckMacroOut (form)
  (let (args op ls)
    (cond
      ((pfId? form) (list form nil))
      ((pfApplication? form)
(setq ls (|pfFlattenApp| form))
(setq op (|pfCheckId| (car ls)))
(setq args (mapcar #'|pfCheckArg| (cdr ls)))
(list op args)
(t (|np TrapForm| form)))))

---

defun pfCheckArg

[pfTuple? p292]
[pf0TupleParts p293]
[pfListOf p245]
[pfCheckId p241]

— defun pfCheckArg —

(defun |pfCheckArg| (args)
  (let (argl)
    (if (|pfTuple?| args)
      (setq argl (|pf0TupleParts| args))
      (setq argl (list args)))
    (|pfListOf| (mapcar #'|pfCheckId| argl))))

---

defun pfCheckId

[pfId? p246]
[|np TrapForm p212]

— defun pfCheckId —

(defun |pfCheckId| (form)
  (if (null (|pfId?| form))
    (|np TrapForm| form)
    form))

---

defun pfFlattenApp

[pfApplication? p255]
[pfCollect1? p242]
CHAPTER 7. PF TREES

--- defun pfFlattenApp ---

(defun pfFlattenApp (x)
  (cond
    ((pfApplication? x)
      (cond
        ((pfCollect1? x) (LIST x))
        (t (append (pfFlattenApp (pfApplicationOp x))
                     (pfFlattenApp (pfApplicationArg x)))))
    (t (list x))))

---

--- defun pfCollect1? ---

(defun pfCollect1? (x)
  (let (a)
    (when (pfApplication? x)
      (setq a (pfApplicationOp x))
    (when (pfId? a) (eq (pfIdSymbol a) '\|) )))

---

--- defun pfCollectVariable1 ---

(defun pfCollectVariable1
  (pfApplicationArg p254)
  (pf0TupleParts p293)
  (pfTaggedToTyped p289)
  (pfTyped p290)
  (pfSuch p244)
  (pfTypedId p291)
  (pfTypedType p291)
7.2. STRUCTURE HANDLERS

— defun pfCollectVariable1 —

(defun pfCollectVariable1 (x)
  (let ((id var a)
    (setq a (pfApplicationArg x))
    (setq var (car (pf0TupleParts a)))
    (setq id (pfTaggedToTyped var))
    (pfTyped)
    (pfSuch (pfTypedId id) (cadr (pf0TupleParts a)))
    (pfTypedType id)))))

defun pfPushMacroBody

[pfMLambda p278]
[pfPushMacroBody p243]

— defun pfPushMacroBody —

(defun pfPushMacroBody (args body)
  (if (null args)
    body
    (pfMLambda (car args) (pfPushMacroBody (cdr args) body))))

defun pfSourceStok

[pfLeaf? p247]
[pfParts p249]
[pfSourceStok p243]
[pfFirst p264]

— defun pfSourceStok —

(defun pfSourceStok (x)
  (cond
    ((pfLeaf? x) x)
    ((null (pfParts x)) 'NoToken)
    (t (pfSourceStok (pfFirst x)))))
defun pfTransformArg

(defun pfTransformArg (args)
  (let (arglist result)
    (if (pfTuple? args)
      (setq arglist (pf0TupleParts args))
      (setq arglist (list args)))
    (pfListOf
      (dolist (|i| arglist (nreverse result))
        (push (pfTaggedToTyped1 |i|) result))))

defun pfTaggedToTyped1

(defun pfTaggedToTyped1 (arg)
  (cond
   ((pfCollect1? arg) (pfCollectVariable1 arg))
   ((pfDefinition? arg) (pfTyped arg (pfNothing)))
   (t (pfTaggedToTyped arg))))

defun pfSuch

(defun pfSuch (arg)
  (cond
   (((pfInfApplication? arg) (pfCollectVariable1 arg))
    (pfTaggedToTyped1 arg (pfNothing)))
   (t (pfTaggedToTyped arg))))
7.3. SPECIAL NODES

(defun |pfSuch| (x y)
  (|pfInfApplication| (|pfId| '|||) x y))

7.3 Special Nodes

defun Create a Listof node

[pfTree p252]

— defun pfListOf —

(defun |pfListOfl| (x)
  (|pfTree| '|listOfl| x))

-defun pfNothing-

[pfTree p252]

— defun pfNothing —

(defun |pfNothing| ()
  (|pfTree| '|nothing| nil))

-defun Is this a Nothing node?

[pfAbSynOp? p412]

— defun pfNothing? —

(defun |pfNothing?| (form)
  (|pfAbSynOp?| form '|nothing|))
7.4 Leaves

defun Create a Document node

— defun pfDocument —

(defun pfDocument (strings)
  (|pfLeaf| 'Document strings))

——

defun Construct an Id node

— defun pfId —

(defun pfId (expr)
  (|pfLeaf| 'id expr))

——

defun Is this an Id node?

— defun pfId? —

(defun pfId? (form)
  (or (|pfAbSynOp?| form 'id) (|pfAbSynOp?| form 'idsy)))

——

defun Construct an Id leaf node

— defun pfIdPos —
7.4. LEAVES

(defun |pfIdPos| (expr pos)
  (|pfLeaf| 'id expr pos))

defun Return the Id part
[tokPart p413]

  — defun pfIdSymbol —

(defun |pfIdSymbol| (form)
  (|tokPart| form))

defun Construct a Leaf node
[tokConstruct p411]
[ifcar p??]
[pfNoPosition p414]

  — defun pfLeaf —

(defun |pfLeaf| (x y &rest z)
  (|tokConstruct| x y (or (ifcar z) (|pfNoPosition|))))

defun Is this a leaf node?
pfAbSynOp p412

  — defun pfLeaf? —

(defun |pfLeaf?| (form)
  (member (|pfAbSynOp| form)
    '(|id| |idsy| |symbol| |string| |char| |float| |expression|
      |integer| |Document| |error|)))
defun Return the token position of a leaf node
[tokPosn p413]

— defun pfLeafPosition —
(defun |pfLeafPosition| (form)
  (|tokPosn| form))

defun Return the Leaf Token
[tokPart p413]

— defun pfLeafToken —
(defun |pfLeafToken| (form)
  (|tokPart| form))

defun Is this a Literal node?
[pfAbSynOp p412]

— defun pfLiteral? 0 —
(defun |pfLiteral?| (form)
  (member (|pfAbSynOp| form)
    '(|integer| |symbol| |expression| |one| |zero| |char| |string| |float|)))

defun Create a LiteralClass node
[pfAbSynOp p412]

— defun pfLiteralClass —
(defun |pfLiteralClass| (form)
  (|pfAbSynOp| form))
7.4. LEAVES

defun Return the LiteralString
[tokPart p413]
  — defun pfLiteralString —
  (defun |pfLiteralString| (form)
    (|tokPart| form))

---

defun Return the parts of a tree node
  — defun pfParts 0 —
  (defun |pfParts| (form)
    (cdr form))

---

defun Return the argument unchanged
  — defun pfPile 0 —
  (defun |pfPile| (part)
    part)

---

defun pfPushBody
[pfLambda p273]
[pfNothing p245]
[pfPushBody p249]
  — defun pfPushBody —
  (defun |pfPushBody| (rt args body)
    (cond
      ((null args) body)
defun An S-expression which people can read.
[pfSexpr,strip p250]
| defun pfSexpr |

(defun pfSexpr (pform)
  (pfSexpr,strip pform))

---

defun Create a human readable S-expression

[pfId? p246]
[pfIdSymbol p247]
[pfLiteral? p248]
[pfLiteralString p249]
[pfLeaf? p247]
[tokPart p413]
[pfApplication? p255]
[pfApplicationArg p254]
[pfTuple? p292]
[pf0TupleParts p293]
[pfApplicationOp p254]
[pfSexpr,strip p250]
[pfAbSynOp p412]
[pfParts p249]

---

(defun pfSexpr,strip (pform)
  (let (args a result)
    (cond
      (((pfId? pform) (pfIdSymbol pform))
        (((pfLiteral? pform) (pfLiteralString pform))
          (((pfLeaf? pform) (tokPart pform))
            (((pfApplication? pform) (pfApplicationOp pform)))))))
  (null (cdr args)) (pfLambda (car args) rt body))
(t
  (pfLambda (car args) (pfNothing)
    (pfPushBody rt (cdr args) body))))

(setq a (|pfApplicationArg| pform))
(if (|pfTuple?| a)
  (setq args (|pf0TupleParts| a))
  (setq args (list a)))
(dolist (p (cons (|pfApplicationOp| pform) args) (nreverse result))
  (push (|pfSexpr,strip| p) result)))
(t
  (cons (|pfAbSynOp| pform)
    (dolist (p (|pfParts| pform) (nreverse result))
      (push (|pfSexpr,strip| p) result))))

---

defun Construct a Symbol or Expression node

[|pfLeaf?| p247]
[|pfSymbol| p251]
[tokPart p413]
[ifcar p?]
[|pfExpression| p264]
[|pfSexpr| p250]

— defun pfSymb —

(defun |pfSymb| (expr &REST optpos)
  (if (|pfLeaf?| expr)
    (|pfSymbol| (tokPart expr) (ifcar optpos))
    (|pfExpression| (|pfSexpr| expr) (ifcar optpos))))

---

defun Construct a Symbol leaf node

[|pfLeaf| p247]
[ifcar p?]

— defun pfSymbol —

(defun |pfSymbol| (expr &REST optpos)
  (|pfLeaf| '|symbol| expr (ifcar optpos)))
defun Is this a Symbol node?

[pfAbSynOp? p412]

— defun pfSymbol? —

(defun |pfSymbol?| (form)
  (|pfAbSynOp?| form '|symbol|))

---

defun Return the Symbol part

[tokPart p413]

— defun pfSymbolSymbol —

(defun |pfSymbolSymbol| (form)
  (|tokPart| form))

---

7.5 Trees

defun Construct a tree node

— defun pfTree 0 —

(defun |pfTree| (x y)
  (cons x y))

---

defun Construct an Add node

[pfNothing p245]
[pfTree p252]

— defun pfAdd —
(defun |pfAdd| (pfbase pfaddin &rest addon)
  (let (lhs)
    (if addon
      (setq lhs addon)
      (setq lhs (|pfNothing|)))
    (|pfTree| '|Add| (list pfbase pfaddin lhs))))

---

defun Construct an And node

[|pfTree| p252]

— defun pfAnd —

(defun |pfAnd| (pfleft pfright)
  (|pfTree| '|And| (list pfleft pfright)))

---

defun pfAttribute

[|pfTree| p252]

— defun pfAttribute —

(defun |pfAttribute| (pfexpr)
  (|pfTree| '|Attribute| (list pfexpr)))

---

defun Return an Application node

[|pfTree| p252]

— defun pfApplication —

(defun |pfApplication| (pfop pfarg)
  (|pfTree| '|Application| (list pfop pfarg)))

---
defun Return the Arg part of an Application node

    — defun pfApplicationArg 0 —

    (defun |pfApplicationArg| (pf)
        (caddr pf))

defun Return the Op part of an Application node

    — defun pfApplicationOp 0 —

    (defun |pfApplicationOp| (pf)
        (cadr pf))

defun Is this an And node?

[pfAbSynOp? p412]

    — defun pfAnd? —

    (defun |pfAnd?| (pf)
        (|pfAbSynOp?| pf '|And|))

defun Return the Left part of an And node

    — defun pfAndLeft 0 —

    (defun |pfAndLeft| (pf)
        (cadr pf))
defun Return the Right part of an And node

   — defun pfAndRight 0 —

   (defun \( pfAndRight \) (pf)
     (caddr pf))

---

defun Flatten a list of lists

   — defun pfAppend 0 —

   (defun \( pfAppend \) (list)
     (apply #'append list))

---

defun Is this an Application node?

   [pfAbSynOp? p412]

   — defun pfApplication? —

   (defun \( pfApplication? \) (pf)
     (\( pfAbSynOp? \) pf 'Application))

---

defun Create an Assign node

   [pfTree p252]

   — defun pfAssign —

   (defun \( pfAssign \) (pflhsitems pfrhs)
     (\( pfTree \) 'Assign (list pflhsitems pfrhs)))

---
defun Is this an Assign node?
[pfAbSynOp? p412]

— defun pfAssign? —

(defun |pfAssign?| (pf)
  (|pfAbSynOp?| pf '|Assign|))

-----

defun Return the parts of an LhsItem of an Assign node
[pfParts p249]
[pfAssignLhsItems p256]

— defun pf0AssignLhsItems 0 —

(defun |pf0AssignLhsItems| (pf)
  (|pfParts| (|pfAssignLhsItems| pf)))

-----

defun Return the LhsItem of an Assign node

— defun pfAssignLhsItems 0 —

(defun |pfAssignLhsItems| (pf)
  (cadr pf))

-----

defun Return the RHS of an Assign node

— defun pfAssignRhs 0 —

(defun |pfAssignRhs| (pf)
  (caddr pf))

-----
defun Construct an application node for a brace

(defun |pfBrace| (a part)
  (|pfApplication| (|pfIdPos| '{}' (|tokPosn| a)) part))

defun Construct an Application node for brace-bars

(defun |pfBraceBar| (a part)
  (|pfApplication| (|pfIdPos| '|{||}' (|tokPosn| a)) part))

defun Construct an Application node for a bracket

(defun |pfBracket| (a part)
  (|pfApplication| (|pfIdPos| '[] (|tokPosn| a)) part))

defun Construct an Application node for bracket-bars
(defun \pfBracketBar\ (a part)
  (|pfApplication| (|pfIdPos| '\[
\]| (|tokPosn| a)) part))

---

defun Create a Break node

(defun \pfBreak\ (pffrom)
  (|pfTree| '\Break| (list pffrom)))

---

defun Is this a Break node?

(defun \pfBreak?\ (pf)
  (|pfAbSynOp?| pf '\Break\))

---

defun Return the From part of a Break node

(defun \pfBreakFrom\ 0
  (cadr pf))
defun Construct a Coerceto node

[pfTree p252]

— defun pfCoerceto —

(defun pfCoerceto (pfexpr pftype)
  (list pfexpr pftype))

---

defun Is this a CoerceTo node?

[pfAbSynOp? p412]

— defun pfCoerceto? —

(defun pfCoerceto? (pf)
  (pfAbSynOp? pf 'Coerceto))

---

defun Return the Expression part of a CoerceTo node

— defun pfCoercetoExpr 0 —

(defun pfCoercetoExpr (pf)
  (cadr pf))

---

defun Return the Type part of a CoerceTo node

— defun pfCoercetoType 0 —

(defun pfCoercetoType (pf)
  (caddr pf))

———
defun Return the Body of a Collect node

— defun pfCollectBody 0 —

(defun |pfCollectBody| (pf)
  (cadr pf))

defun Return the Iterators of a Collect node

— defun pfCollectIterators 0 —

(defun |pfCollectIterators| (pf)
  (caddr pf))

defun Create a Collect node

[pfTree p252]

— defun pfCollect —

(defun |pfCollect| (pfbody pfiterators)
  (|pfTree| '|Collect| (list pfbody pfiterators)))

defun Is this a Collect node?

[pfAbSynOp? p412]

— defun pfCollect? —

(defun |pfCollect?| (pf)
  (|pfAbSynOp?| pf '|Collect|))
7.5. TREES

defun pfDefinition

[pfTree p252]

— defun pfDefinition —

(defun |pfDefinition| (pf lhsItems rhs)
  (list pfTree `|Definition| (list lhsItems rhs)))

———

defun Return the Lhs of a Definition node

— defun pfDefinitionLhsItems 0 —

(defun |pfDefinitionLhsItems| (pf)
  (cadr pf))

———

defun Return the Rhs of a Definition node

— defun pfDefinitionRhs 0 —

(defun |pfDefinitionRhs| (pf)
  (caddr pf))

———

defun Is this a Definition node?

[pfAbSynOp? p412]

— defun pfDefinition? —

(defun |pfDefinition?| (pf)
  (list pfAbSynOp? `|Definition|)))

———
defun Return the parts of a Definition node

[pfParts p249]
[pfDefinitionLhsItems p261]

— defun pf0DefinitionLhsItems —

(defun |pf0DefinitionLhsItems| (pf)
  (|pfParts| (|pfDefinitionLhsItems| pf)))

———

defun Create a Do node

[pfTree p252]

— defun pfDo —

(defun |pfDo| (pfbody)
  (|pfTree| 'Do (list pfbody)))

———

defun Is this a Do node?

[pfAbSynOp? p412]

— defun pfDo? —

(defun |pfDo?| (pf)
  (|pfAbSynOp?| pf 'Do))

———

defun Return the Body of a Do node

— defun pfDoBody 0 —

(defun |pfDoBody| (pf)
  (cadr pf))

———
defun Construct a Sequence node

[|pfTuple p292|
[|pfListOf p245|
[|pfSequence p287|]

— defun pfEnSequence —

(defun |pfEnSequence| (a)
 (cond
   ((null a) (|pfTuple| (|pfListOf| a)))
   ((null (cdr a)) (car a))
   (t (|pfSequence| (|pfListOf| a)))))

defun Construct an Exit node

[|pfTree p252|

— defun pfExit —

(defun |pfExit| (pfcond pfexpr)
   (|pfTree| '|Exit| (list pfcond pfexpr)))

defun Is this an Exit node?

[|pfAbSynOp? p412|

— defun pfExit? —

(defun |pfExit?| (pf)
   (|pfAbSynOp?| pf '|Exit|))

defun Return the Cond part of an Exit

— defun pfExitCond 0 —
(defun pfExitCond (pf)
  (cadr pf))

---

defun Return the Expression part of an Exit

— defun pfExitExpr 0 —

(defun pfExitExpr (pf)
  (caddr pf))

---

defun Create an Export node

[pfTree p252]

— defun pfExport —

(defun pfExport (pfitems)
  (pfTree (Export (list pfitems)))

---

defun Construct an Expression leaf node

[pfLeaf p247]

[ifcar p??]

— defun pfExpression —

(defun pfExpression (expr &rest optpos)
  (pfLeaf (expression expr (ifcar optpos)))

---

defun pfFirst

— defun pfFirst 0 —
(defun pfFirst (form)
  (cadr form))

(defun Create an Application Fix node

  (defun pfFix (pf)
    (pfApplication (pfId 'Y) pf))

(defun Create a Free node

  (defun pfFree (pfitems)
    (pfTree 'Free (list pfitems)))

(defun Is this a Free node?

  (defun pfFree? (pf)
    (pfAbSynOp? pf 'Free))
defun Return the parts of the Items of a Free node

(defun pf0FreeItems (pf)
   (pfParts (pfFreeItems pf)))

defun Return the Items of a Free node

(defun pfFreeItems 0 (pf)
   (cadr pf))

defun Construct a Forin node

(defun pfForin (pflhs pfwhole)
   (pfTree 'Forin (list pflhs pfwhole)))

defun Is this a ForIn node?

(defun pfForin? (pf)
   (pfAbSynOp? pf 'Forin)))
defun Return all the parts of the LHS of a ForIn node

(defun pf0ForinLhs (pf)
  (pfParts pf)
  (pfForinLhs pf))

defun Return the LHS part of a ForIn node

(defun pfForinLhs (pf)
  (cadadr pf))

defun Return the Whole part of a ForIn node

(defun pfForinWhole (pf)
  (cadadr pf))

defun pfFromDom

(defun pfFromDom (pf)
  (pfApplication pf)
  (pfApplication pf)
  (pfApplicationOp pf)
  (pfApplicationArg pf)
  (pfFromdom pf))
(defun |pfFromDom| (dom expr)
  (cond
    ((|pfApplication?| expr)
     (|pfApplication|
      (|pfFromdom| (|pfApplicationOp| expr) dom)
      (|pfApplicationArg| expr))
    (t (|pfFromdom| expr dom))))

---

defun Construct a Fromdom node
[pfTree p252]
— defun pfFromdom —

(defun |pfFromdom| (pfwhat pfdomain)
  (|pfTree| 'Fromdom (list pfwhat pfdomain)))

---

defun Is this a Fromdom mode?
[pfAbSynOp? p412]
— defun pfFromdom? —

(defun |pfFromdom?| (pf)
  (|pfAbSynOp?| pf 'Fromdom))

---

defun Return the What part of a Fromdom node

— defun pfFromdomWhat 0 —

(defun |pfFromdomWhat| (pf)
  (cadr pf))
defun Return the Domain part of a Fromdom node

— defun pfFromdomDomain 0 —

(defun pfFromdomDomain (pf)
  (caddr pf))

---

defun Construct a Hide node

[pfTree p252]

— defun pfHide —

(defun pfHide (a part)
  (declare (ignore a))
  (list part)))

———

defun pfIf

[pfTree p252]

— defun pfIf —

(defun pfIf (pfcond pfthen pfelse)
  (list pfcond pfthen pfelse)))

———

defun Is this an If node?

[pfAbSynOp? p412]

— defun pfIf? —

(defun pfIf? (pf)
  (pfAbSynOp? pf))
defun Return the Cond part of an If

— defun pfIfCond 0 —

(defun \[pfIfCond\] (pf)
  (cadr pf))

defun Return the Then part of an If

— defun pfIfThen 0 —

(defun \[pfIfThen\] (pf)
  (caddr pf))

defun pfIfThenOnly

[pfIf p269]
[pfNothing p245]

— defun pfIfThenOnly —

(defun \[pfIfThenOnly\] (pred cararg)
  \[pfIf\] pred cararg \[pfNothing\]))

defun Return the Else part of an If

— defun pfIfElse 0 —

(defun \[pfIfElse\] (pf)
  (cadddr pf))
defun Construct an Import node

(defun |pfImport| (pfitems)
  (|pfTree| 'Import (list pfitems)))

---

defun Construct an Iterate node

(defun |pfIterate| (pffrom)
  (|pfTree| 'Iterate (list pffrom)))

---

defun Is this an Iterate node?

(defun |pfIterate?| (pf)
  (|pfAbSynOp?| pf 'Iterate))

---

defun Handle an infix application

(defun |pfInfApplication|)

— defun pfInfApplication —
(defun |pfApplication| (op left right)
  (cond
    ((eq (|pfIdSymbol| op) '|and|) (|pfAnd| left right))
    ((eq (|pfIdSymbol| op) '|or|) (|pfOr| left right))
    (t (|pfApplication| op (|pfTuple| (|pfListOf| (list left right)))))))

---

(defun Create an Inline node

[pfTree p252]

— defun pfInline —

(defun |pfInline| (pfitems)
  (|pfTree| '|Inline| (list pfitems)))

---

defun pfLam

[pfAbSynOp? p412]
[pfFirst p264]
[pfNothing p245]
[pfSecond p286]
[pfLambda p273]

— defun pfLam —

(defun |pfLam| (variable body)
  (let (bdy rets)
    (if (|pfAbSynOp?| body '|returntyped|)
      (setq rets (|pfFirst| body))
      (setq rets (|pfNothing|)))
    (if (|pfAbSynOp?| body '|returntyped|)
      (setq bdy (|pfSecond| body))
      (setq bdy body))
    (|pfLambda| variable rets bdy)))

---
defun pfLambda

(defun pfLambda pf args prets pbody)
  (pfTree 'Lambda (list pfargs pformats pbody))

---

defun Return the Body part of a Lambda node

---

defun pfLambdaBody 0

(defun pfLambdaBody pf)
  (cadddr pf))

---

defun Return the Rets part of a Lambda node

---

defun pfLambdaRets 0

(defun pfLambdaRets pf)
  (caddr pf))

---

defun Is this a Lambda node?

---

defun pfLambda? pf

(defun pfLambda? pf)
  (pfAbSynOp? pf 'Lambda))
defun Return the Args part of a Lambda node

    — defun pfLambdaArgs 0 —

(defun |pfLambdaArgs| (pf)
  (cadr pf))

defun Return the Args of a Lambda Node

[pfParts p249]
[pfLambdaArgs p274]

    — defun pf0LambdaArgs —

(defun |pf0LambdaArgs| (pf)
  (|pfParts| (|pfLambdaArgs| pf)))

defun Construct a Local node

[pfTree p252]

    — defun pfLocal —

(defun |pfLocal| (pfitems)
  (|pfTree| 'Local (list pfitems)))

defun Is this a Local node?

[pfAbSynOp? p412]

    — defun pfLocal? —

(defun |pfLocal?| (pf)
  (|pfAbSynOp?| pf 'Local)))
defun Return the parts of Items of a Local node

(defun |pfParts| (pf) (|pfLocalItems| pf))

---

defun Return the Items of a Local node

(defun |pfLocalItems| (pf) (cadr pf))

---

defun Construct a Loop node

(defun |pfLoop| (pfiterators) (|pfTree| '"Loop' (list pfiterators)))

---

defun pfLoop1

(defun |pfLoop1| (body) (|pfLoop| (|pfListOf| (list (|pfDo| body)))))
defun Is this a Loop node?

[pfAbSynOp? p112]

— defun pfLoop? —

(defun |pfLoop?| (pf)
  (|pfAbSynOp?| pf '|Loop|))

defun Return the Iterators of a Loop node

— defun pfLoopIterators 0 —

(defun |pfLoopIterators| (pf)
  (cadr pf))

defun pf0LoopIterators

[pfParts p249]
[pf0LoopIterators p276]

— defun pf0LoopIterators —

(defun |pf0LoopIterators| (pf)
  (|pfParts| (|pfLoopIterators| pf)))

defun pfLp

[pfLoop p275]
[pfListOf p245]
[pfDo p262]

— defun pfLp —
(defun pfLp (iterators body)
  (pfLoop (pfListOf (append iterators (list (pfDo body))))))

---

defun Create a Macro node

[pfTree p252]

— defun pfMacro —

(defun pfMacro (pflhs pfrhs)
  (pfTree '|Macro| (list pflhs pfrhs)))

---

defun Is this a Macro node?

[pfAbSynOp? p412]

— defun pfMacro? —

(defun pfMacro? (pf)
  (pfAbSynOp? pf '|Macro|))

---

defun Return the Lhs of a Macro node

— defun pfMacroLhs 0 —

(defun pfMacroLhs (pf)
  (cadr pf))

---

defun Return the Rhs of a Macro node

— defun pfMacroRhs 0 —
(defun |pfMacroRhs| (pf)
  (caddr pf))

defun Construct an MLambda node

[pfTree p252]

— defun pfMLambda —

(defun |pfMLambda| (pfargs pfbody)
  (|pfTree| 'MLambda (list pfargs pfbody)))

defun Is this an MLambda node?

[pfAbSynOp? p412]

— defun pfMLambda? —

(defun |pfMLambda?| (pf)
  (|pfAbSynOp?| pf 'MLambda)))

defun Return the Args of an MLambda

— defun pfMLambdaArgs 0 —

(defun |pfMLambdaArgs| (pf)
  (cadr pf))

defun Return the parts of an MLambda argument

[pfParts p249]

— defun pf0MLambdaArgs —
7.5. TREES

(defun pf0MLambdaArgs (pf)
  (|pfParts| (|pfMLambdaArgs| pf)))

---

defun pfMLambdaBody

— defun pfMLambdaBody 0 —

(defun |pfMLambdaBody| (pf)
  (caddr pf))

---

defun Is this a Not node?

[pfAbSynOp? p412]

— defun pfNot? —

(defun |pfNot?| (pf)
  (|pfAbSynOp?| pf '|Not|))

---

defun Return the Arg part of a Not node

— defun pfNotArg 0 —

(defun |pfNotArg| (pf)
  (cadr pf))

---

defun Construct a NoValue node

[pfTree p252]

— defun pfNoValue —
(defun pfNovalue (pexpr)
  (pfTree 'Novalue (list pexpr)))

---

(defun Is this a Novalue node?
[pfAbSynOp? p412]
  — defun pfNovalue? —
  (defun pfNovalue? (pf)
    (pfAbSynOp? pf 'Novalue)))

---

(defun Return the Expr part of a Novalue node
  — defun pfNovalueExpr 0 —
  (defun pfNovalueExpr 0 (pf)
    (cadr pf)))

---

(defun Construct an Or node
[pfTree p252]
  — defun pfOr —
  (defun pfOr (pfleft pfright)
    (pfTree 'Or (list pfleft pfright)))

---

(defun Is this an Or node?
[pfAbSynOp? p412]
  — defun pfOr? —
7.5. TREES

(defun pfOr? (pf)
  (|pfAbSynOp?| pf 'Or))

defun Return the Left part of an Or node

— defun pfOrLeft 0 —

(defun pfOrLeft (pf)
  (cadr pf))

defun Return the Right part of an Or node

— defun pfOrRight 0 —

(defun pfOrRight (pf)
  (caddr pf))

defun Return the part of a parenthesised expression

— defun pfParen —

(defun pfParen (a part)
  (declare (ignore a))
  part)

defun pfPretend

[pfTree p252]

— defun pfPretend —
(defun pfPretend (pfexpr pftype)  
  (pfTree 'Pretend (list pfexpr pftype)))

---

(defun Is this a Pretend node?
[pfAbSynOp? p412]
  — defun pfPretend? — 
  (defun pfPretend? (pf) 
    (pfAbSynOp? pf 'Pretend)))

---

(defun Return the Expression part of a Pretend node
  — defun pfPretendExpr 0 —
  (defun pfPretendExpr (pf) 
    (cadr pf))

---

(defun Return the Type part of a Pretend node
  — defun pfPretendType 0 —
  (defun pfPretendType (pf) 
    (caddr pf))

---

(defun Construct a QualType node
[pfTree p252]
  — defun pfQualType —

defun Construct a Restrict node

(defun \(\text{pfRestrict}\) \(\text{pftype}\))
(defun \(\text{pfRestrict}\) \(\text{pfexpr}\) \(\text{pftype}\))

defun Is this a Restrict node?

(defun \(\text{pfRestrict?}\) \(\text{pf}\))

defun Return the Expr part of a Restrict node

(defun \(\text{pfRestrictExpr}\) \(\text{pf}\))

defun Return the Type part of a Restrict node
(defun |pfRestrictType| (pf)
  (caddr pf))

---

defun Construct a RetractTo node

[pfTree p252]

— defun pfRetractTo —

(defun |pfRetractTo| (pfexpr pftype)
  (|pfTree| '|RetractTo| (list pfexpr pftype)))

---

defun Construct a Return node

[pfTree p252]

— defun pfReturn —

(defun |pfReturn| (pfexpr pffrom)
  (|pfTree| '|Return| (list pfexpr pffrom)))

---

defun Is this a Return node?

[pfAbSynOp? p412]

— defun pfReturn? —

(defun |pfReturn?| (pf)
  (|pfAbSynOp?| pf '|Return|))

---

defun Return the Expr part of a Return node

— defun pfReturnExpr 0 —
(defun pfReturnExpr (pf)
  (cadr pf))

defun pfReturnNoName
[pfReturn p284]
[pfNothing p245]

  — defun pfReturnNoName —

(defun pfReturnNoName (value)
  (pfReturn value (pfNothing)))

defun Construct a ReturnTyped node
[pfTree p252]

  — defun pfReturnTyped —

(defun pfReturnTyped (type body)
  (pfTree 'returntyped (list type body)))

defun Construct a Rule node
[pfTree p252]

  — defun pfRule —

(defun pfRule (pflhsitems pfrhs)
  (pfTree 'Rule (list pflhsitems pfrhs)))
defun Return the Lhs of a Rule node

— defun pfRuleLhsItems 0 —

(defun |pfRuleLhsItems| (pf)
  (cadr pf))

defun Return the Rhs of a Rule node

— defun pfRuleRhs 0 —

(defun |pfRuleRhs| (pf)
  (caddr pf))

defun Is this a Rule node?

[pfAbSynOp? p412]

— defun pfRule? —

(defun |pfRule?| (pf)
  (|pfAbSynOp?| pf '|Rule|))

defun pfSecond

— defun pfSecond 0 —

(defun |pfSecond| (form)
  (caddr form))
# 7.5. TREES

`defun Construct a Sequence node

[pfTree p252]

— defun pfSequence —

(defun |pfSequence| (pfargs)
  (|pfTree| '|Sequence| (list pfargs)))

———

defun Return the Args of a Sequence node

— defun pfSequenceArgs 0 —

(defun |pfSequenceArgs| (pf)
  (cadr pf))

———

defun Is this a Sequence node?

[pfAbSynOp? p412]

— defun pfSequence? —

(defun |pfSequence?| (pf)
  (|pfAbSynOp?| pf '|Sequence|))

———

defun Return the parts of the Args of a Sequence node

[pfParts p249]
[pfSequenceArgs p287]

— defun pf0SequenceArgs —

(defun |pf0SequenceArgs| (pf)
  (|pfParts| (|pfSequenceArgs| pf)))

———
defun Create a Suchthat node

[pfTree p252]

— defun pfSuchthat —

(defun |pfSuchthat| (pfcond)
  (|pfTree| 'Suchthat\ (list pfcond)))

—

defun Is this a SuchThat node?

[pfAbSynOp? p412]

— defun pfSuchthat? —

(defun |pfSuchthat?| (pf)
  (|pfAbSynOp?| pf 'Suchthat))

—

defun Return the Cond part of a SuchThat node

— defun pfSuchthatCond 0 —

(defun |pfSuchthatCond| (pf)
  (cadr pf))

—

defun Create a Tagged node

[pfTree p252]

— defun pfTagged —

(defun |pfTagged| (pftag pfexpr)
  (|pfTree| 'Tagged\ (list pftag pfexpr)))

——
7.5. TREES

defun Is this a Tagged node?
[pfAbSynOp? p412]
    — defun pfTagged? —
    (defun |pfTagged?| (pf)
      (|pfAbSynOp?| pf 'Tagged)))

    ——

defun Return the Expression portion of a Tagged node

    — defun pfTaggedExpr 0 —
    (defun |pfTaggedExpr| (pf)
      (caddr pf))

    ——

defun Return the Tag of a Tagged node

    — defun pfTaggedTag 0 —
    (defun |pfTaggedTag| (pf)
      (cadr pf))

    ——

defun pfTaggedToTyped

[pfTagged? p289]
[pfTaggedExpr p289]
[pfNothing p245]
[pfTaggedTag p289]
[pfId? p246]
[pfId p246]
[pfTyped p290]
[pfSuch p244]
[pfInfApplication p271]
--- defun pfTaggedToTyped ---

(defun pfTaggedToTyped (arg)
  (let (a form rt)
    (if (pfTagged? arg)
      (setq rt (pfTaggedExpr arg))
      (setq rt (pfNothing)))
    (if (pfTagged? arg)
      (setq form (pfTaggedTag arg))
      (setq form arg))
    (cond
      ((null (pfId? form))
       (setq a (pfId (gensym)))
       (pfTyped (pfSuch a (pfInfApplication (pfId '=} a form)) rt))
      (t (pfTyped form rt))))

---

defun pfTweakIf

[pfIfElse p270]
[pfNothing? p245]
[pfListOf p245]
[pfTree p252]
[pfIfCond p270]
[pfIfThen p270]

--- defun pfTweakIf ---

(defun pfTweakIf (form)
  (let (b a)
    (setq a (pfIfElse form))
    (setq b (if (pfNothing? a) (pfListOf NIL) a))
    (pfTree 'WIf (list (pfIfCond form) (pfIfThen form) b))))

---

defun Construct a Typed node

[pfTree p252]

--- defun pfTyped ---

(defun pfTyped (pfid pftype)
7.5. TREES

(defun pfTyping |pfTree| p)

(defun pfTyped? |pfAbSynOp?| |Typed| (pf)
  (|pfAbSynOp?| pf |Typed|))

(defun pfTypedType |pfAbSynOp?| |Typed| (pf)
  (caddr pf))

(defun pfTypedId |pfAbSynOp?| |Typed| (pf)
  (cadr pf))

(defun Construct a Typing node
  [pfTree p252]
  — defun pfTyping —

(defun Is this a Typed node?
  [pfAbSynOp? p412]
  — defun pfTyped? —
  (defun pfTypedType |pfAbSynOp?| |Typed| (pf)
    (|pfAbSynOp?| pf |Typed|))

(defun Return the Type of a Typed node
  — defun pfTypedType 0 —
  (defun pfTypedType |pfAbSynOp?| |Typed| (pf)
    (caddr pf))

(defun Return the Id of a Typed node
  — defun pfTypedId 0 —
  (defun pfTypedId |pfAbSynOp?| |Typed| (pf)
    (cadr pf))

---
(defun |pfTyping| (pfitems)
  (|pfTree| '|Typing| (list pfitems)))

defun Return a Tuple node
|pfTree p252|

— defun pfTuple —

(defun |pfTuple| (pfparts)
  (|pfTree| '|Tuple| (list pfparts)))

defun Return a Tuple from a List
|pfTuple p292|
|pfListOf p245|

— defun pfTupleListOf —

(defun |pfTupleListOf| (pfparts)
  (|pfTuple| (|pfListOf| pfparts)))

defun Is this a Tuple node?
|pfAbSynOp? p412|

— defun pfTuple? —

(defun |pfTuple?| (pf)
  (|pfAbSynOp?| pf '|Tuple|))
defun Return the Parts of a Tuple node

---

(defun pfTupleParts 0)

(defun (pfTupleParts pf)
  (cadr pf))

---

defun Return the parts of a Tuple

[pfParts p294]
[pfTupleParts p293]

---

(defun pf0TupleParts)

(defun (pf0TupleParts pf)
  (pfParts (pfTupleParts pf)))

---

defun Return a list from a Sequence node

[pfSequence? p287]
[pfAppend p255]
[pf0SequenceArgs p287]
[pfListOf p245]

---

(defun pfUnSequence)

(defun (pfUnSequence x)
  (if (pfSequence? x)
      (pfListOf (pfAppend (pf0SequenceArgs x)))
      (pfListOf x)))

---

defun The comment is attached to all signatutres

[pfWDeclare p294]
[pfParts p249]
defun pfWDec |
(defun pfWDec (doc name)
  (mapcar #'(lambda (i) (pfWDeclare i doc)) (pfParts name)))

defun Construct a WDeclare node
[pfTree p252]

— defun pfWDeclare —
(defun pfWDeclare (pfsignature pfdoc)
  (pfTree 'WDeclare (list pfsignature pfdoc)))

defun Construct a Where node
[pfTree p252]

— defun pfWhere —
(defun pfWhere (pfcontext pfexpr)
  (pfTree 'Where (list pfcontext pfexpr)))

defun Is this a Where node?
[pfAbSynOp? p412]

— defun pfWhere? —
(defun pfWhere? (pf)
  (pfAbSynOp? pf 'Where))
defun Return the parts of the Context of a Where node

(defun pfParts pf)
(defun pfWhereContext pf)

---

— defun pf0WhereContext —

(defun pf0WhereContext pf)

---

defun Return the Context of a Where node

— defun pfWhereContext 0 —

(defun pfWhereContext pf)

---

— defun pfWhereExpr 0 —

(defun pfWhereExpr pf)

---

defun Construct a While node

(defun pfTree pf)
defun Is this a While node?

[pfAbSynOp? p412]

— defun pfWhile? —

(defun |pfWhile?| (pf)
  (|pfAbSynOp?| pf 'While)))

—

defun Return the Cond part of a While node

— defun pfWhileCond 0 —

(defun |pfWhileCond| (pf)
  (cadr pf))

—

defun Construct a With node

[pfTree p252]

— defun pfWith —

(defun |pfWith| (pfbase pfwithin pfwithinon)
  (|pfTree| 'With (list pfbase pfwithin pfwithinon)))

—

defun Create a Wrong node

[pfTree p252]

— defun pfWrong —

(defun |pfWrong| (pfwhy pfrrubble)
  (|pfTree| 'Wrong (list pfwhy pfrrubble)))

—
defun Is this a Wrong node?

[pfAbSynOp? p412]

— defun pfWrong? —

(defun pfWrong (pf)
  (if pfAbSynOp? pf 'Wrong)))
Chapter 8

Pftree to s-expression translation

Pftree to s-expression translation. Used to interface the new parser technology to the interpreter. The input is a parseTree and the output is an old-parser-style s-expression.

defun Pftree to s-expression translation

[pf2Sex1 p300]

(defun pf2Sex | pf2Sex | (pf)
  (let (|$insideSEQ| |$insideApplication| |$insideRule|)
    (declare (special |$insideSEQ| |$insideApplication| |$insideRule| |$QuietCommand|))
    (setq |$QuietCommand| nil)
    (setq |$insideRule| nil)
    (setq |$insideApplication| nil)
    (setq |$insideSEQ| nil)
    (|pf2Sex1| pf)))

— defun pf2Sex —
defun Pftree to s-expression translation inner function

[pfNothing? p245]
[pfSymbol? p252]
[pfSymbolSymbol p252]
[pfLiteral? p248]
[pfLiteral2Sex p304]
[pfIdSymbol p247]
[pfApplication? p255]
[pfApplication2Sex p305]
[pfTuple? p292]
[pf2Sex1 p300]
[pf0TupleParts p293]
[pff? p269]
[pfIfCond p270]
[pfIfThen p270]
[pfIfElse p270]
[pfTagged? p289]
[pfTaggedTag p289]
[pfTaggedExpr p289]
[pfCoerceto? p259]
[pfCoercetoExpr p259]
[pfCoercetoType p259]
[pfPretend? p282]
[pfPretendExpr p282]
[pfPretendType p282]
[pfFromdom? p268]
[opTran p323]
[pfFromdomWhat p268]
[pfFromdomDomain p269]
[pfSequence? p287]
[pfSequence2Sex p310]
[pfExit? p263]
[pfExitCond p263]
[pfExitExpr p264]
[pfLoop? p276]
[loopIters2Sex p311]
[pf0LoopIterators p276]
[pfCollect? p260]
[pfCollect2Sex p314]
[pfForin? p266]
[pfForinLhs p267]
[pfForinWhole p267]
[pfWhile? p296]
[pfWhileCond p296]
[pfSuchthat? p288]
(defun pf2Sex1 (pf)
  (let (args idList type op tagPart tag s)
    (declare (special \$insideSEQ \$insideRule \$QuietCommand))
    (cond
      ((\$pfNothing?$) pf '\noBranch\)
      ((\$pfSymbol?$) pf)
        (if (eq \$insideRule '\left\)
            (progn
              (setq s (\$pfSymbolSymbol$ pf))
              (list '\constant\ (list 'quote s)))
              (list 'quote (\$pfSymbolSymbol$ pf))))
      ((\$pfLiteral?$) pf) (\$pfLiteral2Sex$ pf))
      ((\$pfId?$) pf)
        (if \$insideRule
            (progn
              (setq s (\$pfIdSymbol$ pf))
              (if (member s '(\%pi \%e \%i))
                  s
                (list 'quote s)))
              (\$pfIdSymbol$ pf)))
      ((\$pfApplication?$) pf) (\$pfApplication2Sex$ pf))
      ((\$pfTuple?$) pf) (cons '\Tuple\ (mapcar #'\$pf2Sex1$ (\$pf0TupleParts$ pf))))
      ((\$pfIf?$) pf)
        (list 'if (\$pf2Sex1$ (\$pfIfCond$ pf))
              (\$pf2Sex1$ (\$pfIfThen$ pf))
              (\$pf2Sex1$ (\$pfIfElse$ pf))))
      ((\$pfTagged?$) pf)
        (setq tag (\$pfTaggedTag$ pf))
        (setq tagPart
          (if (\$pfTuple?$ tag)
              (cons '\Tuple\ (mapcar #'\$pf2Sex1$ (\$pf0TupleParts$ tag)))
              (\$pf2Sex1$ tag)))
        (list '[: tagPart (\$pf2Sex1$ (\$pfTaggedExpr$ pf))])
      ((\$pfCoerceto?$) pf)
        (list '[: (\$pf2Sex1$ (\$pfCoercetoExpr$ pf))
              (\$pf2Sex1$ (\$pfCoercetoType$ pf))])
      ((\$pfPretend?$) pf)
        (list '\pretend\ (\$pf2Sex1$ (\$pfPretendExpr$ pf))
              (\$pf2Sex1$ (\$pfPretendType$ pf))))
      ((\$pfFromdom?$) pf)
        (setq op (\$opTran$ (\$pf2Sex1$ (\$pfFromdomWhat$ pf))))
        (when (eq op '\braceFromCurly\) (setq op '\seq\))
(list '|$elt| (list 'pf2Sex1 (list 'pfFromdomDomain pf) op))
((list 'pfSequence? pf) (list 'pfSequence2Sex pf))
((list 'pfExit? pf)
 (if ($insideSEQ)
  (list 'exit (list 'pf2Sex1 (list 'pfExitCond pf))
   (list 'pf2Sex1 (list 'pfExitExpr pf)))
  (list 'pf2Sex1 (list 'pfExitCond pf)
   (list 'pf2Sex1 (list 'pfExitExpr pf) 'noBranch))))
((list 'pfLoop? pf) (cons 'repeat (list 'loopIters2Sex (list 'pf0LoopIterators pf))))
((list 'pfCollect? pf) (list 'pfCollect2Sex pf))
((list 'pfForin? pf)
 (cons 'in
  (append (mapcar #'|pf2Sex1| (list 'pf0ForinLhs pf))
   (list 'pf2Sex1 (list 'pfForinWhole pf))))))
((list 'pfSuchthat? pf)
 (if (eq $insideRule '|left|)
  (keyedSystemError "S2GE0017" (list 'pf2Sex1 pfSuchThat))
  (list '\|| (list 'pf2Sex1 (list 'pfSuchthatCond pf)))))
((list 'pfDo? pf) (list 'pf2Sex1 (list 'pfDoBody pf)))
((list 'pfTyped? pf)
 (setq type (list 'pfTypedType pf))
 (if (list 'pfNothing? type)
  (list 'pf2Sex1 (list 'pfTypedId pf))
  (list ': 'pf2Sex1 (list 'pfTypedId pf) (list 'pf2Sex1 (list 'pfTypedType pf)))))
((list 'pfAssign? pf)
 (setq idList (mapcar #'|pf2Sex1| (list 'pf0AssignLhsItems pf))
 (if (not (eql (length idList) 1))
  (setq idList (cons '|Tuple| idList))
  (setq idList (car idList))))
 (list 'let idList (list 'pfAssignRhs pf)))
((list 'pfDefinition? pf) (list 'pfDefinition2Sex pf))
((list 'pfLambda? pf) (list 'pfLambda2Sex pf))
((list 'pfMLambda? pf) '/throwAway)
((list 'pfRestrict? pf)
 (list 'pfRestrictExpr pf)
 (list 'pfRestrictType pf)))
((list 'pfFree? pf) (cons 'free (mapcar #'|pf2Sex1| (list 'pf0FreeItems pf))))
((list 'pfLocal? pf) (cons 'local (mapcar #'|pf2Sex1| (list 'pf0LocalItems pf))))
((list 'pfWrong? pf) (list 'spadThrow))
((list 'pfAnd? pf)
 (list 'and (list 'pfAndLeft pf)
   (list 'pfAndRight pf)))
((list 'pfOr? pf)
 (list 'or (list 'pfOrLeft pf)
   (list 'pfOrRight pf)))
((list 'pfNot? pf) (list 'not (list 'pfNotArg pf))
 ((list 'pfNovalue? pf)
  (setq $QuietCommand t)
  (list 'seq (list 'pf2Sex1 (list 'pfNovalueExpr pf))))
defun Convert a Literal to an S-expression

(defun pfLiteral2Sex (pf)
  (let (s type)
    (declare (special $insideRule))
    (setq type (pfLiteralClass pf))
    (cond
      ((eq type 'integer!) (read-from-string (pfLiteralString pf)))
      ((or (eq type 'string!) (eq type 'char!))
       (pfLiteralString pf))
      ((eq type 'float!) (float2Sex (pfLiteralString pf)))
      ((eq type 'symbol)
       (if $insideRule
          (progn
            (setq s (pfSymbolSymbol pf))
            (list 'quote s))
          (pfSymbolSymbol pf)))
      ((eq type 'expression) (list 'quote (pfLeafToken pf)))
      (t
       (keyedSystemError "S2GE0017" (list "pf2Sex1")))))

——– defun pfLiteral2Sex ——–
defun Convert a float to an S-expression

(defvar $useBFasDefault t)

(defun float2Sex (num)
  (let (exp frac bfForm fracPartString intPart dotIndex expPart mantPart eIndex)
    (declare (special $useBFasDefault))
    (setq eIndex (search "e" num))
    (if eIndex
      (setq mantPart (subseq num 0 eIndex))
      (setq mantPart num))
    (if eIndex
      (setq expPart (read-from-string (subseq num (+ eIndex 1))))
      (setq expPart 0))
    (setq dotIndex (search "." mantPart))
    (if dotIndex
      (setq intPart (read-from-string (subseq mantPart 0 dotIndex)))
      (setq intPart (read-from-string mantPart)))
    (if dotIndex
      (setq fracPartString (subseq mantPart (+ dotIndex 1))))
    (setq fracPartString 0)
    (setq bfForm
      (make-float intPart (read-from-string fracPartString)
        (length fracPartString) expPart))
    (if $useBFasDefault
      (progn
        (setq frac (cadr bfForm))
        (setq exp (cddr bfForm))
        (list (list '|$elt| (list '|Float|) '|float|) frac exp 10)
      )
    )
    bfForm)))

defun Change an Application node to an S-expression

(defvar pfOp2Sex p308)
(defvar pfApplicationOp p254)
(defvar opTran p323)
(defvar pf0TupleParts p293)
\[
\text{CHAPTER 8. PFTREE TO S-EXPRESSION TRANSLATION}
\]

\[
[\text{pfApplicationArg} \text{ p254}]
[\text{pfTuple?} \text{ p292}]
[\text{pf2Sex1} \text{ p300}]
[\text{pf2Sex} \text{ p299}]
[\text{pfSuchThat2Sex} \text{ p307}]
[\text{hasOptArgs?} \text{ p309}]
[\text{$\$insideApplication} \text{ p??}]
[\text{$\$insideRule} \text{ p??}]
\]

--- \text{defun pfApplication2Sex} ---

\[
\text{(defun |pfApplication2Sex| (pf)}
\] 
\[
\text{ (let ((|\$insideApplication| x val realOp tmp1 qt argSex typeList args op))}
\] 
\[
\text{ (declare (special |\$insideApplication| |\$insideRule|))}
\] 
\[
\text{ (setq |\$insideApplication| t)}
\] 
\[
\text{ (setq op (|pfOp2Sex| (|pfApplicationOp| pf)))}
\] 
\[
\text{ (setq op (|opTran| op))}
\] 
\[
\text{ (cond}
\] 
\[
\text{ ((eq op '->)}
\] 
\[
\text{ (setq args (|pf0TupleParts| (|pfApplicationArg| pf)))}
\] 
\[
\text{ (if (|pfTuple?| (car args))}
\] 
\[
\text{ (setq typeList (mapcar #'|pf2Sex1| (|pf0TupleParts| (car args))))}
\] 
\[
\text{ (setq typeList (list (|pf2Sex1| (car args))))}
\] 
\[
\text{ (setq args (cons (|pf2Sex1| (cadr args)) typeList))}
\] 
\[
\text{ (cons 'Mapping args))}
\] 
\[
\text{ ((and (eq op ':) (eq |\$insideRule| 'left)))}
\] 
\[
\text{ (list 'multiple (|pf2Sex| (|pfApplicationArg| pf))}
\] 
\[
\text{ (and (eq op '?) (eq |\$insideRule| 'left)))}
\] 
\[
\text{ (list 'optional (|pf2Sex| (|pfApplicationArg| pf))))}
\] 
\[
\text{ (t)}
\] 
\[
\text{ (setq args (|pfApplicationArg| pf))}
\] 
\[
\text{ (cond}
\] 
\[
\text{ (|pfTuple?| args)}
\] 
\[
\text{ (if (and (eq op '|\||) (eq |\$insideRule| 'left)))}
\] 
\[
\text{ (|pfSuchThat2Sex| args)}
\] 
\[
\text{ (progn}
\] 
\[
\text{ (setq argSex (cdr (|pf2Sex1| args))))}
\] 
\[
\text{ (cond}
\] 
\[
\text{ ((eq op '>)} (list '< (cadr argSex) (car argSex)))}
\] 
\[
\text{ ((eq op '>=)} (list 'not (list '< (car argSex) (cadr argSex))))}
\] 
\[
\text{ ((eq op '<=)} (list 'not (list '< (cadr argSex) (car argSex))))}
\] 
\[
\text{ ((eq op 'and)} (list 'and (car argSex) (cadr argSex)))}
\] 
\[
\text{ ((eq op 'or)} (list 'or (car argSex) (cadr argSex)))}
\] 
\[
\text{ ((eq op 'iterate)} (list 'iterate))}
\] 
\[
\text{ ((eq op 'by)} (cons 'by argSex))}
\] 
\[
\text{ ((eq op 'braceFromCurly)}(}
\] 
\[
\text{ (if (and (consp argSex) (eq (car argSex) 'seq))}
\] 
\[
\text{ argSex}
\] 
\[
\text{ (cons 'seq argSex)))}
\]
((and (consp op)
  (progn
    (setq qt (car op))
    (setq tmp1 (cdr op))
    (and (consp tmp1)
      (eq (cdr tmp1) nil)
      (progn
        (setq realOp (car tmp1))
        t)))
    (eq qt 'quote))
  (cons 'applyQuote (cons op argSex))))
((setq val (hasOptArgs? argSex)) (cons op val))
(t (cons op argSex)))))))
((and (consp op)
  (progn
    (setq qt (car op))
    (setq tmp1 (cdr op))
    (and (consp tmp1)
      (eq (cdr tmp1) 'quote))
      (list 'applyQuote op (pf2Sex1 args)))
      (eq op 'braceFromCurly)
(x)
(if (and (consp x) (eq (car x) 'seq))
  x
  (list 'seq x)))
((eq op 'by) (list 'by (pf2Sex1 args))
(t (list op (pf2Sex1 args))))))))

---

defun Convert a SuchThat node to an S-expression

[pf0TupleParts p293]
[pf2Sex1 p300]
[pf2Sex p299]
[$predicateList p299]

— defun pfSuchThat2Sex —

(defun pfSuchThat2Sex (args)
  (let (rhsSex lhsSex argList name)
    (declare (special $predicateList))
    (setq name (gentemp)))
(setq argList (|pf0TupleParts| args))
(setq lhsSex (|pf2Sex1| (car argList)))
(setq rhsSex (|pf2Sex| (cadr argList)))
(setq |$predicateList|
  (cons (cons name (cons lhsSex rhsSex)) |$predicateList|))

---

defun pfOp2Sex

[pf2Sex1 p300]
[pmDontQuote? p309]
[pfSymbol? p252]
[$quotedOpList p??]
[$insideRule p??]

— defun pfOp2Sex —

(defun |pfOp2Sex| (pf)
  (let (realOp tmp1 op alreadyQuoted)
    (declare (special |$quotedOpList| |$insideRule|))
    (setq alreadyQuoted (|pfSymbol?| pf))
    (setq op (|pf2Sex1| pf))
    (cond
      ((and (consp op) 'quote)
       (progn
        (setq tmp1 (cdr op))
        (and (consp tmp1)
          (eq (cdr tmp1) nil)
          (progn
            (setq realOp (car tmp1)) t))))
      (cond
        ((eq |$insideRule| '|left|) realOp)
        ((eq |$insideRule| '|right|)
         (cond
          (|pmDontQuote?| realOp) realOp)
          t
          (setq |$quotedOpList| (cons op |$quotedOpList|))
          op))))
    ((eq realOp '|\|) realOp)
    ((eq realOp '|::|) realOp)
    ((eq realOp '?) realOp)
    (t op)))
  (t op)))
defun pmDontQuote?

--- defun pmDontQuote? 0 ---

(defun pmDontQuote? (sy)
  (member sy
    '+ * ** ^ / |log| |exp| |pi| |sqrt| |ei| |li| |erf| |ci|
    |si| |dilog| |sin| |cos| |tan| |cot| |sec| |csc| |asin|
    |acos| |atan| |acot| |asec| |acsc| |sinh| |cosh| |tanh|
    |coth| |sech| |csch| |asinh| |acosh| |atanh| |acoth|
    |asech| |acscl)))

---

defun hasOptArgs?

--- defun hasOptArgs? 0 ---

(defun hasOptArgs? (argSex)
  (let (rhs lhs opt nonOpt tmp1 tmp2)
    (dolist (arg argSex)
      (cond
        ((and (consp arg) (eq (car arg) 'optarg)
            (progn
              (setq tmp1 (cdr arg))
              (and (consp tmp1)
                (progn
                  (setq lhs (car tmp1))
                  (setq tmp2 (cdr tmp1))
                  (and (consp tmp2)
                    (eq (cdr tmp2) nil)
                    (progn
                      (setq rhs (car tmp2))
                      t)))))))
        (setq opt (cons (list lhs rhs) opt)))
    (t (setq nonOpt (cons arg nonOpt))))
  (when opt
    (nconc (nreverse nonOpt) (list (cons '|'construct| (nreverse opt))))))

---
 CHAPTER 8. PF TREE TO S-EXPRESSION TRANSLATION

defun Convert a Sequence node to an S-expression

(defun |pfSequence2Sex| (pf)
  (let ((|$insideSEQ| tmp1 ruleList seq)
  (declare (special |$insideSEQ|))
  (setq |$insideSEQ| t)
  (setq seq ([pfSequence2Sex0] (mapcar #'|pf2Sex1| (|pf0SequenceArgs| pf)))))
  (cond
    ((and (consp seq)
      (eq (car seq) 'seq)
      (progn (setq ruleList (cdr seq)) 't)
      (consp ruleList)
      (progn
        (setq tmp1 (car ruleList))
        (and (consp tmp1) (eq (car tmp1) '|rule|)))
      (list '|ruleset| (cons '|construct| ruleList)))
    (t seq))))

defun pfSequence2Sex0

TPDHERE: rewrite this using (dolist (item seqList)...)
(defun |pfSequence2Sex0| (seqList)
  (let (value tmp2 cond tmp1 item seqTranList)
    (if (null seqList)
      'noBranch
      (progn
        (lambda ()
          (loop
            (if (not seqList)
                (return nil)
                (progn
                  (setq item (car seqList))
                  (cond
                    ((and (consp item)
                      (eq (car item) '|exit|)
                      (progn
                        (setq tmp1 (cdr item))
                        (and (consp tmp1)
                          (progn
                            (setq cond (car tmp1))
                            (setq tmp2 (cdr tmp1))
                            (and (consp tmp2)
                              (eq (cdr tmp2) nil)
                              (progn
                                (setq value (car tmp2))
                                t)))))
                    (setq item
                      (list 'if cond value (|pfSequence2Sex0| (cdr seqList))))))))
          (setq seqTranList (cons item seqTranList))
          (setq seqList (cdr seqList))))))))
    (if (eql (length seqTranList) 1)
      (car seqTranList)
      (cons 'seq (nreverse seqTranList))))))

defun Convert a loop node to an S-expression

TPDHERE: rewrite using dsetq

;loopIters2Sex iterList ==
; result := nil
; for iter in iterList repeat
;   sex := pf2Sex1 iter
;   sex is ['IN, var, ['SEGMENT, i, ['BY", incr]]] =>
CHAPTER 8. PFTREE TO S-EXPRESSION TRANSLATION

; result := ['STEP, var, i, incr, :result]
; sex is ['IN, var, ["BY", ['SEGMENT, i, j], incr]] =>
; result := ['STEP, var, i, incr, j, :result]
; sex is ['IN, var, ['SEGMENT, i, j]] =>
; result := ['STEP, var, i, 1, j, :result]
; result := [sex, :result]
; nreverse result

[pf2Sex1 p300]

— defun loopIters2Sex —

(defun |loopIters2Sex| (iterList)
  (let (j incr i var sex result tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 tmp8)
    (dolist (iter iterList (nreverse result))
      (setq sex (|pf2Sex1| iter))
      (cond
       ((and (consp sex)
            (eq (car sex) 'in)
            (progn
              (setq tmp1 (cdr sex))
              (and (consp tmp1)
                (progn
                  (setq var (car tmp1))
                  (setq tmp2 (cdr tmp1))
                  (and (consp tmp2)
                    (eq (cdr tmp2) nil)
                    (progn
                      (setq tmp3 (car tmp2))
                      (and (consp tmp3)
                        (eq (car tmp3) 'segment)
                        (progn
                          (setq tmp4 (cdr tmp3))
                          (and (consp tmp4)
                            (progn
                              (setq i (car tmp4))
                              (setq tmp5 (cdr tmp4))
                              (and (consp tmp5)
                                (eq (cdr tmp5) nil)
                                (progn
                                  (setq tmp6 (car tmp5))
                                  (and (consp tmp6)
                                    (eq (car tmp6) 'by)
                                    (progn
                                      (setq tmp7 (cdr tmp6))
                                      (and (consp tmp7)
                                        (eq (cdr tmp7) nil)
                                        (progn
                                          (setq incr (car tmp7))
                                          t))))))))))))))))))})))
(setq result (cons (list 'step var i incr) result)))
((and (consp sex)
  (eq (car sex) 'in)
  (progn
   (setq tmp1 (cdr sex))
   (and (consp tmp1)
     (progn
      (setq var (car tmp1))
      (setq tmp2 (cdr tmp1))
      (and (consp tmp2)
        (eq (cdr tmp2) nil)
        (progn
         (setq tmp3 (car tmp2))
         (and (consp tmp3)
           (eq (car tmp3) 'by)
           (progn
            (setq tmp4 (cdr tmp3))
            (and (consp tmp4)
              (progn
               (setq tmp5 (car tmp4))
               (and (consp tmp5)
                 (eq (car tmp5) 'segment)
                 (progn
                  (setq tmp6 (cdr tmp5))
                  (and (consp tmp6)
                    (progn
                     (setq i (car tmp6))
                     (setq tmp7 (cdr tmp6))
                     (and (consp tmp7)
                       (eq (cdr tmp7) nil)
                       (progn
                        (setq j (car tmp7))
                        t)))))))))
    (progn
     (setq tmp8 (cdr tmp4))
     (and (consp tmp8)
      (eq (cdr tmp8) nil)
      (progn
       (setq incr (car tmp8))
       (progn
        (setq result (cons (list 'step var i incr j) result))))))
((and (consp sex)
  (eq (car sex) 'in)
  (progn
   (setq tmp1 (cdr sex))
   (and (consp tmp1)
     (progn
      (setq var (car tmp1))
      (setq tmp2 (cdr tmp1))
      (and (consp tmp2)
        (eq (car tmp2) 'step)
        (setq result (cons (list 'step var i incr) result))))))
(eq (cdr tmp2) nil)
(progn
  (setq tmp3 (car tmp2))
  (and (consp tmp3)
    (eq (car tmp3) 'segment)
    (progn
      (setq tmp4 (cdr tmp3))
      (and (consp tmp4)
        (progn
          (setq i (car tmp4))
          (setq tmp5 (cdr tmp4))
          (and (consp tmp5)
            (eq (cdr tmp5) nil)
            (progn
              (setq j (car tmp5))
              t)))))))))))))
(setq result (cons (list 'step var i 1 j) result)))
(t (setq result (cons sex result)))))))

---

defun Change a Collect node to an S-expression

[loopIters2Sex p311]
[pfParts p249]
[pfCollectIterators p260]
[pf2Sex1 p300]
[pfCollectBody p260]

— defun pfCollect2Sex —

(defun |pfCollect2Sex| (pf)
  (let (var cond sex tmp1 tmp2 tmp3 tmp4)
    (setq sex
      (cons 'collect
        (append (|loopIters2Sex| (|pfParts| (|pfCollectIterators| pf)))
          (list (|pf2Sex1| (|pfCollectBody| pf))))))
    (cond
      ((and (consp sex)
        (eq (car sex) 'collect)
        (progn
          (setq tmp1 (cdr sex))
          (and (consp tmp1)
            (progn
              (setq tmp2 (car tmp1))
              (and (consp tmp2)
                (eq (car tmp2) '\|\|)))))
      (else result)))
   )))
(progn
 (setq tmp3 (cdr tmp2))
 (and (consp tmp3)
  (eq (cdr tmp3) nil)
  (progn
   (setq cond (car tmp3))
   t))))

(progn
 (setq tmp4 (cdr tmp1))
 (and (consp tmp4)
  (eq (cdr tmp4) nil)
  (progn (setq var (car tmp4)) t)))))

(symbolp var))
 (list '|\| var cond))
 (t sex))))

defun Convert a Definition node to an S-expression

(defvar pf2Sex1 p300)
(defvar pf0DefinitionLhsItems p262)
(defvar pfDefinitionRhs p261)
(defvar systemError p??)
(defvar pfLambdaTran p316)
(defvar $insideApplication p??)

— defun pfDefinition2Sex —

(defun pfDefinition2Sex (pf)
 (let (body argList tmp1 rhs id idList)
   (declare (special $insideApplication))
   (if $insideApplication
    (list 'optarg
      (|pf2Sex1| (car (|pf0DefinitionLhsItems| pf)))
      (|pf2Sex1| (|pfDefinitionRhs| pf)))
    (progn
      (setq idList (mapcar #'|pf2Sex1| (|pf0DefinitionLhsItems| pf)))
      (if (not (eql (length idList) 1))
        (systemError
          "lhs of definition must be a single item in the interpreter")
      (progn
        (setq id (car idList))
        (setq rhs (|pfDefinitionRhs| pf))
        (setq tmp1 (|pfLambdaTran| rhs))
        (setq argList (car tmp1))
        (setq body (cdr tmp1))
        )
      )
    ))
  )
)
defun Convert a Lambda node to an S-expression

(defun pfLambdaTran (pf)
  (let (retType argList argTypeList)
    (cond
      ((pfLambda? pf)
       (dolist (arg (pf0LambdaArgs pf))
        (if (pfTyped? arg)
            (progn
              (setq argList
                (cons (pfCollectArgTran (pfTypedId arg)) argList))
              (if (pfNothing? (pfTypedType arg))
                  (setq argTypeList (cons nil argTypeList))
                (setq argTypeList
                  (cons (pf2Sex1 (pfTypedType arg)) argTypeList))))
            (systemError "definition args should be typed")))
      (setq argList (nreverse argList))
      (unless (pfNothing? (pfLambdaRets pf))
        (setq retType (pf2Sex1 (pfLambdaRets pf)))))
    (setq argTypeList (cons retType (nreverse argTypeList)))
    (cons argList
      (list argTypeList
        (mapcar #'(lambda (x) (declare (ignore x)) nil) argTypeList)
        (pf2Sex1 (pfLambdaBody pf))))))
(t (cons 'id (list '(nil) '(nil) (|pf2Sex1| pf))))

---

defun pfCollectArgTran

(defvar pfCollect? p260)
(defvar pf2sex1 p??)
(defvar pfParts p249)
(defvar pfCollectIterators p260)
(defvar pfCollectBody p260)

---

defun pfCollectArgTran

(defun |pfCollectArgTran| (pf)
  (let (cond tmp2 tmp1 id conds)
    (cond
      (pfCollect? pf)
        (setq conds (mapcar #'|pf2sex1| (pfParts (pfCollectIterators pf))))
        (setq id (pf2Sex1 (pfCollectBody pf)))
        (cond
          (and (consp conds) ; conds is [ "|", cond ]
            (eq (cdr conds) nil)
            (progn
              (setq tmp1 (car conds))
              (and (consp tmp1)
                (eq (car tmp1) '\|)
                (progn
                  (setq tmp2 (cdr tmp1))
                  (and (consp tmp2)
                    (eq (car tmp2) '\|)
                    (progn
                      (setq cond (car tmp2))
                      t)))))
        (list '\| id cond))
        (t (cons id conds))))
      (t (pf2Sex1 pf))))

---

defun Convert a Lambda node to an S-expression

(defvar pfLambdaTran p316)

---

defun pfLambda2Sex

(defun |pfLambda2Sex| (pf)
  (let (body argList tmp1)
    (setq tmp1 (|pfLambdaTran| pf))
    (setq argList (car tmp1))
    (setq body (cdr tmp1))
    (cons 'adef (cons argList body))))

---

**defun Convert a Rule node to an S-expression**

[pfLhsRule2Sex p318]  
[pfRuleLhsItems p286]  
[pfRhsRule2Sex p319]  
[pfRuleRhs p286]  
[ruleLhsTran p322]  
[rulePredicateTran p319]  
[$multiVarPredicateList p??]  
[$predicateList p??]  
[$quotedOpList p??]

---

| defun pfRule2Sex |

(defun |pfRule2Sex| (pf)
  (let (|$multiVarPredicateList| |$predicateList| |$quotedOpList| rhs lhs)
    (declare (special |$multiVarPredicateList| |$predicateList| |$quotedOpList|))
    (setq |$quotedOpList| nil)
    (setq |$predicateList| nil)
    (setq |$multiVarPredicateList| nil)
    (setq lhs (|pfLhsRule2Sex| (|pfRuleLhsItems| pf)))
    (setq rhs (|pfRhsRule2Sex| (|pfRuleRhs| pf)))
    (setq lhs (|ruleLhsTran| lhs))
    (|rulePredicateTran|)
    (if |$quotedOpList|
      (list '|rule| lhs rhs (cons '|construct| |$quotedOpList|))
      (list '|rule| lhs rhs))))

---

**defun Convert the Lhs of a Rule to an S-expression**

[pf2Sex1 p300]  
[$insideRule p??]

---

| defun pfLhsRule2Sex |

(defun |pfLhsRule2Sex| (lhs)
(let ([|$insideRule|])
 (declare (special|$insideRule|))
 (setq|$insideRule| '|left|)
 ([pf2Sex1| lhs])))

---

defun Convert the Rhs of a Rule to an S-expression

[pf2Sex1 p300]

$insideRule p??]
———
defun pfRhsRule2Sex ——

(defun |pfRhsRule2Sex| (rhs)
(let ([|$insideRule|])
 (declare (special|$insideRule|))
 (setq|$insideRule| '|right|)
 ([pf2Sex1| rhs]))

———

defun Convert a Rule predicate to an S-expression

;rulePredicateTran rule ==
 ; null $multiVarPredicateList => rule
 ; varList := patternVarsOf [rhs for [.,.,:rhs] in $multiVarPredicateList]
 ; predBody :=
 ; CDR $multiVarPredicateList =>
 ; ['AND, [:pvarPredTran(rhs, varList) for [.,.,:rhs] in
 ; $multiVarPredicateList]]
 ; [ [.,...,rhs],.. ] := $multiVarPredicateList
 ; pvarPredTran(rhs, varList)
 ; ['suchThat, rule,
 ; ['construct, : ["QUOTE", var] for var in varList]],
 ; ['ADEF, '(predicateVariable),
 ; '((Boolean) (List (Expression (Integer))))), '(() ()),
 ; predBody]]

[patternVarsOf p321]
[pvarPredTran p322]
[$multiVarPredicateList p??]
———
defun rulePredicateTran ——
(defun rulePredicateTran (rule)
  (let (predBody varList rhs tmp1 result)
    (declare (special $multiVarPredicateList))
    (if (null $multiVarPredicateList)
        rule
        (progn
          (setq varList
            (patternVarsOf
              ((lambda (t1 t2 t3)
                (loop
                  (cond
                    ((or (atom t2)
                      (progn
                        (setq t3 (car t2))
                        nil))
                      (return (nreverse t1))))
                    t
                    (and (consp t3)
                      (progn
                        (setq tmp1 (cdr t3))
                        (and (consp tmp1)
                          (progn
                            (setq rhs (cdr tmp1))
                            t))))
                      (setq t1 (cons rhs t1))))))
          (setq t2 (cdr t2)))
          nil $multiVarPredicateList
          nil)))
    (setq predBody
      (cond
        ((cdr $multiVarPredicateList)
          (cons 'and
            ((lambda (t4 t5 t6)
              (loop
                (cond
                  ((or (atom t5)
                    (progn
                      (setq t6 (car t5))
                      nil))
                    (return (nreverse t4))))
                  t
                  (and (consp t6)
                    (progn
                      (setq tmp1 (cdr t6))
                      (and (consp tmp1)
                        (progn
                          (setq rhs (cdr tmp1))
                          t))))
                    (setq t4
                      (append (reverse (lpvarPredTran rhs varList))
                      t4))))))))
      (setq $multiVarPredicateList nil)
      result)))
(setq t5 (cdr t5)))
(nil |$multiVarPredicateList| nil)))
(t
(progn
(setq rhs (cddar |$multiVarPredicateList|))
(pvarPredTran| rhs varList)))))))
(dolist (var varList) (push (list 'quote var) result))
(list '|suchThat| rule
(cons '|construct| (nreverse result))
(list 'adef '|(predicateVariable|)
'(|Boolean|)
'(|List| (|Expression| (|Integer|)))
'(nil nil) predBody))))

defun patternVarsOf

(patternVarsOf1 p321)

— defun patternVarsOf —

(defun |patternVarsOf1| (expr)
 (|patternVarsOf1| expr nil))

— defun patternVarsOf1 —

(defun |patternVarsOf1| (expr varList)
 (let (arg1 op)
 (cond
 ((null expr) varList)
 ((atom expr)
  (cond
   ((null (symbolp expr)) varList)
   ((member expr varList) varList)
   (t (cons expr varList))))
  ((and (consp expr)
       (progn
        (setq op (car expr))
        (setq t5 (cdr t5)))))
 (setq t5 (cdr t5)))
(nil |$multiVarPredicateList| nil)))
(t
(progn
(setq rhs (cddar |$multiVarPredicateList|))
(pvarPredTran| rhs varList)))))))
(dolist (var varList) (push (list 'quote var) result))
(list '|suchThat| rule
(cons '|construct| (nreverse result))
(list 'adef '|(predicateVariable|)
'(|Boolean|)
'(|List| (|Expression| (|Integer|)))
'(nil nil) predBody))))

defun patternVarsOf

(patternVarsOf1 p321)

— defun patternVarsOf —

(defun |patternVarsOf1| (expr)
 (|patternVarsOf1| expr nil))

— defun patternVarsOf1 —

(defun |patternVarsOf1| (expr varList)
 (let (arg1 op)
 (cond
 ((null expr) varList)
 ((atom expr)
  (cond
   ((null (symbolp expr)) varList)
   ((member expr varList) varList)
   (t (cons expr varList))))
  ((and (consp expr)
       (progn
        (setq op (car expr))
        (setq t5 (cdr t5))))
(setq argl (cdr expr))
(progn
  (dolist (arg argl)
    (setq varList (|patternVarsOf| arg varList)))
  varList))
(t varList)))

defun pvarPredTran

— defun pvarPredTran —

(defun |pvarPredTran| (rhs varList)
  (let ((i 0))
    (dolist (var varList rhs)
      (setq rhs (nsubst (list '|elt| '|predicateVariable| (incf i)) var rhs)))
  )(t varList))

defun Convert the Lhs of a Rule node to an S-expression

[|patternVarsOf| p321]
[|nsubst| p?]
[$predicateList| p??]
[$multiVarPredicateList| p??]

— defun ruleLhsTran —

(defun |ruleLhsTran| (ruleLhs)
  (let (predicate var vars predRhs predLhs name)
    (declare (special |$predicateList| |$multiVarPredicateList|))
    (dolist (pred |$predicateList|)
      (setq name (car pred))
      (setq predLhs (cadr pred))
      (setq predRhs (cddr pred))
      (setq vars (|patternVarsOf| predRhs))
      (cond
        ((cdr vars)
          (setq ruleLhs (nsubst predLhs name ruleLhs)))
        (t (setq |$multiVarPredicateList| (cons pred |$multiVarPredicateList|))
          (setq var (cadr predLhs))))
  ))
(setq predicate
  (list '|suchThat| predLhs (list 'adef (list var)
    '(((|Boolean|) (|Expression| (|Integer|))) 'nil nil) predRhs)))
(setq ruleLhs (nsubst predicate name ruleLhs)))

------

defvar $dotdot

| initvars |
(defvar $dotdot '|..|)

------

defun Translate ops into internal symbols

[$dotdot p323]

| defun opTran 0 |
(defun |opTran| (op)
  (declare (special |$dotdot|))
  (cond
    ((equal op |$dotdot|) 'segment)
    ((eq op '[]) '|construct|)
    ((eq op '{}) '|braceFromCurly|)
    ((eq op 'is) '|is|)
    (t op)))

------
Chapter 9

Keyed Message Handling

Throughout the interpreter there are messages printed using a symbol for a database lookup. This was done to enable translation of these messages languages other than English.

Axiom messages are read from a flat file database and returned as one long string. They are preceded in the database by a key and this is how they are referenced from code. For example, one key is S2IL0001 which means:

| S2   | Scratchpad II designation |
| I    | from the interpreter      |
| L    | originally from LISPLIB BOOT |
| 0001 | a sequence number         |

Each message may contain formatting codes and parameter codes. The formatting codes are:

- `%b` turn on bright printing
- `%ceoff` turn off centering
- `%ceon` turn on centering
- `%d` turn off bright printing
- `%f` user defined printing
- `%i` start indentation of 3 more spaces
- `%l` start a new line
- `%m` math-print an expression
- `%rjoff` turn off right justification (actually ragged left)
- `%rjon` turn on right justification (actually ragged left)
- `%s` pretty-print as an S-expression
- `%u` unindent 3 spaces
- `%x#` insert # spaces

The parameter codes look like `%1, `%2b, `%3p, `%4m, `%5bp, `%6s where the digit is the parameter number and the letters following indicate additional formatting. You can indicate as many additional formatting qualifiers as you like, to the degree they make sense.
• The “p” code means to call prefix2String on the parameter, a standard way of printing abbreviated types.

• The “P” operator maps prefix2String over its arguments.

• The “o” operation formats the argument as an operation name.

• The “b” means to print that parameter in a bold (bright) font.

• The “c” means to center that parameter on a new line.

• The “r” means to right justify (ragged left) the argument.

• The “f” means that the parameter is a list \([\text{fn}, :\text{args}]\) and that “\text{fn}” is to be called on “\text{args}” to get the text.

Look in the file with the name defined in $\text{defaultMsgDatabaseName}$ above for examples.

defvar $\text{cacheMessages}$

This is used for debugging

— initvars —

(defvar $\text{cacheMessages}$ t)

—

defvar $\text{msgAlist}$

— initvars —

(defvar $\text{msgAlist}$ nil)

—

defvar $\text{testingErrorPrefix}$

— initvars —

(defvar $\text{testingErrorPrefix}$ "Daly Bug")

—
defvar $texFormatting

  — initvars —

  (defvar |$texFormatting| nil)

——

defvar $*msghash*

  — initvars —

  (defvar *msghash* nil "hash table keyed by msg number")

——

defvar $msgdbPrims

  — initvars —

  (defvar |$msgdbPrims|
     '(|%b| |%d| |%i| |%u| |%n| |%x| |%ce| |%rj| "%U" "%b" "%d"
       "%l" "%i" "%a" "%U" "%n" "%x" "%ce" "%rj"))

——

defvar $msgdbPunct

  — initvars —

  (defvar |$msgdbPunct|
     '(|.| |,| ! |:| |;| ? |] |)) "." "," "!" ";" ",;" "?" "]" ")")

——
defvar $msgdbNoBlanksBeforeGroup

    — initvars —

    (defvar |$msgdbNoBlanksBeforeGroup|
     '(" " | | "%" % ,@|$msgdbPrims| ,@|$msgdbPunct|))

defvar $msgdbNoBlanksAfterGroup

    — initvars —

    (defvar |$msgdbNoBlanksAfterGroup|
     '(" " | | "%" % ,@|$msgdbPrims| [ |(| "([" ")])

defun Fetch a message from the message database

If the *msghash* hash table is empty we call cacheKeyedMsg to fill the table, otherwise we do a key lookup in the hash table. [object2Identifier p328]

(cacheKeyedMsg p328)
[$defaultMsgDatabaseName p6]
[*msghash* p327]

    — defun fetchKeyedMsg —

    (defun |fetchKeyedMsg| (key ignore)
        (declare (ignore ignore) (special *msghash* |$defaultMsgDatabaseName|))
        (setq key (|object2Identifier| key))
        (unless *msghash*
            (setq *msghash* (make-hash-table))
            (cacheKeyedMsg |$defaultMsgDatabaseName|))
        (gethash key *msghash*))

defun Cache messages read from message database

[done p328]  
[done p328]
(defun cacheKeyedMsg (file)
  (let ((line "") (msg ") key)
          (declare (special *msghash*))
    (with-open-file (in file)
      (catch 'done
        (loop
          (setq line (read-line in nil nil))
          (cond
            (null line)
            (when key (setf (gethash key *msghash*) msg))
            (throw 'done nil))
            ((= (length line) 0))
            ((char= (schar line 0) #\S)
              (when key (setf (gethash key *msghash*) msg))
              (setq key (intern line "BOOT"))
              (setq msg ""))
            ('else
              (setq msg (concatenate 'string msg line)))))

    (setq line (read-line in nil nil))
    (cond
      (null line)
      (when key (setf (gethash key *msghash*) msg))
      (throw 'done nil))
    ((= (length line) 0))
    ((char= (schar line 0) #\S)
      (when key (setf (gethash key *msghash*) msg))
      (setq key (intern line "BOOT"))
      (setq msg ""))
    ('else
      (setq msg (concatenate 'string msg line)))))))

(defun getKeyedMsg
  (defun getKeyedMsg (key) (fetchKeyedMsg key nil))

(defun sayKeyedMsg
  (defun sayKeyedMsg (key args)
    (let (($texFormatting))
      (declare (special $texFormatting)))
    (let ((key args)
          (let ($texFormatting))
            (declare (special $texFormatting))))
defun Handle msg formatting and print to file

[segmentKeyedMsg p330]
[getKeyedMsg p329]
[substituteSegmentedMsg p774]
[flowSegmentedMsg p774]
[ sayMSG2File p331]
[ sayMSG p331]
[ printMsgsToFile p740]
[ linelength p774]
[ margin p774]
[ displayMsgNumber p746]

—— defun sayKeyedMsgLocal ——

(defun |sayKeyedMsgLocal| (key args)
  (let (msg msgp)
    (declare (special |printMsgsToFile| $linelength $margin |displayMsgNumber|))
    (setq msg (|segmentKeyedMsg| (|getKeyedMsg| key))
    (setq msg (|substituteSegmentedMsg| msg args))
    (when |displayMsgNumber| (setq msg '("%b", key |:| "%d", . ,msg))
    (setq msgp (|flowSegmentedMsg| msg $linelength $margin))
    (when |printMsgsToFile| (|sayMSG2File| msgp))
    (|sayMSG| msgp)))

——

defun Break a message into words

|string2Words p774|

—— defun segmentKeyedMsg ——

(defun |segmentKeyedMsg| (msg) (|string2Words| msg))
defun Write a msg into spadmsg.listing file

(makePathname p1042)
(makeiostream p982)
sayBrightly1 p1049
(shut p982)

— defun sayMSG2File —

(defun sayMSG2File (msg)
  (let (file str)
    (setq file (makePathname `spadmsg `listing `a))
    (setq str (defiostream `((mode . output) (file . ,file)) 255 0))
    (sayBrightly1 msg str)
    (shut str)))

— defun sayMSG —

(defun sayMSG (x)
  (declare (special $algebraOutputStream))
  (when x (sayBrightly1 x $algebraOutputStream)))
Chapter 10

Stream Utilities

The input stream is parsed into a large s-expression by repeated calls to Delay. Delay takes a function f and an argument x and returns a list consisting of ("nonnullstream" f x). Eventually multiple calls are made and a large list structure is created that consists of ("nonnullstream" f x ("nonnullstream" f1 x1 ("nonnullstream" f2 x2... This delay structure is given to StreamNull which walks along the list looking at the head. If the head is “nonnullstream” then the function is applied to the argument.

So, in effect, the input is “zipped up” into a Delay data structure which is then evaluated by calling StreamNull. This ”zipped stream” parser was a research project at IBM and Axiom was the testbed (which explains the strange parsing technique).

defun npNull
[StreamNull p333]

— defun npNull —

(defun |npNull| (x) (|StreamNull| x))

defun StreamNull
[eqcar p??]

— defun StreamNull 0 —

(defun |StreamNull| (x)
(let (st)
  (cond
    ((or (null x) (eqcar x 'nullstream)) t)
    (t
      (lambda nil
        (loop
          (cond
            ((not (eqcar x 'nonnullstream)) (return nil))
            (t
              (setq st (apply (cadr x) (cddr x)))
              (rplaca x (car st))
              (rplacd x (cdr st))))))))
  (eqcar x 'nullstream))))
Chapter 11

Code Piles

The `insertpiles` function converts a line-list to a line-forest where a line is a token-dequeue and has a column which is an integer. An A-forest is an A-tree-list An A-tree has a root which is an A, and subtrees which is an A-forest.

A forest with more than one tree corresponds to a Scratchpad pile structure (t1;t2;t3;...;tn), and a tree corresponds to a pile item. The (; and ) tokens are inserted into a >1-forest, otherwise the root of the first tree is concatenated with its forest. column t is the number of spaces before the first non-space in line t

defun insertpiles

---

(defun insertpiles (s)
  (let (stream a t1 h1 t2 h tmp1)
    (cond
      ((npNull s) (list nil 0 nil s))
      (t
        (setq tmp1 (list (car s) (cdr s)))
        (setq h (car tmp1))
        (setq t2 (cadr tmp1))
        (cond
          ((pilePlusComment h)
            (setq tmp1 (pilePlusComments s))
            (setq h1 (car tmp1))
            (setq tmp1 (list h1 h t1 h1 t2 h2 tmp1))))))

---
(setq t1 (cadr tmp1))
(setq a (pileTree (- 1) t1))
(cons (list (pileCforest
              (append h1 (cons (elt a 2) nil))))
(elt a 3)))
(t
(setq stream (cadar s))
(setq a (pileTree -1 s))
(cons (list (list (elt a 2) stream)) (elt a 3))))

---

defun pilePlusComment

tokType p413
npNull p333
pilePlusComment p336
pilePlusComments p336

— defun pilePlusComment —

(defun pilePlusComment (arg)
  (eq (tokenType (caar arg)) 'comment))

---

defun pilePlusComments

— defun pilePlusComments —

(defun pilePlusComments (s)
  (let (t1 h1 t2 h tmp1)
    (cond
      (null s) (list nil s))
    (t
     (setq tmp1 (list (car s) (cdr s)))
     (setq h (car tmp1))
     (setq t2 (cadr tmp1))
     (cond
      (null (pilePlusComment h))
      (setq tmp1 (pilePlusComments t2))
      (setq h1 (car tmp1))
      (setq t1 (cadr tmp1))
      (list (cons h h1) t1))
(t
  (list nil s)))))))

---

defun pileTree

[npNull p333]
[pileColumn p337]
[pileForests p337]

— defun pileTree —

(defun |pileTree| (n s)
  (let ((hh t1 h tmp1)
      (cond
        ((|npNull| s) (list nil n nil s))
        (t
          (setq tmp1 (list (car s) (cdr s)))
          (setq h (car tmp1))
          (setq t1 (cadr tmp1))
          (setq hh (|pileColumn| (car h)))
          (cond
            ((< n hh) (|pileForests| (car h) hh t1))
            (t (list nil n nil s))))))))

---

defun pileColumn

[tokPosn p413]

— defun pileColumn —

(defun |pileColumn| (arg)
  (cdr (|tokPosn| (caar arg))))

---

defun pileForests

[pileForest p338]
[npNull p333]
defun pileForests (h n s)
(let (t1 h1 tmp1)
  (setq tmp1 (pileForest n s))
  (setq h1 (car tmp1))
  (setq t1 (cadr tmp1))
  (cond
    ((npNull h1) (list t n h s))
    (t (pileForests (pileCtree h h1) n t1))))

---

defun pileForest

(let (t1 h1 t2 h hh b tmp)
  (setq b (car tmp))
  (setq hh (cadr tmp))
  (setq h (caddr tmp))
  (setq t2 (cadddr tmp))
  (cond
    (b
      (setq tmp (pileForest1 hh t2))
      (setq h1 (car tmp))
      (setq t1 (cadr tmp))
      (list (cons h h1) t1))
    (t
      (list nil s)))))

---

defun pileForest1

(let (t1 h1 t2 h hh b tmp)
  (setq b (car tmp))
  (setq hh (cadr tmp))
  (setq h (caddr tmp))
  (setq t2 (cadddr tmp))
  (cond
    (b
      (setq tmp (pileForest1 hh t2))
      (setq h1 (car tmp))
      (setq t1 (cadr tmp))
      (list (cons h h1) t1))
    (t
      (list nil s)))))
--- defun pileForest1 ---

(defun pileForest1 (n s)
  (let (t1 h1 t2 h n1 b tmp)
    (setq tmp (eqpileTree n s))
    (setq b (car tmp))
    (setq n1 (cadr tmp))
    (setq h (caddr tmp))
    (setq t2 (cadddr tmp))
    (cond
      (b
        (setq tmp (pileForest1 n t2))
        (setq h1 (car tmp))
        (setq t1 (cadr tmp))
        (list (cons h h1) t1))
      (t (list nil s))))

---

defun eqpileTree

[npNull p333]
[pileColumn p337]
[pileForests p337]

--- defun eqpileTree ---

(defun eqpileTree (n s)
  (let (hh t1 h tmp)
    (cond
      ((npNull s) (list nil n nil s))
      (t
        (setq tmp (list (car s) (cdr s)))
        (setq h (car tmp))
        (setq t1 (cadr tmp))
        (setq hh (pileColumn (car h)))
        (cond
          ((equal hh n) (pileForests (car h) hh t1))
          (t (list nil n nil s)))))))

---
defun pileCtree

[dqAppend p344]
[pileCforest p340]

(defun |pileCtree| (x y)
  (dqAppend| x (pileCforest| y)))

---

defun pileCforest

Only enpiles forests with >= 2 trees [tokPart p413]
[enPile p340]
[separatePiles p341]

(defun |pileCforest| (x)
  (let (f)
    (cond
      ((null x) nil)
      ((null (cdr x)) (setq f (car x))
        (cond
          ((eq (tokPart (caar f)) 'if) (enPile f))
          (t f)))
      (t (enPile (separatePiles| x))))))

---

defun enPile

[dqConcat p343]
[dqUnit p343]
tokConstruct p411]
[firstTokPosn p341]
[lastTokPosn p341]

(defun |enPile| (x)
  (dqConcat| x)
(list
  (|dqUnit| (|tokConstruct| 'key| 'settab (|firstTokPosn| x)))
  x
  (|dqUnit| (|tokConstruct| 'key| 'backtab (|lastTokPosn| x))))

---

defun firstTokPosn

tokPosn p413

— defun firstTokPosn —

(defun |firstTokPosn| (arg) (|tokPosn| (caar arg)))

---

defun lastTokPosn

tokPosn p413

— defun lastTokPosn —

(defun |lastTokPosn| (arg) (|tokPosn| (cadr arg)))

---

defun separatePiles

dqUnit p343
tokConstruct p411
tokPosn p341
dqConcat p343
[separatePiles p341]

— defun separatePiles —

(defun |separatePiles| (x)
  (let (semicolon a)
    (cond
      ((null x) nil)
      ((null (cdr x)) (car x))
      (else (append (|separatePiles| (|car| x)) (|lastTokPosn| (|cdr| x))))))
(t
  (setq a (car x))
  (setq semicolon
    (dqUnit ((tokConstruct 'key 'backset (lastTokPosn a))))
    (dqConcat (list a semicolon (separatePiles (cdr x))))))}
Chapter 12

Deque Functions

The dqUnit makes a unit dq i.e. a dq with one item, from the item

```lisp
(defun dqUnit
  (defun dqUnit 0
    (let (a)
      (setq a (list s))
      (cons a a)))
```

The dqConcat function concatenates a list of dq's, destroying all but the last [dqAppend p344]
[dqConcat p343]

```lisp
(defun dqConcat
  (defun dqConcat (ld)
    (cond
      ((null ld) nil)
      ((null (cdr ld)) (car ld))
      (t (|dqAppend| (car ld) (|dqConcat| (cdr ld))))))
```

343
defun dqAppend

The dqAppend function appends 2 dq’s, destroying the first

(defun dqAppend (x y)
  (cond
   ((null x) y)
   ((null y) x)
   (t
    (rplacd (cdr x) (car y))
    (rplacd x (cdr y)) x)))

defun dqToList

(defun dqToList (s)
  (when s (car s)))
Chapter 13

Message Handling

13.1 The Line Object

defun Line object creation

This is called in only one place, the incLine1 function.

— defun lnCreate 0 —

(defun lnCreate (extraBlanks string globalNum &rest optFileStuff)
  (let ((localNum (first optFileStuff))
        (filename (second optFileStuff)))
    (unless localNum (setq localNum 0))
    (list extraBlanks string globalNum localNum filename)))

—

defun Line element 0; Extra blanks

— defun lnExtraBlanks 0 —

(defun lnExtraBlanks (lineObject) (elt lineObject 0))

—

defun Line element 1; String

— defun lnString 0 —
(defun lnString (lineObject) (elt lineObject 1))

---

defun Line element 2; Gblal number

-- defun lnGlobalNum 0 --

(defun lnGlobalNum (lineObject) (elt lineObject 2))

---

defun Line element 2; Set Gblal number

-- defun lnSetGlobalNum 0 --

(defun lnSetGlobalNum (lineObject num)
  (setf (elt lineObject 2) num))

---

defun Line element 3; Local number

-- defun lnLocalNum 0 --

(defun lnLocalNum (lineObject) (elt lineObject 3))

---

defun Line element 4; Place of origin

-- defun lnPlaceOfOrigin 0 --

(defun lnPlaceOfOrigin (lineObject) (elt lineObject 4))

---
defun Line element 4: Is it a filename?

(lnFileName? p347)

— defun lnImmediate? 0 —

(defun |lnImmediate?| (lineObject) (null (|lnFileName?| lineObject)))

———

defun Line element 4: Is it a filename?

— defun lnFileName? 0 —

(defun |lnFileName?| (lineObject)
  (let (filename)
    (when (consp (setq filename (elt lineObject 4))) filename)))

———

defun Line element 4; Get filename

[lnFileName? p347]
[ncBug p368]

— defun lnFileName —

(defun |lnFileName| (lineObject)
  (let (fN)
    (if (setq fN (|lnFileName?| lineObject))
      fN
      (|ncBug| "there is no file name in %1" (list lineObject)))))

———

13.2 Messages

defun msgCreate

msgObject  tag -- category of msg
  -- attributes as a-list
'imPr => don’t save for list processing
toWhere, screen or file
'norep => only display once in list
pos -- position with possible FROM/TO tag
key -- key for message database
argL -- arguments to be placed in the msg test
prefix -- things like "Error: 

text -- the actual text

[setMsgForcedAttrList p364]
[putDatabaseStuff p365]
[initImPr p367]
[initToWhere p368]

--- defun msgCreate ---
(defun msgCreate | (tag posWTag key argL optPre &rest optAttr)
(let (msg)
  (when (consp key) (setq tag '|old|))
  (setq msg (list tag posWTag key argL optPre nil))
  (when (car optAttr) (|setMsgForcedAttrList| msg (car optAttr)))
  (|putDatabaseStuff| msg)
  (|initImPr| msg)
  (|initToWhere| msg)
  msg))

---
defun getMsgPosTagOb

--- defun getMsgPosTagOb 0 ---
(defun getMsgPosTagOb | (msg) (elt msg 1))

---
defun getMsgKey

--- defun getMsgKey 0 ---
(defun getMsgKey | (msg) (elt msg 2))

---
13.2. Messages

(defun getMsgArgL
  (msg) (elt msg 3))

(defun getMsgPrefix
  (msg) (elt msg 4))

(defun setMsgPrefix
  (msg val) (setf (elt msg 4) val))

(defun getMsgText
  (msg) (elt msg 5))

(defun setMsgText
  (msg val) (setf (elt msg 5) val))
CHAPTER 13. MESSAGE HANDLING

(defun |setMsgText| (msg val)
  (setf (elt msg 5) val))

---

defun getMsgPrefix?

— defun getMsgPrefix? 0 —

(defun |getMsgPrefix| (msg)
  (let ((pre (|getMsgPrefix| msg)))
    (unless (eq pre '|noPre|) pre)))

---

defun getMsgTag

The valid message tags are: line, old, error, warn, bug, unimple, remark, stat, say, debug
[ncTag p415]

— defun getMsgTag 0 —

(defun |getMsgTag| (msg) (|ncTag| msg))

---

defun getMsgTag?

[IFCAR p??]
[getMsgTag p350]

— defun getMsgTag? 0 —

(defun |getMsgTag| (msg) (|ncTag| msg))

---
defun line?

(getMsgTag p350)

— defun line? —

(defun line? (msg) (eq (getMsgTag msg) 'line))

——

defun leader?

(getMsgTag p350)

— defun leader? —

(defun leader? (msg) (eq (getMsgTag msg) 'leader))

——

defun toScreen?

(getMsgToWhere p363)

— defun toScreen? —

(defun toScreen? (msg) (not (eq (getMsgToWhere msg) 'fileOnly)))

——

defun ncSoftError

Messages for the USERS of the compiler. The program being compiled has a minor error.
Give a message and continue processing. [desiredMsg p352]
[processKeyedError p353]
[msgCreate p347]
[$newcompErrorCount p26]

— defun ncSoftError —

(defun ncSoftError (pos erMsgKey erArgL &rest optAttr)
  (declare (special $newcompErrorCount))
)
defun ncHardError

The program being compiled is seriously incorrect. Give message and throw to a recovery point. [desiredMsg p352]
[processKeyedError p353]
[msgCreate p347]
[ncError p67]
[$newcompErrorCount p26]

(defun ncHardError (pos erMsgKey erArgL &rest optAttr)
  (let (erMsg)
    (declare (special |$newcompErrorCount|))
    (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
    (if (|desiredMsg| erMsgKey)
      (setq erMsg
        (|processKeyedError|
          (|msgCreate| '|error| pos erMsgKey erArgL
            "Error" optAttr))))
      (|ncError|)))

defun desiredMsg

— defun desiredMsg 0 —

(defun desiredMsg (erMsgKey &rest optCatFlag)
  (declare (ignore erMsgKey))
  (cond
    ((null (null optCatFlag)) (car optCatFlag))
    (t t)))
defun processKeyedError

(getMsgTag? p350)
(getMsgKey p348)
(getMsgPrefix? p350)
sayBrightly p??
(CallName p??)
[msgImPr? p358]
[msgOutputter p353]
[$ncMsgList p25]

<table>
<thead>
<tr>
<th>defun processKeyedError</th>
</tr>
</thead>
</table>
| (defun |processKeyedError| (msg)
| (prog (pre erMsg)
| (declare (special |$ncMsgList|))
| (cond
| ((eq (|getMsgTag?| msg) '|old|)
| (setq erMsg (|getMsgKey| msg)))
| (cond
| ((setq pre (|getMsgPrefix?| msg))
| (setq erMsg (cons '|%b| (cons pre (cons '|%d| erMsg)))))))
| (|sayBrightly| (cons "old msg from " (cons (|CallName| 4) erMsg)))
| ((|msgImPr?| msg) (|msgOutputter| msg))
| (t (setq |$ncMsgList| (cons msg |$ncMsgList|))))

— defun processKeyedError —

defun msgOutputter

(getStFromMsg p354)
[leader? p351]
[line? p351]
toScreen? p351
[flowSegmentedMsg p??]
sayBrightly p??]
toFile? p363
[alreadyOpened? p363]
[$linelength p774]

<table>
<thead>
<tr>
<th>— defun msgOutputter —</th>
</tr>
</thead>
</table>
| (defun |msgOutputter| (msg)
| (let (alreadyOpened shouldFlow st)
| (declare (special |$linelength|))
| (setq st (|getStFromMsg| msg))

— defun msgOutputter —
(setq shouldFlow (null (or (|leader?| msg) (|line?| msg))))
(when (|toScreen?| msg)
  (when shouldFlow (setq st (|flowSegmentedMsg| st $linelength 0))
    (|sayBrightly| st))
  (when (|toFile?| msg)
    (when shouldFlow (setq st (|flowSegmentedMsg| st (- $linelength 6) 0)))
    (setq alreadyOpened (|alreadyOpened?| msg))))

defun listOutputter

[msgOutputter p353]

  — defun listOutputter —

(defun |listOutputter| (outputList)
  (dolist (msg outputList)
    (msgOutputter msg)))

— defun listOutputter —

defun getStFromMsg

[getPreStL p355]
[getMsgPrefix? p350]
[getMsgTag p350]
[getMsgText p349]
[getPosStL p356]
[getMsgKey? p362]
[pname p1045]
[getMsgLitSym p362]
[tabbing p362]

  — defun getStFromMsg —

(defun |getStFromMsg| (msg)
  (let (st posStL preStL)
    (setq preStL (|getPreStL| (|getMsgPrefix?| msg))
      st
      (cond
       ((eq (|getMsgTag| msg) '|line|)
        (cons ""
          (cons "%x1" (append preStL (cons (|getMsgText| msg) nil)))))
      (t
        (null (|leader?| msg)
          (null (|line?| msg))))
    (when (|toScreen?| msg)
      (when shouldFlow (setq st (|flowSegmentedMsg| st $linelength 0))
        (|sayBrightly| st))
      (when (|toFile?| msg)
        (when shouldFlow (setq st (|flowSegmentedMsg| st (- $linelength 6) 0)))
        (setq alreadyOpened (|alreadyOpened?| msg))))
  (when (|toFile?| msg)
    (when shouldFlow (setq st (|flowSegmentedMsg| st (- $linelength 6) 0)))
    (setq alreadyOpened (|alreadyOpened?| msg))))

— defun listOutputter —

— defun getStFromMsg —
13.2. MESSAGES

(setq posStL (|getPosStL| msg))
(setq st
  (cons posStL
    (cons (|getMsgLitSym| msg)
      (cons ""
        (append preStL
          (cons (|tabbing| msg)
            (|getMsgText| msg))))))))

---

defvar $preLength

— initvars —

(defvar $preLength 11)

---

defun getPreStL

[size p1045]
[$preLength p355]

— defun getPreStL 0 —

(defun |getPreStL| (optPre)
  (let (spses extraPlaces)
    (declare (special $preLength))
    (cond
      ((null optPre) (list " "))
      (t
        (setq spses
          (cond
            ((< 0 (setq extraPlaces (- (size optPre) (size optPre)) 3)))
            (make-string extraPlaces))
          "")
        (list '|%b| optPre spses ":" '|%d|))))

---
defun getPosStL

(showMsgPos? p357)
[msgPos p359]
[msgImPr? p358]
[decideHowMuch p359]
[listDecideHowMuch p361]
[ppos p357]
[remLine p362]
[remFile p357]
[$lastPos p??]

— defun getPosStL —

(defun getPosStL (msg)
  (let (printedOrigin printedLineNum printedFileName fullPrintedPos howMuch
        msgPos)
    (declare (special |$lastPos|))
    (cond
      ((null (|showMsgPos?| msg)) "")
      (t
       (setq msgPos (|getMsgPos| msg))
       (setq howMuch
            (if (|msgImPr?| msg)
                (|decideHowMuch| msgPos |$lastPos|)
                (|listDecideHowMuch| msgPos |$lastPos|)))
       (setq |$lastPos| msgPos)
       (setq fullPrintedPos (|ppos| msgPos))
       (setq printedFileName
            (cons "%%2" (cons "[" (append (|remFileName| fullPrintedPos) (cons "]" nil)))))
       (setq printedLineNum
            (cons "%%2" (cons "[" (append (|remLine| fullPrintedPos) (cons "]" nil)))))
       (setq printedOrigin
            (cons "%%2" (cons "[" (append fullPrintedPos (cons "]" nil)))))
       (cond
         ((eq howMuch 'org)
          (cons "" (append printedOrigin (cons '|%l| nil))))
         ((eq howMuch 'line)
          (cons "" (append printedLineNum (cons '|%l| nil))))
         ((eq howMuch 'file)
          (cons "" (append printedFileName (cons '|%l| nil))))
         ((eq howMuch 'all)
          (cons ""
           (append printedFileName
                (cons '|%l|
                (append printedLineNum
                     (cons '|%l| nil)))))))
    (t "")))))))
13.2. MESSAGES

——

defun ppos
[ pfNoPosition? p412 ]
[ pfImmediate? p?? ]
[ pfCharPosn p235 ]
[ pfLinePosn p235 ]
[ porigin p87 ]
[ pfFileName p236 ]

— defun ppos —

(defun ppos (p)
  (let (org lpos cpos)
    (cond
      (pfNoPosition? p) (list "no position"))
      (pfImmediate? p) (list "console"))
    (t
      (setq cpos (pfCharPosn p))
      (setq lpos (pfLinePosn p))
      (setq org (porigin (pfFileName p)))
      (list org " " "line" " " lpos)))))

——

defun remFile
[ IFCDR p?? ]
[ IFCAR p?? ]

— defun remFile —

(defun remFile (positionList) (ifcdr (ifcdr positionList)))

——

defun showMsgPos?
[ msgImPr? p358 ]
[ leader? p351 ]
(defun showMsgPos? (msg)
  (declare (special $erMsgToss))
  (or $erMsgToss
      (and (null (msgImPr? msg)) (null (leader? msg))))
)

---

(defvar $imPrGuys
  (list 'imPr))

---

(defun msgImPr? (msg)
  (eq (getMsgCatAttr msg $imPrGuys) 'imPr))

---

(defun getMsgCatAttr (msg cat)
  (ifcdr (qassq cat (ncAlist msg)))
)

---
defun getMsgPos

(getMsgFTTag? p)
(getMsgPosTagOb p)

--- defun getMsgPos ---

(defun |getMsgPos| (msg)
  (if (|getMsgFTTag?| msg)
    (cadr (|getMsgPosTagOb| msg))
    (|getMsgPosTagOb| msg)))

---

defun getMsgFTTag?

(ifcar p)
(getMsgPosTagOb p)

--- defun getMsgFTTag? ---

(defun |getMsgFTTag?| (msg)
  (ifcar (member (ifcar (|getMsgPosTagOb| msg)) (list 'from 'to 'fromto)))

---

defun decideHowMuch

When printing a msg, we wish not to show pos information that was shown for a previous msg with identical pos info. org prints out the word noposition or console

(ponnopos? p)
(popoPosImmediate? p)
(poFileName p)
(polinePosn p)

--- defun decideHowMuch ---

(defun |decideHowMuch| (pos oldPos)
  (cond
   ((or (and (poNopos? pos) (poNopos? oldPos))
        (and (poPosImmediate? pos) (poPosImmediate? oldPos)))
     'none)
   ((or (poNopos? pos) (poPosImmediate? pos)) 'org)
   ((or (poNopos? oldPos) (poPosImmediate? oldPos)) 'all)
   (not (equal (poFileName oldPos) (poFileName pos))) 'all)
((not (equal (poLinePosn oldPos) (poLinePosn pos))) 'line)
(t 'none)))

defun poNopos?

— defun poNopos? 0 —

(defun |poNopos?| (posn)
  (equal posn (list '|noposition|)))

——

defun poPosImmediate?

[poNopos? p360]
[lnImmediate? p347]
[poGetLineObject p361]

— defun poPosImmediate? —

(defun |poPosImmediate?| (txp)
  (unless (|poNopos?| txp) (|lnImmediate?| (|poGetLineObject| txp))))

——

defun poFileName

[lnFileName p347]
[poGetLineObject p361]

— defun poFileName —

(defun |poFileName| (posn)
  (if posn
    ((lnFileName |poGetLineObject| posn)
     (caar posn)))

——
defun poGetLineObject

— defun poGetLineObject 0 —

(defun |poGetLineObject| (posn)
   (car posn))

defun poLinePosn

[lnLocalNum p346]
[poGetLineObject p361]

— defun poLinePosn —

(defun |poLinePosn| (posn)
   (if posn
      (|lnLocalNum| (|poGetLineObject| posn))
      (cdar posn)))

defun listDecideHowMuch

[poNopos? p360]
[poPosImmediate? p360]
[poGlobalLinePosn p70]

— defun listDecideHowMuch —

(defun |listDecideHowMuch| (pos oldPos)
   (cond
      ((or (and (|poNopos?| pos) (|poNopos?| oldPos))
           (and (|poPosImmediate?| pos) (|poPosImmediate?| oldPos)))
       'none)
      ((|poNopos?| pos) 'org)
      ((|poNopos?| oldPos) 'none)
      ((< (|poGlobalLinePosn| pos) (|poGlobalLinePosn| oldPos))
       (if (|poPosImmediate?| pos) 'org 'line)
       (t 'none)))

— —
defun remLine

— defun remLine 0 —

(defun remLine (positionList) (list (ifcar positionList)))

——

defun getMsgKey?

[identp p1046]

— defun getMsgKey? 0 —

(defun getMsgKey? (msg)
  (let ((val (getMsgKey msg))
        (when (identp val) val)))

——

defun getMsgLitSym

[getMsgKey? p362]

— defun getMsgLitSym —

(defun getMsgLitSym (msg)
  (if (getMsgKey? msg) " " "*"))

——

defun tabbing

[getMsgPrefix? p350]
[preLength p355]

— defun tabbing —

(defun tabbing (msg)
  (let (chPos)
    (declare (special preLength)))
(setq chPos 2)
(when (|getMsgPrefix?| msg) (setq chPos (- (+ chPos |$preLength|) 1)))
(cons '|%t| chPos))

---

defvar $toWhereGuys

— initvars —
(defvar |$toWhereGuys| (list '|fileOnly| '|screenOnly|))

---

defun getMsgToWhere

[getMsgCatAttr p358]

— defun getMsgToWhere —
(defun |getMsgToWhere| (msg) (|getMsgCatAttr| msg |$toWhereGuys|))

---

defun toFile?

[getMsgToWhere p363]
[$fn p??]

— defun toFile? —
(defun |toFile?| (msg)
  (declare (special |$fn|))
  (and (consp |$fn|) (not (eq (|getMsgToWhere| msg '|screenOnly|)))))

---

defun alreadyOpened?

[msgImPr? p358]

— defun alreadyOpened? —
(defun alreadyOpened? (msg) (null (msgImPr? msg)))

---

(defun setMsgForcedAttrList
  (setMsgForcedAttr p364)
  (whichCat p365)

  — defun setMsgForcedAttrList —

  (defun setMsgForcedAttrList (msg attrlist)
    (dolist (attr attrlist)
      (setMsgForcedAttr msg (whichCat attr) attr)))

---

(defun setMsgForcedAttr
  (setMsgCatlessAttr p365)
  (ncPutQ p416)

  — defun setMsgForcedAttr —

  (defun setMsgForcedAttr (msg cat attr)
    (if (eq cat 'catless)
        (setMsgCatlessAttr msg attr)
      (ncPutQ msg cat attr)))

---

(defun $attrCats
  (initvars)

  — initvars —

  (defvar $attrCats (list '$imPrGuys '$toWhereGuys '$repGuys))

---
13.2. MESSAGES

defun whichCat
[ListMember? p??]
[ListMember? p??]

— defun whichCat —

(defun whichCat (attr)
(let ((found 'catless) done)
(declare (special $attrCats))
(loop for cat in $attrCats do
  (when (ListMember? attr (eval cat))
    (setq found cat)
    (setq done t))
  until done)
found))

defun setMsgCatlessAttr

TPDHERE: Changed from —catless— to ’—catless— [ncPutQ p416]

[ncPutQ p416]
[qassq p416]
[ncAlist p415]

— defun setMsgCatlessAttr —

(defun setMsgCatlessAttr (msg attr)
  (ncPutQ msg 'catless)
  (cons attr (ifcdr (qassq 'catless (ncAlist msg))))))

defun putDatabaseStuff

TPDHERE: The variable al is undefined [getMsgInfoFromKey p366]
[setMsgUnforcedAttrList p366]
[setMsgText p349]

— defun putDatabaseStuff —

(defun putDatabaseStuff (msg)
  (let (attributes text tmp)
(setq tmp (|getMsgInfoFromKey| msg))
(setq text (car tmp))
(setq attributes (cadr tmp))
(when attributes (|setMsgUnforcedAttrList| msg al))
(|setMsgText| msg text)))

---

defun getMsgInfoFromKey

[getMsgKey? p362]
[getErFromDbL p.7]
[getMsgKey p348]
[segmentKeyedMsg p330]
[removeAttributes p??]
[substituteSegmentedMsg p??]
[getMsgArgL p349]
[$msgDatabaseName p7]

— defun getMsgInfoFromKey —

(defun |getMsgInfoFromKey| (msg)
  (let ((|$msgDatabaseName| attributes tmp msgText msgKey))
    (declare (special |$msgDatabaseName|))
    (setq |$msgDatabaseName| nil)
    (setq msgText
      (cond
        (msgKey (|fetchKeyedMsg| msgKey nil))
        (t (|getMsgKey| msg)))
    (setq msgText (|segmentKeyedMsg| msgText))
    (setq tmp (|removeAttributes| msgText))
    (setq msgText (car tmp))
    (setq attributes (cadr tmp))
    (setq msgText (|substituteSegmentedMsg| msgText (|getMsgArgL| msg)))
    (list msgText attributes)))

---

defun setMsgUnforcedAttrList

[setMsgUnforcedAttr p367]
[whichCat p365]

— defun setMsgUnforcedAttrList —
\begin{Verbatim}
(defun \texttt{setMsgUnforcedAttrList} (msg attrlist)
  (dolist (attr attrlist)
    (\texttt{setMsgUnforcedAttr} msg \texttt{(whichCat attr)} attr)))
\end{Verbatim}

\textbf{defun setMsgUnforcedAttr}

\begin{Verbatim}
(setMsgCatlessAttr p365)
  \texttt{qassq p??}
  \texttt{ncAlist p415}
  \texttt{ncPutQ p416}
\end{Verbatim}

\begin{Verbatim}
\begin{verbatim}
---
(defun \texttt{setMsgUnforcedAttr} (msg cat attr)
  (cond
    ((eq cat \texttt{\textquotesingle \textsc{catless\textquotesingle}})
     (\texttt{setMsgCatlessAttr} msg attr))
    ((null (qassq cat (\texttt{ncAlist} msg)))
     (\texttt{ncPutQ} msg cat attr))))
---
\end{verbatim}
\end{Verbatim}

\textbf{defvar \$imPrTagGuys}

\begin{Verbatim}
\begin{verbatim}
---
(defvar \texttt{\$imPrTagGuys} (list \texttt{\textquotesingle \textsc{unimple\textquotesingle}}
\texttt{\textquotesingle \textsc{bug\textquotesingle}}
\texttt{\textquotesingle \textsc{debug\textquotesingle}}
\texttt{\textquotesingle \textsc{say\textquotesingle}}
\texttt{\textquotesingle \textsc{warn\textquotesingle}}))
---
\end{verbatim}
\end{Verbatim}

\textbf{defun initImPr}

\begin{Verbatim}
\begin{verbatim}
---
(defun \texttt{initImPr} (msg)
  (declare (special \texttt{\$imPrTagGuys} \texttt{\$erMsgToss})))
---
\end{verbatim}
\end{Verbatim}
(when (or $erMsgToss (member (getMsgTag msg) $imPrTagGuys))
  (setMsgUnforcedAttr msg '$imPrGuys 'imPr))))

defun initToWhere

[getMsgCatAttr p358]
[setMsgUnforcedAttr p367]

— defun initToWhere —

(defun initToWhere (msg)
  (if (member 'trace (getMsgCatAttr msg 'catless))
      (setMsgUnforcedAttr msg '$toWhereGuys 'screenOnly)))

defun ncBug

Bug in the compiler: something which shouldn’t have happened did. [processKeyedError p353]
[msgCreate p347]
[enable-backtrace p??]
[ncAbort p??]
[$nupos p26]
[$newcompErrorCount p26]

— defun ncBug —

(defun ncBug (erMsgKey erArgL &rest optAttr)
  (let (erMsg)
    (declare (special $nupos $newcompErrorCount))
    (setq $newcompErrorCount (+ $newcompErrorCount 1))
    (setq erMsg
      (processKeyedError
        (msgCreate 'bug $nupos erMsgKey erArgL "Bug!" optAttr)))
    (break)
    (ncAbort)))

———
13.2. MESSAGES

```lisp
(defun processMsgList
  (erMsgSort p369)
  [makeMsgFromLine p371]
  [poGlobalLinePosn p70]
  [getMsgPos p359]
  [queueUpErrors p372]
  [listOutputter p354]
  [$noRepList p??]
  [$outputList p??]

  — defun processMsgList —

  (defun |processMsgList| (erMsgList lineList)
    (let (|$noRepList| |$outputList| st globalNumOfLine msgLine)
      (declare (special |$noRepList| |$outputList|))
      (setq |$outputList| nil)
      (setq |$noRepList| nil)
      (setq erMsgList (|erMsgSort| erMsgList))
      (dolist (line lineList)
        (setq msgLine (|makeMsgFromLine| line))
        (setq |$outputList| (cons msgLine |$outputList|))
        (setq globalNumOfLine (|poGlobalLinePosn| (|getMsgPos| msgLine)))
        (setq erMsgList (|queueUpErrors| globalNumOfLine erMsgList)))
      (setq |$outputList| (append erMsgList |$outputList|))
      (setq st "---------SOURCE-TEXT-&-ERRORS------------------------")
      (|listOutputter| (reverse |$outputList|))))

——-

defun erMsgSort

  [erMsgSep p370]
  [listSort p??]

  — defun erMsgSort —

  (defun |erMsgSort| (erMsgList)
    (let (msgWOPos msgWPos tmp)
      (setq tmp (|erMsgSep| erMsgList))
      (setq msgWPos (car tmp))
      (setq msgWOPos (cadr tmp))
      (setq msgWPos (|listSort| #'|erMsgCompare| msgWPos))
      (setq msgWOPos (reverse msgWOPos))
      (append msgWOPos msgWPos))
```
defun erMsgCompare

(defun erMsgCompare (ob1 ob2)
  (compareposns (getMsgPos ob2) (getMsgPos ob1)))

defun compareposns

(defun compareposns (a b)
  (let (c d)
    (setq c (poGlobalLinePosn a))
    (setq d (poGlobalLinePosn b))
    (if (equal c d)
      (not (< (poCharPosn a) (poCharPosn b)))
      (not (< c d))))

defun erMsgSep

(defun erMsgSep (erMsgList)
  (let (msgWOPos msgWPos)
    (dolist (msg erMsgList)
      (if (poNopos? (getMsgPos msg))
        (setq msgWOPos (cons msg msgWOPos))
      ))
    )
13.2. MESSAGES

(setq msgWPos (cons msg msgWPos)))
(list msgWPos msgWPos))

defun makeMsgFromLine

(defun makeMsgFromLine (line)
(let (localNumOfLine stNum globalNumOfLine textOfLine posOfLine)
(declare (special $preLength))
(setq posOfLine (getLinePos line))
(setq textOfLine (getLineText line))
(setq globalNumOfLine (poGlobalLinePosn posOfLine))
(setq stNum (princ-to-string (poLinePosn posOfLine)))
(setq localNumOfLine
  (strconc (rep #\space (- $preLength 7 (size stNum))) stNum))
(list '|line| posOfLine nil nil (strconc "Line" localNumOfLine) textOfLine)))

defun rep

TPDHERE: This function should be replaced by fillerspaces

(defun rep 0)

(defun rep (c n)
(if (< 0 n)
  (make-string n :initial-element (character c))
  ""))
defun getLinePos

— defun getLinePos 0 —

(defun |getLinePos| (line) (car line))

——

defun getLineText

— defun getLineText 0 —

(defun |getLineText| (line) (cdr line))

——

defun queueUpErrors

;queueUpErrors(globalNumOfLine,msgList)==
;  thisPosMsgs := []
;  notThisLineMsgs := []
;  for msg in msgList _
;    while thisPosIsLess(getMsgPos msg,globalNumOfLine) repeat
;      --these are msgs that refer to positions from earlier compilations
;      if not redundant (msg,notThisPosMsgs) then
;        notThisPosMsgs := [msg,:notThisPosMsgs]
;        msgList := rest msgList
;      for msg in msgList _
;        while thisPosIsEqual(getMsgPos msg,globalNumOfLine) repeat
;          if not redundant (msg,thisPosMsgs) then
;            thisPosMsgs := [msg,:thisPosMsgs]
;            msgList := rest msgList
;        if thisPosMsgs then
;          thisPosMsgs := processChPosesForOneLine thisPosMsgs
;          $outputList := NCONC(thisPosMsgs,$outputList)
;        if notThisPosMsgs then
;          $outputList := NCONC(notThisPosMsgs,$outputList)
;      msgList

[processChPosesForOneLine p376]
[$outputList p??]

— defun queueUpErrors —
(DEFUN queueUpErrors (globalNumOfLine msgList)
  (PROG (notThisPosMsgs notThisLineMsgs thisPosMsgs)
    (DECLARE (SPECIAL $outputList))
    (RETURN
      (PROGN
        (SETQ thisPosMsgs NIL)
        (SETQ notThisLineMsgs NIL)
        ((LAMBDA (bfVar#7 msg)
          (LOOP
            (COND
              ((OR (ATOM bfVar#7) (PROGN (SETQ msg (CAR bfVar#7)) NIL) (NOT (thisPosIsLess (getMsgPos msg) globalNumOfLine)))
               (RETURN NIL))
              ('T (PROGN
                (COND
                  ((NULL (redundant msg notThisPosMsgs))
                   (SETQ notThisPosMsgs)
                   (CONS msg notThisPosMsgs)))
                  (SETQ msgList (CDR msgList))))
          (SETQ bfVar#7 (CDR bfVar#7)))))
        (SETQ notThisPosMsgs (CONS msg notThisPosMsgs))
        (SETQ msgList (CDR msgList))))
      (COND
        (thisPosMsgs)
        (SETQ thisPosMsgs (processChPosesForOneLine thisPosMsgs)
        (SETQ $outputList (NCONC thisPosMsgs $outputList))))
      (COND
        (notThisPosMsgs)
        (SETQ $outputList (NCONC notThisPosMsgs $outputList))))
    (msgList))))
-----

```lisp
defun thisPosIsLess
[poNopos? poGlobalLinePosn]

  (defun |thisPosIsLess| (pos num)
    (unless (|poNopos?| pos) (< (|poGlobalLinePosn| pos) num)))
```

-----

```lisp
defun thisPosIsEqual
[poNopos? poGlobalLinePosn]

  (defun |thisPosIsEqual| (pos num)
    (unless (|poNopos?| pos) (equal (|poGlobalLinePosn| pos) num)))
```

-----

```lisp
defun redundant
redundant(msg, thisPosMsgs) ==
  found := NIL
  if msgNoRep? msg then
    for item in $noRepList repeat
      sameMsg?(msg, item) => return (found := true)
    $noRepList := [msg, $noRepList]
  found or MEMBER(msg, thisPosMsgs)

[msgNoRep? sameMsg? $noRepList]
```

-----
(defun redundant (msg thisPosMsgs)
  (prog (found)
    (declare (special $noRepList))
    (return
      (progn
        (cond
          (msgNoRep? msg)
          ((lambda (Var9 item)
              (loop
                (cond
                  ((or (atom Var9) (progn (setq item (car Var9)) nil))
                    (return nil))
                  (t
                    (cond
                      ((sameMsg? msg item) (return (setq found t)))))))
              (setq Var9 (cdr Var9)))
            $noRepList nil)
          (setq $noRepList (list msg $noRepList)))
          (or found (member msg thisPosMsgs))))))

defvar $repGuys

  — initvars —

  (defvar $repGuys (list 'noRep 'rep))

  —

defun msgNoRep?

[getMsgCatAttr p358]

  — defun msgNoRep? —

  (defun msgNoRep? (msg) (eq (|getMsgCatAttr| msg '$repGuys) 'noRep)))
defun sameMsg?

| getMsgKey p348 |
| getMsgArgL p349 |

— defun sameMsg? —

(defun |sameMsg?| (msg1 msg2)
  (and (equal (|getMsgKey| msg1) (|getMsgKey| msg2))
       (equal (|getMsgArgL| msg1) (|getMsgArgL| msg2))))

defun processChPosesForOneLine

| posPointers p378 |
| getMsgFTTag? p359 |
| putFTText p379 |
| poCharPosn p377 |
| getMsgPos p359 |
| getMsgPrefix p349 |
| setMsgPrefix p349 |
| strconc p? |
| size p1045 |
| makeLeaderMsg p377 |
| $preLength p355 |

— defun processChPosesForOneLine —

(defun |processChPosesForOneLine| (msgList)
  (let (leaderMsg oldPre posLetter chPosList)
    (declare (special $preLength))
    (setq chPosList (posPointers msgList))
    (dolist (msg msgList)
      (when (getMsgFTTag? msg) (putFTText msg chPosList))
      (setq posLetter (cdr (assoc (poCharPosn (getMsgPos msg)) chPosList)))
      (setq oldPre (getMsgPrefix msg))
      (setq msgPrefix msg)
      (strconc oldPre
        (make-string (- $preLength 4 (size oldPre))) posLetter))
      (setq leaderMsg (makeLeaderMsg (list leaderMsg)))
    (nconc msgList (list leaderMsg))))
defun poCharPosn

— defun poCharPosn 0 —

(defun poCharPosn (posn)
  (cdr posn))

———

defun makeLeaderMsg

makeLeaderMsg chPosList ==
  st := MAKE_-FULL_-CVEC ($preLength- 3)
  oldPos := -1
  for [posNum,:posLetter] in reverse chPosList repeat
    st := STRCONC(st, 
      rep(char ".", (posNum - oldPos - 1)),posLetter)
    oldPos := posNum
  ['leader,$nopos,'nokey,NIL,NIL,[st] ]

[$nopos p26]
[$preLength p355]

— defun makeLeaderMsg —

(defun makeLeaderMsg! (chPosList)
  (let (posLetter posNum oldPos st)
    (declare (special $nopos |$preLength|))
    (setq st (make-string (- |$preLength| 3)))
    (setq oldPos -1)
    ((lambda (Var15 Var14)
      (loop
        (cond
          ((or (atom Var15) (progn (setq Var14 (car Var15)) nil))
            (return nil))
          (t
            (and (consp Var14)
              (progn
                (setq posNum (car Var14))
                (setq posLetter (cdr Var14))
                (t)
                (progn
                  (setq st
                    (strconc st (|rep| #\. (- posNum oldPos 1)) posLetter))
                  (setq oldPos posNum)))))))
    (setq Var15 (cdr Var15))))
(reverse chPosList) nil)
(list '|leader| |$nopos| 'nokey| nil nil (list st))))

defun posPointers

TPDHERE: getMsgFTTag is nonsense [poCharPosn p377]
[getMsgPos p359]
[IFCAR p??]
[getMsgPos2 p378]
[insertPos p379]
[getMsgFTTag p??]

— defun posPointers —

(defun |posPointers| (msgList)
(let (posLetterList pos ftPosList posList increment pointers)
(declare (special |getMsgFTTag|))
(setq pointers "ABCDEFGHIJKLMONPQRS")
(setq increment 0)
(dolist (msg msgList)
  (setq pos (|poCharPosn| (|getMsgPos| msg)))
  (unless (equal pos (ifcar posList))
    (setq posList (cons pos posList)))
  ; this should probably read TPDHERE
  ; (when (eq (|getMsgPosTagOb| msg) 'fromto))
  (when (eq |getMsgFTTag| 'fromto)
    (setq ftPosList (cons (|poCharPosn| (|getMsgPos2| msg)) ftPosList))))
(dolist (toPos ftPosList)
  (setq posList (|insertPos| toPos posList)))
(dolist (pos posList)
  (setq posLetterList
    (cons (cons pos (elt pointers increment)) posLetterList))
  (setq increment (+ increment 1)))
posLetterList))

——

defun getMsgPos2

[getMsgFTTag? p359]
[getMsgPosTagOb p348]
[ncBug p368]
— defun getMsgPos2 —

(defun getMsgPos2 (msg)
  (if (getMsgFTTag? msg)
      (caddr (getMsgPosTagOb msg))
      (ncBug "not a from to" nil)))

----------

defun insertPos

This function inserts a position in the proper place of a position list. This is used for the 2nd pos of a fromto [done p??]

— defun insertPos 0 —

(defun insertPos 0 (newPos posList)
  (let (pos top bot done)
    (setq bot (cons 0 posList))
    (do () (done)
      (setq top (cons (car bot) top))
      (setq bot (cdr bot))
      (setq pos (car bot))
      (setq done
        (cond
          ((< pos newPos) nil)
          ((equal pos newPos) t)
          ((< newPos pos)
            (setq top (cons newPos top))
            t))))
    (cons (cdr (reverse top)) bot)))

----------

defun putFTText

[getMsgFTTag? p359]
[poCharPosn p377]
[getMsgPos p359]
[setMsgText p349]
[getMsgText p349]
[getMsgPos2 p378]

— defun putFTText —
(defun putFTText (msg chPosList)
  (let ((charMarker2 pos2 markingText charMarker pos tag)
    (setq tag (getMsgFTTag? msg))
    (setq pos (poCharPosn (getMsgPos msg)))
    (setq charMarker (cdr (assoc pos chPosList)))
    (cond
      ((eq tag 'from)
       (setq markingText (list "(from " charMarker " and on) "))
       (setMsgText msg (append markingText (getMsgText msg))))
      ((eq tag 'to)
       (setq markingText (list "(up to " charMarker ") "))
       (setMsgText msg (append markingText (getMsgText msg))))
      ((eq tag 'fromto)
       (setq pos2 (poCharPosn (getMsgPos2 msg)))
       (setq charMarker2 (cdr (assoc pos2 chPosList)))
       (setq markingText (list "(from " charMarker " up to " charMarker2 ") "))
       (setMsgText msg (append markingText (getMsgText msg)))))))

defun From
This is called from parameter list of nc message functions
— defun From 0 —

(defun From (pos) (list 'from pos))

defun To
This is called from parameter list of nc message functions
— defun To 0 —

(defun To (pos) (list 'to pos))

defun FromTo
This is called from parameter list of nc message functions
— defun FromTo 0 —
(defun |FromTo| (pos1 pos2) (list 'fromto pos1 pos2))
Chapter 14

The Interpreter Syntax

14.1 syntax assignment

--- assignment.help ---

Immediate, Delayed, and Multiple Assignment

Immediate Assignment

A variable in Axiom refers to a value. A variable has a name beginning with an uppercase or lowercase alphabetic character, "%", or "!". Successive characters (if any) can be any of the above, digits, or "?". Case is distinguished. The following are all examples of valid, distinct variable names:

a tooBig? a1B2c3%!?
A %j numberOfPoints
beta6 %J numberofpoints

The ":=" operator is the immediate assignment operator. Use it to associate a value with a variable. The syntax for immediate assignment for a single variable is:

variable := expression

The value returned by an immediate assignment is the value of expression.

a := 1
1
Type: PositiveInteger
The right-hand side of the expression is evaluated, yielding 1. The value is then assigned to a.

\[
\text{b := a}
\]

\[
1
\]

Type: PositiveInteger
The right-hand side of the expression is evaluated, yielding 1. This value is then assigned to b. Thus a and b both have the value 1 after the sequence of assignments.

\[
\text{a := 2}
\]

\[
2
\]

Type: PositiveInteger
What is the value of b if a is assigned the value 2?

\[
\text{b}
\]

\[
1
\]

Type: PositiveInteger
The value of b is left unchanged.

This is what we mean when we say this kind of assignment is immediate. The variable b has no dependency on a after the initial assignment. This is the usual notion of assignment in programming languages such as C, Pascal, and Fortran.

====================================================================
Delayed Assignment
====================================================================

Axiom provides delayed assignment with "==". This implements a delayed evaluation of the right-hand side and dependency checking. The syntax for delayed assignment is

\[
\text{variable == expression}
\]

The value returned by a delayed assignment is the unique value of Void.

\[
\text{a == 1}
\]

Type: Void

\[
\text{b == a}
\]

Type: Void

Using a and b as above, these are the corresponding delayed assignments.
14.1. SYNTAX ASSIGNMENT

a
Compiling body of rule a to compute value of type PositiveInteger
1
  Type: PositiveInteger

The right-hand side of each delayed assignment is left unevaluated until
the variables on the left-hand sides are evaluated.

b
Compiling body of rule b to compute value of type PositiveInteger
1
  Type: PositiveInteger

This gives the same results as before. But if we change a to 2

a == 2
  Compiled code for a has been cleared.
  Compiled code for b has been cleared.
  1 old definition(s) deleted for function or rule a
  Type: Void

Then a evaluates to 2, as expected

a
Compiling body of rule a to compute value of type PositiveInteger
2
  Type: PositiveInteger

but the value of b reflects the change to a

b
Compiling body of rule b to compute value of type PositiveInteger
2
  Type: PositiveInteger

====================================================================
Multiple Immediate Assignments
====================================================================

It is possible to set several variables at the same time by using a
tuple of variables and a tuple of expressions. A tuple is a collection
of things separated by commas, often surrounded by parentheses. The
syntax for multiple immediate assignment is

( var1, var2, ..., varN ) := ( expr1, expr2, ..., exprN )

The value returned by an immediate assignment is the value of exprN.

( x, y ) := ( 1, 2 )
  2
CHAPTER 14. THE INTERPRETER SYNTAX

Type: PositiveInteger

This sets x to 1 and y to 2. Multiple immediate assignments are parallel in the sense that the expressions on the right are all evaluated before any assignments on the left are made. However, the order of evaluation of these expressions is undefined.

\[(x, y) := (y, x)\]

Type: PositiveInteger

\[x\]

Type: PositiveInteger

\[y\]

Type: PositiveInteger

The variable x now has the previous value of y.

\[y\]

Type: PositiveInteger

The variable y now has the previous value of x.

There is no syntactic form for multiple delayed assignments.

---

14.2 syntax blocks

--- blocks.help ---

====================================================================
Blocks
====================================================================

A block is a sequence of expressions evaluated in the order that they appear, except as modified by control expressions such as leave, return, iterate, and if-then-else constructions. The value of a block is the value of the expression last evaluated in the block.

To leave a block early, use ";=>". For example,

\[i < 0 \Rightarrow x\]

The expression before the ";=>" must evaluate to true or false. The expression following the ";=>" is the return value of the block.
A block can be constructed in two ways:

1. the expressions can be separated by semicolons and the resulting expression surrounded by parentheses, and
2. the expressions can be written on succeeding lines with each line indented the same number of spaces (which must be greater than zero).

A block entered in this form is called a pile

Only the first form is available if you are entering expressions directly to Axiom. Both forms are available in .input files. The syntax for a simple block of expressions entered interactively is

\[( \text{expression1}; \text{expression2}; \ldots; \text{expressionN} )\]

The value returned by a block is the value of an "\(\Rightarrow\)" expression, or \text{expressionN} if no "\(\Rightarrow\)" is encountered.

In .input files, blocks can also be written in piles. The examples given here are assumed to come from .input files.

\begin{verbatim}
a :=
i := \gcd(234, 672)
i := 2i**5 - i + 1
1 / i

1
-----
23323
\end{verbatim}

Type: Fraction Integer

In this example, we assign a rational number to a using a block consisting of three expressions. This block is written as a pile. Each expression in the pile has the same indentation, in this case two spaces to the right of the first line.

\begin{verbatim}
a := ( i := \gcd(234, 672); i := 2i**5 - i + 1; 1 / i )

1
-----
23323
\end{verbatim}

Type: Fraction Integer

Here is the same block written on one line. This is how you are required to enter it at the input prompt.

\begin{verbatim}
( a := 1; b := 2; c := 3; [a,b,c] )
[1,2,3]
\end{verbatim}

Type: List PositiveInteger
AAxiom gives you two ways of writing a block and the preferred way in an .input file is to use a pile. Roughly speaking, a pile is a block whose constituent expressions are indented the same amount. You begin a pile by starting a new line for the first expression, indenting it to the right of the previous line. You then enter the second expression on a new line, vertically aligning it with the first line. And so on. If you need to enter an inner pile, further indent its lines to the right of the outer pile. Axiom knows where a pile ends. It ends when a subsequent line is indented to the left of the pile or the end of the file.

Also See:
- )help if
- )help repeat
- )help while
- )help for
- )help suchthat
- )help parallel
- )help lists

14.3 system clef

— clef.help —

Entering printable keys generally inserts new text into the buffer (unless in overwrite mode, see below). Other special keys can be used to modify the text in the buffer. In the description of the keys below, “n means Control-n, or holding the CONTROL key down while pressing "n". Errors will ring the terminal bell.

`^A/^E`: Move cursor to beginning/end of the line.
`^F/^B`: Move cursor forward/backward one character.
`^D`: Delete the character under the cursor.
`^H, DEL`: Delete the character to the left of the cursor.
`^K`: Kill from the cursor to the end of line.
`^L`: Redraw current line.
`^O`: Toggle overwrite/insert mode. Initially in insert mode. Text added in overwrite mode (including yanks) overwrite existing text, while insert mode does not overwrite.
`^P/^N`: Move to previous/next item on history list.

14.4. SYNTAX COLLECTION

`^R/^S` : Perform incremental reverse/forward search for string on the history list. Typing normal characters adds to the current search string and searches for a match. Typing `^R/^S` marks the start of a new search, and moves on to the next match. Typing `^H` or DEL deletes the last character from the search string, and searches from the starting location of the last search. Therefore, repeated DEL's appear to unwind to the match nearest the point at which the last `^R` or `^S` was typed. If DEL is repeated until the search string is empty the search location begins from the start of the history list. Typing ESC or any other editing character accepts the current match and loads it into the buffer, terminating the search.

`^T` : Toggle the characters under and to the left of the cursor.

`^Y` : Yank previously killed text back at current location. Note that this will overwrite or insert, depending on the current mode.

`^U` : Show help (this text).

TAB : Perform command completion based on word to the left of the cursor. Words are deemed to contain only the alphanumeric and the `! ? _` characters.

NL, CR : returns current buffer to the program.

DOS and ANSI terminal arrow key sequences are recognized, and act like:

- up : same as `^P`
- down : same as `^N`
- left : same as `^B`
- right : same as `^F`

---

14.4 syntax collection

--- collection.help ---

====================================================================
Collection -- Creating Lists and Streams with Iterators
====================================================================

All of the loop expressions which do not use the repeat leave or iterate words can be used to create lists and streams. For example:

This creates a simple list of the integers from 1 to 10:

```plaintext
list := [i for i in 1..10]
[1,2,3,4,5,6,7,8,9,10]
```

Type: List PositiveInteger
Create a stream of the integers greater than or equal to 1:

\[
\text{stream} := [\text{i for i in 1..}]
\]

\[
[1,2,3,4,5,6,7,\ldots]
\]

Type: Stream PositiveInteger

This is a list of the prime numbers between 1 and 10, inclusive:

\[
[i \text{ for } i \text{ in } 1..10 \text{ | prime? } i]
\]

\[
[2,3,5,7]
\]

Type: List PositiveInteger

This is a stream of the prime integers greater than or equal to 1:

\[
[i \text{ for } i \text{ in } 1.. \text{ | prime? } i]
\]

\[
[2,3,5,7,11,13,17,\ldots]
\]

Type: Stream PositiveInteger

This is a list of the integers between 1 and 10, inclusive, whose squares are less than 700:

\[
[i \text{ for } i \text{ in } 1..10 \text{ while } i*i < 700]
\]

\[
[1,2,3,4,5,6,7,8,9,10]
\]

Type: List PositiveInteger

This is a stream of the integers greater than or equal to 1 whose squares are less than 700:

\[
[i \text{ for } i \text{ in } 1.. \text{ while } i*i < 700]
\]

\[
[1,2,3,4,5,6,7,\ldots]
\]

Type: Stream PositiveInteger

The general syntax of a collection is

\[
[\text{collectExpression iterator1 iterator2 ... iteratorN}]
\]

where each iterator is either a for or a while clause. The loop terminates immediately when the end test of any iterator succeeds or when a return expression is evaluated in collectExpression. The value returned by the collection is either a list or a stream of elements, one for each iteration of the collectExpression.

Be careful when you use while to create a stream. By default Axiom tries to compute and display the first ten elements of a stream. If the while condition is not satisfied quickly, Axiom can spend a long (potentially infinite) time trying to compute the elements. Use

\[
)\text{set streams calculate}
\]
to change the defaults to something else. This also affects the number of terms computed and displayed for power series. For the purposes of these examples we have use this system command to display fewer than ten terms.

---

### 14.5 Syntax for

--- for.help ---

```plaintext
====================================================================
for loops
====================================================================

Axiom provide the for and in keywords in repeat loops, allowing you to integrate across all elements of a list, or to have a variable take on integral values from a lower bound to an upper bound. We shall refer to these modifying clauses of repeat loops as for clauses. These clauses can be present in addition to while clauses (See `help while`). As with all other types of repeat loops, leave (see `help leave`) can be used to prematurely terminate evaluation of the loop.

The syntax for a simple loop using for is

```plaintext
for iterator repeat loopbody
```

The iterator has several forms. Each form has an end test which is evaluated before loopbody is evaluated. A for loop terminates immediately when the end test succeeds (evaluates to true) or when a leave or return expression is evaluated in loopbody. The value returned by the loop is the unique value of Void.

```plaintext
for i in n..m repeat
```

If for is followed by a variable name, the in keyword and then an integer segment of the form n..m, the end test for this loop is the predicate i > m. The body of the loop is evaluated m-n+1 times if this number is greater than 0. If this number is less than or equal to 0, the loop body is not evaluated at all.

The variable i has the value n, n+1, ..., m for successive iterations of the loop body. The loop variable is a local variable within the loop body. Its value is not available outside the loop body and its value and
type within the loop body completely mask any outer definition of a variable with the same name.

```plaintext
for i in 10..12 repeat output(i**3)
1000
1331
1728
```

Type: Void

The loop prints the values of $10^3$, $11^3$, and $12^3$.

```plaintext
a := [1,2,3]
[1,2,3]
```

Type: List PositiveInteger

```plaintext
for i in 1..#a repeat output(a.i)
1
2
3
```

Type: Void

Iterate across this list using "." to access the elements of a list and the # operation to count its elements.

This type of iteration is applicable to anything that uses ".". You can also use it with functions that use indices to extract elements.

```plaintext
m := matrix [ [1,2],[4,3],[9,0] ]
+- -+
| 1 2 |
| 4 3 |
| 9 0 |
+- -+
```

Type: Matrix Integer

Define m to be a matrix.

```plaintext
for i in 1..nrows(m) repeat output row(m.i)
[1,2]
[4,3]
[9,0]
```

Type: Void

Display the rows of m.

You can iterate with for-loops.

```plaintext
for i in 1..5 repeat
  if odd?(i) then iterate
  output(i)
```
14.5. SYNTAX FOR

Display the even integers in a segment.

```
for i in n..m by s repeat
```

By default, the difference between values taken on by a variable in
loops such as

```
for i in n..m repeat ...
```

is 1. It is possible to supply another, possibly negative, step value
by using the by keyword along with for and in. Like the upper and lower
bounds, the step value following the by keyword must be an integer. Note
that the loop

```
for i in 1..2 by 0 repeat output(i)
```

will not terminate by itself, as the step value does not change the
index from its initial value of 1.

```
for i in 1..5 by 2 repeat output(i)
  1
  3
  5
```

Type: Void

This expression displays the odd integers between two bounds.

```
for i in 5..1 by -2 repeat output(i)
  5
  3
  1
```

Type: Void

Use this to display the numbers in reverse order.

```
for i in n.. repeat
```

If the value after the ".." is omitted, the loop has no end test. A
potentially infinite loop is thus created. The variable is given the
successive values n, n+1, n+2, ... and the loop is terminated only
if a leave or return expression is evaluated in the loop body. However,
you may also add some other modifying clause on the repeat, for example,
a while clause, to stop the loop.

```
for i in 15.. while not prime?(i) repeat output(i)
  15
  16
  Type: Void
```

This loop displays the integers greater than or equal to 15 and less than the first prime number greater than 15.

```
for x in l repeat
```

Another variant of the for loop has the form:

```
for x in list repeat loopbody
```

This form is used when you want to iterate directly over the elements of a list. In this form of the for loop, the variable x takes on the value of each successive element in l. The end test is most simply stated in English: "are there no more x in l?"

```
l := [0, -5, 3]
[0, -5, 3]
  Type: List Integer
```

```
for x in l repeat output(x)
  0
  -5
  3
  Type: Void
```

This displays all of the elements of the list l, one per line.

Since the list constructing expression

```
expand [n..m]
```

creates the list

```
[n, n+1, ..., m]
```

you might be tempted to think that the loops

```
for i in n..m repeat output(i)
```

and

```
for x in expand [n..m] repeat output(x)
```
are equivalent. The second form first creates the expanded list
(no matter how large it might be) and then does the iteration. The
first form potentially runs in much less space, as the index variable
i is simply incremented once per loop and the list is not actually
created. Using the first form is much more efficient.

Of course, sometimes you really want to iterate across a specific list.
This displays each of the factors of 2400000:

```axiom
for f in factors(factor(2400000)) repeat output(f)
[factor= 2, exponent= 8]
[factor= 3, exponent= 1]
[factor= 5, exponent= 5]
Type: Void
```

---

### 14.6 syntax if

--- if.help ---

====================================================================

**If-then-else**

====================================================================

Like many other programming languages, Axiom uses the three keywords
if, then, and else to form conditional expressions. The else part of
the conditional is optional. The expression between the if and then
keywords is a predicate: an expression that evaluates to or is
convertible to either true or false, that is, a Boolean.

The syntax for conditional expressions is

```axiom
if predicate then expression1 else expression2
```

where the "else expression2" part is optional. The value returned from
a conditional expression is expression1 if the predicate evaluates to
true and expression2 otherwise. If no else clause is given, the value
is always the unique value of Void.

An if-then-else expression always returns a value. If the else clause
is missing then the entire expression returns the unique value of Void.
If both clauses are present, the type of the value returned by if is
obtained by resolving the types of the values of the two clauses.
The predicate must evaluate to, or be convertible to, an object of type Boolean: true or false. By default, the equal sign "=" creates an equation.

\[
\begin{align*}
x + 1 &= y \\
x + 1 &= y
\end{align*}
\]

Type: Equation Polynomial Integer

This is an equation, not a boolean condition. In particular, it is an object of type Equation Polynomial Integer.

However, for predicates in if expressions, Axiom places a default target type of Boolean on the predicate and equality testing is performed. Thus you need not qualify the "=" in any way. In other contexts you may need to tell Axiom that you want to test for equality rather than create an equation. In these cases, use "@" and a target type of Boolean.

The compound symbol meaning "not equal" in Axiom is "~=". This can be used directly without a package call or a target specification. The expression "a ~= b" is directly translated to "not(a = b)".

Many other functions have return values of type Boolean. These include <, <=, >, >=, ~=, and member?. By convention, operations with names ending in "?" return Boolean values.

The usual rules for piles are suspended for conditional expressions. In .input files, the then and else keywords can begin in the same column as the corresponding if by may also appear to the right. Each of the following styles of writing if-then-else expressions is acceptable:

```plaintext
if i>0 then output("positive") else output("nonpositive")

if i>0 then output("positive")
  else output("nonpositive")

if i>0 then output("positive")
  else output("nonpositive")

if i>0
  then output("positive")
  else output("nonpositive")

if i>0
  then output("positive")
  else output("nonpositive")
```

A block can follow the then or else keywords. In the following two assignments to a, the then and else clauses each are followed by two line piles. The value returned in each is the value of the second line.

```plaintext
a :=
```
14.7 syntax iterate

--- iterate.help ---

iterate in loops

Axiom provides an iterate expression that skips over the remainder of a loop body and starts the next loop execution. We first initialize a counter.

i := 0

Type: NonNegativeInteger

Display the even integers from 2 to 5:

repeat
  i := i + 1
  if i > 5 then leave

These are both equivalent to the following:

a :=
  if i > 0 then
    j := sin(i * pi()); exp(j + 1/j)
  else
    j := cos(i * 0.5 * pi()); log(abs(j)**5 + i)

These are both equivalent to the following:

a :=
  if i > 0 then (j := sin(i * pi()); exp(j + 1/j))
  else (j := cos(i * 0.5 * pi()); log(abs(j)**5 + i))
CHAPTER 14. THE INTERPRETER SYNTAX

if odd?(i) then iterate
output(i)
2
4

Type: Void

14.8 syntax leave

--- leave.help ---

====================================================================
leave in loops
====================================================================

The leave keyword is often more useful in terminating a loop. A leave causes control to transfer to the expression immediately following the loop. As loops always return the unique value of Void, you cannot return a value with leave. That is, leave takes no argument.

f() ==
i := 1
repeat
  if factorial(i) > 1000 then leave
  i := i + 1
i

Type: Void

This example is a modification of the last example in the previous section. Instead of using return we’ll use leave.

f()
7

Type: PositiveInteger

The loop terminates when factorial(i) gets big enough. The last line of the function evaluates to the corresponding "good" value of i and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Lets consider a loop within a loop, that is, a loop with a nested loop. First, we initialize two counter variables.

(i,j) := (1,1)
**14.9 SYNTAX PARALLEL**

Type: PositiveInteger

```
repeat
  repeat
    if (i + j) > 10 then leave
    j := j + 1
    if (i + j) > 10 then leave
    i := i + 1

Type: Void
```

Nested loops must have multiple leave expressions at the appropriate nesting level. How would you rewrite this so \((i + j) > 10\) is only evaluated once?

```
leave vs => in loop bodies
```

---

**parallel iteration**

---

**14.9 syntax parallel**

--- parallel.help ---

--- parallel iteration ---
Sometimes you want to iterate across two lists in parallel, or perhaps you want to traverse a list while incrementing a variable.

The general syntax of a repeat loop is

\[
\text{iterator1, iterator2, ..., iteratorN repeat loopbody}
\]

where each iterator is either a for or a while clause. The loop terminates immediately when the end test of any iterator succeeds or when a leave or return expression is evaluated in loopbody. The value returned by the loop is the unique value of Void.

\[
\begin{align*}
l &:= \{1,3,5,7\} \\
&\quad \text{Type: List PositiveInteger} \\
\end{align*}
\]

\[
\begin{align*}
m &:= \{100,200\} \\
&\quad \text{Type: List PositiveInteger} \\
\end{align*}
\]

\[
\begin{align*}
sum &:= 0 \\
&\quad \text{Type: NonNegativeInteger} \\
\end{align*}
\]

Here we write a loop to iterate across two lists, computing the sum of the pairwise product of the elements:

\[
\begin{align*}
\text{for } x \text{ in } l \text{ for } y \text{ in } m \text{ repeat} \\
&\quad \text{sum} := \text{sum} + x*y \\
&\quad \text{Type: Void} \\
\end{align*}
\]

The last two elements of \( l \) are not used in the calculation because \( m \) has two fewer elements than \( l \).

\[
\begin{align*}
\text{sum} \\
700 \\
&\quad \text{Type: NonNegativeInteger} \\
\end{align*}
\]

This is the "dot product".

Next we write a loop to compute the sum of the products of the loop elements with their positions in the loop.

\[
\begin{align*}
l &:= \{2,3,5,7,11,13,17,19,23,29,31,37\} \\
&\quad \text{Type: List PositiveInteger} \\
\end{align*}
\]

\[
\begin{align*}
\text{sum} &:= 0 \\
&\quad \text{Type: List PositiveInteger} \\
\end{align*}
\]

\[
\begin{align*}
\text{sum} \\
0 \\
\end{align*}
\]
14.9. SYNTAX PARALLEL

Type: NonNegativeInteger

for i in 0.. for x in l repeat sum := i * x
Type: Void

Here looping stops when the list l is exhausted, even though the
for i in 0.. specifies no terminating condition.

sum
Type: NonNegativeInteger

When "|" is used to qualify any of the for clauses in a parallel
iteration, the variables in the predicates can be from an outer
scope or from a for clause in or to the left of the modified clause.

This is correct:

for i in 1..10 repeat
  for j in 200..300 | ood? (i+j) repeat
    output [i,j]

But this is not correct. The variable j has not been defined outside
the inner loop:

for i in 1..01 | odd? (i+j) repeat -- wrong, j not defined
  for j in 200..300 repeat
    output [i,j]

It is possible to mix several of repeat modifying clauses on a loop:

for i in 1..10
  for j in 151..160 | odd? j
    while i + j < 160 repeat
      output [i,j]
[1,151]
[3,153]
Type: Void

Here are useful rules for composing loop expressions:

1. while predicates can only refer to variables that are global (or
   in an outer scope) or that are defined in for clauses to the left
   of the predicate.
2. A "such that" predicate (something following "|") must directly
   follow a for clause and can only refer to variables that are
   global (or in an outer scope) or defined in the modified for clause
   or any for clause to the left.
14.10 syntax repeat

--- repeat.help ---

====================================================================
Repeat Loops
====================================================================

A loop is an expression that contains another expression, called the loop body, which is to be evaluated zero or more times. All loops contain the repeat keyword and return the unique value of Void. Loops can contain inner loops to any depth.

The most basic loop is of the form

    repeat loopbody

Unless loopbody contains a leave or return expression, the loop repeats forever. The value returned by the loop is the unique value of Void.

Axiom tries to determine completely the type of every object in a loop and then to translate the loop body to Lisp or even to machine code. This translation is called compilation.

If Axiom decides that it cannot compile the loop, it issues a message stating the problem and then the following message:

    We will attempt to step through and interpret the code

It is still possible that Axiom can evaluate the loop but in interpret-code mode.

====================================================================
Return in Loops
====================================================================

A return expression is used to exit a function with a particular value. In particular, if a return is in a loop within the function, the loop is terminated whenever the return is evaluated.

    f() ==
    i := 1
    repeat
        if factorial(i) > 1000 then return i
    i := i + 1
Type: Void

f()
Type: Void

When factorial(i) is big enough, control passes from inside the loop all the way outside the function, returning the value of i (so we think). What went wrong? Isn't it obvious that this function should return an integer? Well, Axiom makes no attempt to analyze the structure of a loop to determine if it always returns a value because, in general, this is impossible. So Axiom has this simple rule: the type of the function is determined by the type of its body, in this case a block. The normal value of a block is the value of its last expression, in this case, a loop. And the value of every loop is the unique value of Void. So the return type of f is Void.

There are two ways to fix this. The best way is for you to tell Axiom what the return type of f is. You do this by giving f a declaration

f():() -> Integer

prior to calling for its value. This tells Axiom "trust me -- an integer is returned". Another way is to add a dummy expression as follows.

f() ==
  i := 1
  repeat
    if factorial(i) > 1000 then return i
    i := i + 1
  0

Type: Void

Note that the dummy expression will never be evaluated but it is the last expression in the function and will determine the return type.

f()
7

Type: PositiveInteger

leave in loops

The leave keyword is often more useful in terminating a loop. A leave causes control to transfer to the expression immediately following the loop. As loops always return the unique value of Void, you cannot return a value with leave. That is, leave takes no argument.

f() ==
i := 1
repeat
  if factorial(i) > 1000 then leave
  i := i + 1
i
  Type: Void

This example is a modification of the last example in the previous section. Instead of using return we’ll use leave.

f()
7

Type: PositiveInteger

The loop terminates when factorial(i) gets big enough. The last line of the function evaluates to the corresponding "good" value of i and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Let’s consider a loop within a loop, that is, a loop with a nested loop. First, we initialize two counter variables.

(i,j) := (1,1)
1

Type: PositiveInteger

repeat
  repeat
    if (i + j) > 10 then leave
    j := j + 1
    if (i + j) > 10 then leave
    i := i + 1
  Type: Void

Nested loops must have multiple leave expressions at the appropriate nesting level. How would you rewrite this so (i + j) > 10 is only evaluated once?

====================================================================
leave vs => in loop bodies
====================================================================

Compare the following two loops:

i := 1
repeat
  i := i + 1
  i > 3 => i
  output(i)

i := 1
repeat
  i := i + 1
  if i > 3 then leave
  output(i)

In the example on the left, the values 2 and 3 for i are displayed but
then the "+=" does not allow control to reach the call to output again. The loop will not terminate until you run out of space or interrupt the execution. The variable \( i \) will continue to be incremented because the "+=" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed, and both the block and the loop will terminate. This is one of the reasons why both "+=" and leave are provided. Using a while clause with the "+=" lets you simulate the action of leave.

---

iterate in loops
---

Axiom provides an iterate expression that skips over the remainder of a loop body and starts the next loop execution. We first initialize a counter.

\[
i := 0
\]

\[
0
\]

\[
\text{Type: NonNegativeInteger}
\]

Display the even integers from 2 to 5:

\[
\begin{align*}
\text{repeat} \\
i := i + 1 \\
\text{if } i > 5 \text{ then leave} \\
\text{if } \text{odd?}(i) \text{ then iterate} \\
\text{output}(i) \\
2 \\
4
\end{align*}
\]

\[
\text{Type: Void}
\]

Also See:
- `)help blocks`
- `)help if`
- `)help while`
- `)help for`
- `)help suchthat`
- `)help parallel`
- `)help lists`

---

14.11 syntax suchthat

— suchthat.help —

A for loop can be followed by a "|" and then a predicate. The predicate qualifies the use of the values from the iterator that follows the for. Think of the vertical bar "|" as the phrase "such that".

```plaintext
for n in 0..4 | odd? n repeat output n
1
3
Type: Void
```

This loop expression prints out the integers n in the given segment such that n is odd.

A for loop can also be written

```plaintext
for iterator | predicate repeat loopbody
```

which is equivalent to:

```plaintext
for iterator repeat if predicate then loopbody else iterate
```

The predicate need not refer only to the variable in the for clause. Any variable in an outer scope can be part of the predicate.

```plaintext
for i in 1..50 repeat
    for j in 1..50 | factorial(i+j) < 25 repeat
        output [i,j]
    [1],1
    [1],2
    [1],3
    [2],1
    [2],2
    [3],1
Type: Void
```
The Axiom Interactive Language has the following features documented here.

More information is available by typing

)help feature

where feature is one of:

 assignment -- Immediate and delayed assignments
 blocks -- Blocks of expressions
 collection -- creating lists with iterators
 for -- for loops
 if -- If-then-else statements
 iterate -- using iterate in loops
 leave -- using leave in loops
 parallel -- parallel iterations
 repeat -- repeat loops
 suchthat -- suchthat predicates
 while -- while loops

---

The repeat in a loop can be modified by adding one or more while clauses. Each clause contains a predicate immediately following the while keyword. The predicate is tested before the evaluation of the body of the loop. The loop body is evaluated whenever the predicate in a while clause is true.

The syntax for a simple loop using while is

while predicate repeat loopbody
The predicate is evaluated before loopbody is evaluated. A while loop terminates immediately when predicate evaluates to false or when a leave or return expression is evaluated. See \texttt{\textbackslash help repeat} for more information on leave and return.

Here is a simple example of using while in a loop. We first initialize the counter.

\begin{verbatim}
i := 1
1
Type: PositiveInteger
while i < 1 repeat
  output "hello"
i := i + 1
Type: Void
\end{verbatim}

The steps involved in computing this example are
(1) set i to 1
(2) test the condition i < 1 and determine that it is not true
(3) do not evaluate the loop body and therefore do not display "hello"

\begin{verbatim}
(x, y) := (1, 1)
1
Type: PositiveInteger
while x < 4 and y < 10 repeat
  if x + y > 7 then leave
\end{verbatim}

A leave expression can be included in a loop body to terminate a loop even if the predicate in any while clauses are not false.

\begin{verbatim}
(x, y) := (1, 1)
1
Type: PositiveInteger
while x < 4 and y < 10 repeat
  if x + y > 7 then leave
\end{verbatim}
output [x,y]
x := x + 1
y := y + 2
[1,1]
[2,3]

Type: Void
Chapter 15

Abstract Syntax Trees (ptrees)

Abstract Syntax Trees

These functions create and examine abstract syntax trees. These are called pform, for short.

!! This file also contains constructors for concrete syntax, although they should be somewhere else.

THE PFORM DATA STRUCTURE

Leaves: [hd, tok, pos]
Trees: [hd, tree, tree, ...]
hd is either an id or (id . alist)

defun Construct a leaf token

The tokConstruct function is a constructor and selectors for leaf tokens. A leaf token looks like [head, token, position] where head is either an id or (id . alist) [ifcar p??]
[pfNoPosition? p412]
[ncPutQ p416]

— defun tokConstruct —

(defun |tokConstruct| (head token &rest position)
  (let (result)
    (setq result (cons head token))
    (cond
      ((ifcar position)
       (cond
        ((|pfNoPosition?| (car position)) result)
        (t (|ncPutQ| result '|posn| (car position)) result))))
(t result))

-----

defun Return a part of a node
[ifcar p??]

— defun pfAbSynOp —

(defun |pfAbSynOp| (form)
(let (hd)
  (setq hd (car form))
  (or (ifcar hd) hd)))

-----

defun Compare a part of a node
[eqcar p??]

— defun pfAbSynOp? —

(defun |pfAbSynOp?| (form op)
(let (hd)
  (setq hd (car form))
  (or (eq hd op) (eqcar hd op))))

-----

defun pfNoPosition?
[poNoPosition? p413]

— defun pfNoPosition? —

(defun |pfNoPosition?| (pos)
  (|poNoPosition?| pos))

-----
defun poNoPosition?
[eqcar p??]

— defun poNoPosition? 0 —

(defun |poNoPosition?| (pos)
  (eqcar pos ’|noposition|))

—

defun tokType
[ncTag p415]

— defun tokType —

(defun |tokType| (x) (|ncTag| x))

—

defun tokPart

— defun tokPart 0 —

(defun |tokPart| (x) (cdr x))

—

defun tokPosn
[qassq p??]
[ncAlist p415]
[pfNoPosition p414]

— defun tokPosn —

(defun |tokPosn| (x)
  (let (a)
    (setq a (qassq ’|posn| (|ncAlist| x)))
    (cond
(a (cdr a))
(t (|pfNoPosition|))))

---

defun pfNoPosition

[poNoPosition p414]

— defun pfNoPosition —

(defun |pfNoPosition| () (|poNoPosition|))

---

defun poNoPosition

[$nopos p26]

— defun poNoPosition 0 —

(defun |poNoPosition| ()
  (declare (special |$nopos|))
  |$nopos|)

———
Chapter 16

Attributed Structures

For objects which are pairs where the CAR field is either just a tag (an identifier) or a pair which is the tag and an association list.

defun ncTag

Pick off the tag [ncBug p368]
[qcar p??]
[identp p1046]

— defun ncTag —

(defun ncTag (x)
  (cond
    ((null (consp x)) (ncBug 's2cb0031 nil))
    (t
      (setq x (qcar x))
      (cond
        ((identp x) x)
        ((null (consp x)) (ncBug 's2cb0031 nil))
        (t (qcar x))))))

defun ncAlist

Pick off the property list [ncBug p368]
[qcar p??]
[identp p1046]
[qcdr p??]
--- \texttt{defun ncAlist} ---

(defun \texttt{ncAlist} (x)
  (cond
   ((null (consp x)) (\texttt{ncBug} 's2cb0031 nil))
   (t
    (setq x (qcar x))
    (cond
     ((identp x) nil)
     ((null (consp x)) (\texttt{ncBug} 's2cb0031 nil))
     (t (qcdr x))))))

---

defun \texttt{ncEltQ}

Get the entry for key \texttt{k} on \texttt{x}'s association list [qassq //??//]
\[\texttt{ncAlist p415}\]
\[\texttt{ncBug p368}\]

--- \texttt{defun ncEltQ} ---

(defun \texttt{ncEltQ} (x k)
  (let (r)
    (setq r (qassq k (\texttt{ncAlist} x)))
    (cond
     ((null r) (\texttt{ncBug} 's2cb0007 (list k)))
     (t (cdr r))))))

---

defun \texttt{ncPutQ}

;-- Put \texttt{(k . v)} on the association list of \texttt{x} and return \texttt{v}
;-- case1: \texttt{ncPutQ(x,k,v)} where \texttt{k} is a key (an identifier), \texttt{v} a value
;-- put the pair \texttt{(k . v)} on the association list of \texttt{x} and return \texttt{v}
;-- case2: \texttt{ncPutQ(x,k,v)} where \texttt{k} is a list of keys, \texttt{v} a list of values
;-- equivalent to [\texttt{ncPutQ(x,key,val)} for key in \texttt{k} for val in \texttt{v}]
;\texttt{ncPutQ(x,k,v)} ==
; LISTP \texttt{k} =>
; for key in \texttt{k} for val in \texttt{v} repeat \texttt{ncPutQ(x,key,val)}
; \texttt{v}
; \texttt{r} := QASSQ(k,ncAlist x)
; if NULL \texttt{r} then
(defun |ncPutQ| (x k v)
  (let (r)
    (cond
      ((listp k)
        ((lambda (Var1 key Var2 val)
            (loop
              (cond
                ((or (atom Var1)
                    (progn (setq key (car Var1)) nil)
                    (atom Var2)
                    (progn (setq val (car Var2)) nil))
                  (return nil))
                (t
                  (|ncPutQ| x key val)))
              (setq Var1 (cdr Var1))
              (setq Var2 (cdr Var2))))
        k nil v nil)
        v)
      (t
        (setq r (qassq k (|ncAlist| x)))
        (cond
          ((null r)
            (setq r (cons (cons k v) (|ncAlist| x)))
            (rplaca x (cons (|ncTag| x) r)))
          (t
            (rplacd r v))))
    v))))
Chapter 17

Function Selection

New Selection of Modemaps

selection of applicable modemaps is done in two steps: first it tries to find a modemap inside an argument domain, and if this fails, by evaluation of pattern modemaps

the result is a list of functions with signatures, which have the following form:

[sig, elt, cond] where

- sig is the signature gained by evaluating the modemap condition
- elt is the slot number to get the implementation
- cond are runtime checks which are the results of evaluating the modemap condition

the following flags are used:

- $Coerce is NIL, if function selection is done which requires exact matches (e.g. for coercion functions)
- if $SubDom is true, then runtime checks have to be compiled

defun ofCategory

(defun ofCategory
 (dom cat)
 (let ((|$Subst| |$hope|)
  (declare (special |$Subst| |$hope|)))

419
CHAPTER 17. FUNCTION SELECTION

(cond
  ((identp dom) nil)
  ((and (listp cat) (eq (car cat) '|Join|))
   (every #'(lambda (c) (|ofCategory| dom c)) (cdr cat)))
  (t (not (eq (|hasCaty| dom cat nil) '|failed|))))

---

(defun isPartialMode
  The isPartialMode function tests whether m contains $EmptyMode$. The constant $EmptyMode$ evaluates to $EmptyMode$. This constant is inserted in a modemap during compile time if the modemap is not yet complete. [contained p??]

(defun isPartialMode (m)
  (declare (special $EmptyMode))
  (contained $EmptyMode m))

---

(defun hasCaty
  This calls hasCat, which looks up a hashtable and returns:

  1. T, NIL or a (has x1 x2) condition, if cat is not parameterized
  2. a list of pairs (argument to cat,condition) otherwise

then the substitution sl is augmented, or the result is 'failed [hasAttSig p426]
[subCopy p??]
[constructSubst p434]
[hasSig p423]
[hasAtt p424]
[hasCat p??]
[opOf p??]
[kdr p??]
[mkDomPvar p433]
[domArg p422]
[augmentSub p??]
[domArg2 p422]
[unifyStruct p428]
[hasCaty1 p432]
— defun hasCaty —

(defun hasCaty (d cat sl)
  (let (x y S z cond sp dom zp s1 ncond i)
    (declare (special $domPvar$))
    (cond
      ((and (consp cat) (eq (qcar cat) 'category) (consp (qcdr cat)))
       (hasAttSig| d (subCopy| (qcddr cat) (constructSubst| d) sl))
      ((and (consp cat) (eq (qcar cat) 'signature) (consp (qcddr cat)))
       (eq (qcdadr cat) nil))
      ((and (consp cat) (eq (qcar cat) 'attribute)
       (eq (qcdadr cat) nil)))
      (hasAtt| d (subCopy| (qcadr cat) (constructSubst| d) sl))
      ((setq x (hasCat| (opOf| d) (opOf| cat)))
       (cond
        ((setq y (kdr cat))
         (setq s (constructSubst| d))
         (do ((next x (cdr next)) (endtest nil (null (eq s1 '|failed|))))
             ((or (atom next) endtest) nil)
           (setq z (caar next))
           (setq cond (cdar next))
           (setq sp
             (loop for item in s
               collect (cons (car item) (mkDomPvar| (car item) (cdr item) z y))))
         (when $domPvar$
           (setq i -1)
           (setq dom
             (cons (car d)
               (loop for arg in (rest d)
                 collect (domArg| arg (incf i) z y))))
         (setq sl1 (augmentSub| $domPvar| dom (copy sl))))
       (setq zp
         (loop for a in z
           collect (domArg2| a s sp))
       (setq sl1 (unifyStruct| y zp (copy sl)))
       (cond
         ((null (eq s1 '|failed|))
           (setq sl1
             (cond
              ((atom cond) s1)
              (t
               (setq ncond (subCopy| cond s))
               (cond
                ((and (consp ncond) (eq (qcar ncond) 'has))
                 (eq (qcdadr ncond) nil)
                 (eq (qcdadr ncond) nil)
                 (eq (qcdadr ncond) nil))
                (cond
                 ((and (consp ncond) (eq (qcar ncond) 'has))
                  (equal (qcdadr ncond) d)
                  (eq (qcdadr ncond) nil)
                  (eq (qcdadr ncond) nil)))
                (cond
                 ((and (consp ncond) (eq (qcar ncond) 'has))
                  (equal (qcdadr ncond) nil)
                  (equal (qcdadr ncond) nil))))
         (setq sl1)
         (cond
          ((atom cond) s1)
          (t
           (setq ncond (subCopy| cond s))
           (cond
            ((and (consp ncond) (eq (qcar ncond) 'has))
             (equal (qcdadr ncond) d)
             (eq (qcdadr ncond) nil)
             (eq (qcdadr ncond) nil))
            (cond
             ((and (consp ncond) (eq (qcar ncond) 'has))
              (equal (qcdadr ncond) nil)
              (equal (qcdadr ncond) nil))))
          (setq sl1)
          (cond
           ((atom cond) s1)
           (t
            (setq ncond (subCopy| cond s))
            (cond
             ((and (consp ncond) (eq (qcar ncond) 'has))
              (equal (qcdadr ncond) d)
              (eq (qcdadr ncond) nil)
              (eq (qcdadr ncond) nil))))
          (setq sl1)
          (cond
           ((atom cond) s1)
           (t
            (setq ncond (subCopy| cond s))
            (cond
             ((and (consp ncond) (eq (qcar ncond) 'has))
              (equal (qcdadr ncond) d)
              (eq (qcdadr ncond) nil)
              (eq (qcdadr ncond) nil))))
          (setq sl1)
CHAPTER 17. FUNCTION SELECTION

(defun domArg
  (type i subs y)
  (let (p)
    (declare (special $FormalMapVariableList))
    (if (setq p (member (elt $FormalMapVariableList i) subs))
      (elt y (- (length subs) (length p)))
      type))))

(defun domArg2
  (arg sl1 sl2)
  (declare (special $domPvar))
  (cond
    (isSharpVar arg) (subCopy arg sl1))
    (and (eq arg '$) $domPvar)))
defun hasSig

The function hasSig tests whether domain dom has function foo with signature sig under substitution sl. [constructor? p??] [constructSubst p??] [assq p1050] [getOperationAlistFromLisplib p??] [hasCate p433] [subCopy p??] [hasSigAnd p125] [hasSigOr p126] [keyedSystemError p??] [unifyStruct p428] [$domPvar p47]

— defun hasSig —

(defun |hasSig| (dom foo sig sl)
  (let ((|$domPvar| fun s0 p x cond s)
        (declare (special |$domPvar|)))
    (cond
      ((setq fun (|constructor?| (car dom)))
       (setq s0 (|constructSubst| dom))
       (cond
        ((setq p (assq foo (|getOperationAlistFromLisplib| (car dom))))
         (do ((next (cdr p) (cdr next))
              (endtest nil (null (eq s |failed|))))
             ((or (atom next) endtest) nil)
             (setq x (caar next))
             (setq cond (caddar next))
             (setq s
              (cond
              (atom cond) (copy s1))
             ((and (consp cond) (eq (qcar cond) |has|)
              (consp (qcdr cond)) (consp (qcddr cond))
              (eq (qcdr (qcddr cond)) nil))
              (hasCate (|subCopy| (qcadr cond) s0)
                        (|subCopy| (qcaddr cond) s0)
                        (copy s1)))))
             ((and (consp cond)
              (or (eq (qcdr cond) |and|) (eq (qcar cond) |and|)))
              (hasSigAnd (qcdr cond) s0 sl))
             ((and (consp cond)
defun hasAtt

The hasAtt function tests whether dom has attribute att under sl needs s0 similar to hasSig.

(defun hasAtt (dom att sl)
  (let ((|$domPvar| fun atts u x cond s)
    (declare (special |$domPvar|)))
    (cond
      ((setq fun (car dom))
       (cond
        (setq atts
          (|subCopy| (getdatabase fun 'attributes) (|constructSubst| dom)))
        (cond
          ((consp (setq u (|getInfovec| (car dom))))
            (do ((next atts (cdr next))
                 (endtest nil (null (eq s '|failed|))))
                ((or (atom next) endtest) nil)
            (setq x (caar next))
            (setq cond (cdar next))
            (setq s (|unifyStruct| x att (copy sl)))
            (cond
              ((and (null (atom cond)) (null (eq s '|failed|)))
                (setq s (|hasCatExpression| cond s))))))
          s)
        (t
          (t '|failed|)))))
      (t '|failed|))))

— defun hasAtt —
(do ((next atts (cdr next))
     (endtest nil (null (eq s 'failed)))))
  ((or (atom next) endtest) nil)
  (setq x (caar next))
  (setq cond (cadar next))
  (setq s (unifyStruct x att (copy sl)))
  (cond
    ((and (null (atom cond)) (null (eq s 'failed)))
     (setq s (hasCatExpression cond s))))
  (t 'failed))
(t 'failed)))

---

defun hasSigAnd

[hasCate p433]
[subCopy p??]
[keyedSystemError p??]

— defun hasSigAnd —

(defun hasSigAnd (andCls s0 sl)
  (let (sa dead)
    (setq sa 'failed)
    (loop for cls in andCls
do
      (when dead (return))
      (setq sa
        (cond
          ((atom cls) (copy sl))
          ((and (consp cls) (eq (qcar cls) 'has) (consp (qcdr cls))
             (consp (qcddr cls)) (eq (qcdddr cls) nil))
            (hasCate (subCopy (qcadr cls) s0)
             (subCopy (qcaddr cls) s0)
             (copy sl)))
          (t
            (keyedSystemError 'S2GE0016
              (list "hasSigAnd" "unexpected condition for signature"))))
      (when (eq sa 'failed) (setq dead t))
    )
    sa))

---
defun hasSigOr

(hasCate p433)
(hasSigAnd p425)
[keyedSystemError p??]

— defun hasSigOr —

(defun hasSigOr (orCls s0 sl)
  (let (sa found)
    (setq sa 'failed)
    (loop for cls in orCls
      until found
      do
        (setq sa
          (cond
            ((atom cls) (copy sl))
            ((and (consp cls) (eq (qcar cls) 'has) (consp (qcdr cls))
              (consp (qcdrr cls)) (eq (qcdddr cls) nil))
              (hasCate
                (subCopy (qcadr cls) s0)
                (subCopy (qcaddr cls) s0)
                (copy s1)))
            ((and (consp cls)
              (or (eq (qcar cls) 'and) (eq (qcar cls) 'and))
              (hasSigAnd (qcdr cls) s0 sl))
              t
              (keyedSystemError 'S2GE0016
                (list "hasSigOr" "unexpected condition for signature")))
            (t (unless (eq sa 'failed) (setq found t))))
        (setq sa found))
    sa))

— defun hasAttSig —

defun hasAttSig

The argument d is domain, x is a list of attributes and signatures. The result is an augmented
SL, if d has x, 'failed otherwise. [hasAtt p424]
(hasSig p423)
[keyedSystemError p??]

— defun hasAttSig —

(defun hasAttSig (d x sl)
  (loop for y in x
    until (eq sl 'failed)
    do
      (setq sl
(cond
  ((and (consp y) (eq (qcar y) 'attribute)
        (consp (qcdr y)) (eq (qcddr y) nil))
    (hasAtt d (qcadr y) sl))
  ((and (consp y) (eq (qcar y) 'signature)
        (consp (qcdr y)) (consp (qcddr y)) (eq (qcdddr y) nil))
    (hasSig d (qcadr y) (qcaddr y) sl))
  (t
    (keyedSystemError 'S2GE0016
      (list "hasAttSig" "unexpected form of unnamed category"))))))

---

defun hasCate1

[hasCate p433]
[$domPvar p47]

— defun hasCate1 —

(defun hasCate1 (dom cat sl domPvar)
  (let ($domPvar)
    (declare (special $domPvar)))
  (setq $domPvar domPvar)
  (hasCate dom cat sl))

---

defun hasCatExpression

[hasCatExpression p427]
[hasCate p433]
[keyedSystemError p??]

— defun hasCatExpression —

(defun hasCatExpression (cond sl)
  (let (y)
    (cond
      ((and (consp cond) (eq (qcar cond) 'or))
        (when
          (let (result)
            (loop for x in (qcdr cond)
              do (setq result

          ))
          nil)
          (hasCatExpression cond sl))
      (t
        (hasCatExpression cond sl))))
  )
)
(or result
  (not (eq (setq y (|hasCatExpression| x sl)) '|failed|))))

((and (consp cond) (eq (qcar cond) 'and))
  (when
    (let ((result t))
      (loop for x in (qcdr cond)
        do (setq result
          (and result
            (not (eq (setq sl (|hasCatExpression| x sl)) '|failed|))))
          result)
          sl))
  ((and (consp cond) (eq (qcar cond) '|has|)
        (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
    (|hasCate| (qcadr cond) (qcaddr cond) sl))
  (t
    (|keyedSystemError| 'S2GE0016
      (list "hasSig" "unexpected condition for attribute"))))

defun unifyStruct

[isPatternVar p431]
[unifyStructVar p429]
[unifyStruct p428]

— defun unifyStruct —

(defun |unifyStruct| (s1 s2 sl)
  (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
  (cond
    ((equal s1 s2) sl)
    (t
      (when (and (consp s1) (eq (qcar s1) '|:|))
        (setq s1 (qcadr s1)))
      (when (and (consp s2) (eq (qcar s2) '|:|))
        (setq s2 (qcadr s2)))
      (when (null (atom s1)) (eq (car s1) '|#|))
        (setq s1 (length (cadr s1))))
      (when (null (atom s2)) (eq (car s2) '|#|))
        (setq s2 (length (cadr s2))))
    (cond
      ((equal s1 s2) sl)
((isPatternVar s1) (unifyStructVar s1 s2 sl))
((isPatternVar s2) (unifyStructVar s2 s1 sl))
((or (atom s1) (atom s2)) 'failed)
(t
  (loop until (or (null s1) (null s2) (eq sl 'failed)))
    do
      (setq sl (unifyStruct (car s1) (car s2) sl))
      (setq s1 (cdr s1))
      (setq s2 (cdr s2)))
  (if (or s1 s2) 'failed sl))))

---

defun unifyStructVar

The first argument is a pattern variable, which is not substituted by sl [contained p??]

[lassoc p??]
[unifyStruct p428]
[constructor? p??]
[subCopy p??]
[containsVars p430]
[canCoerce p??]
[resolveTT p??]
[isPatternVar p431]
[augmentSub p??]
[$domPvar p47]
[$Coerce p??]
[$Subst p??]
[$hope p??]

--- defun unifyStructVar ---

(defun unifyStructVar (v ss sl)
  (let (ps s1 s0 ns0 ns1 s3)
    (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
    (cond
      ((contained v ss) 'failed))
    (t
      (setq ps (lassoc ss sl))
      (setq s1 (if ps ss))
      (cond
        ((or (setq s0 (lassoc v sl)) (setq s0 (lassoc v |$Subst|)))
         (setq s (unifyStruct s0 s1 (copy sl)))
        (cond
          ((eq s 'failed))
          (cond
            ((isPatternVar s1) (unifyStructVar s1 s2 sl))
            ((isPatternVar s2) (unifyStructVar s2 s1 sl))
            ((or (atom s1) (atom s2)) 'failed)
            (t
              (loop until (or (null s1) (null s2) (eq sl 'failed)))
                do
                  (setq sl (unifyStruct (car s1) (car s2) sl))
                  (setq s1 (cdr s1))
                  (setq s2 (cdr s2)))
              (if (or s1 s2) 'failed sl))))))
  )

--- defun unifyStructVar ---
CHAPTER 17. FUNCTION SELECTION

(defun containsVars
  (The function containsVars tests whether term t contains a * variable. [isPatternVar p431] [containsVars1 p431]
— defun containsVars —

(defun containsVars (arg)
  (if (atom arg)
      (isPatternVar arg)
      (containsVars1 arg)))

defun isPatternVar

— defun isPatternVar —

(defun isPatternVar (v)
  (and (identp v)
       (member v	dolist
                        ((** *1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *20))
       t))

defun containsVars1

The function containsVars1 tests whether term t contains a * variable. This is a recursive version, which works on a list. [isPatternVar p431][containsVars1 p431]

— defun containsVars1 —

(defun containsVars1 (arg)
  (let ((t1 (car arg)) (t2 (cdr arg))
        (atom t1)
        (or (isPatternVar t1)
            (if (atom t2) (isPatternVar t2) (containsVars1 t2)))
        (or (containsVars1 t1)
            (if (atom t2) (isPatternVar t2) (containsVars1 t2))))))
defun hasCaty1

The cond is either a (has a b) or an OR clause of such conditions. SL is augmented, if cond is true, otherwise the result is 'failed

(defun hasCaty1 (cond sl)
  (let ([$domPvar] a s)
    (declare (special [$domPvar]))
    (setq [$domPvar] nil)
    (cond
      ((and (consp cond) (eq (qcar cond) 'has))
        (cond
          ((and (consp (qcdr cond)) (eq (qcddr cond) nil))
            (hasCate (qcadr cond) (qcaddr cond) sl))
          ((and (consp cond) (eq (qcdr cond) nil))
            (hasCate a (qcaddr x) sl))
          (t (hasCaty1 x sl))))
      ((and (consp cond) (eq (qcdr cond) 'and))
        (loop for x in (qcdr cond)
          while (not (eq s 'failed!))
          do
            (setq s
              (cond
                ((and (consp x) (eq (qcar x) 'has))
                  (hasCate (qcadr x) (qcaddr x) nil))
                ((and (consp x) (eq (qcdr x) nil))
                  (hasCate a (qcaddr x) s))
                (t (hasCaty1 x s)))))
      ((and (consp cond) (eq (qcdr cond) 'or))
        (do ((next (qcdr cond) (cdr next)) (x nil)
          (nextitem nil (null (eq s 'failed!)))))
          ((or (atom next)
            (progn (setq x (car next)) nil)
            nextitem)
            nil)
        (setq s
          (cond
            ((and (consp x) (eq (qcar x) 'has))
              (hasCate (qcadr x) (qcaddr x) (copy sl)))
            ((and (consp x) (eq (qcdr x) nil))
              (hasCate (qcadr x) (copy sl)))
            (eq (qcaar x) 'has) (hasCate (qcdr x) (copy sl)))
          (t (hasCaty1 x s))))
    s))
  (declaim (special [keyedSystemError]))
  [keyedSystemError p??])
  [$domPvar p47]
(\{\text{hasCate}\} \ (\text{qcadar} \, x) \ (\text{qcaddar} \, x) \ (\text{copy} \, \text{sl}))
(t \ (\text{hasCaty1} \ x \ (\text{copy} \, \text{sl})))))

s)
(t
(\text{keyedSystemError} \ 'S2GE0016
 (\text{list} \ "\text{hasCaty1}" \ "\text{unexpected condition from category table}")))))))

defun mkDomPvar

[\text{domArg} \, p422]
[\text{length} \, p??]
[\text{$\text{FormalMapVariableList}$} \, p??]

— defun mkDomPvar —

(defun \text{mkDomPvar} \, p \, d \, \text{subs} \, y)
(let (l)
 (\text{declare} \ (\text{special} \ \text{$\text{FormalMapVariableList}$}))
 (if (setq l (member p \text{$\text{FormalMapVariableList}$}))
 (\text{domArg} \, d \ (- \ (\text{\|} \ \text{$\text{FormalMapVariableList}$}) \ (\text{\|} \, l) \, \text{subs} \, y)
 d)))

defun hasCate

[\text{isPatternVar} \, p431]
[\text{hasCate1} \, p427]
[\text{hasCateSpecial} \, p434]
[\text{containsVariables} \, p??]
[\text{subCopy} \, p??]
[\text{hasCaty} \, p420]
[\text{$\text{EmptyMode}$} \, p??]
[\text{$\text{Subst}$} \, p??]
[\text{$\text{hope}$} \, p??]

— defun hasCate —

(defun \text{hasCate} \, (\text{dom} \, \text{cat} \, \text{sl})
(let (\text{ns1} \, p \, s \, \text{sl1})
(\text{declare} \ (\text{special} \ \text{\text{\|$\text{hope}$}} \ \text{\text{\|\text{Subst}}}$ \ \text{\text{\|\text{EmptyMode}}}))))
(\text{cond}
((equal dom #$EmptyMode#) nil)
((isPatternVar dom)
 (cond
   ((and (setq p (assq dom sl))
       (not (eq (setq nsl (hasCate (cdr p) cat sl)) 'failed)))
    nsl)
   ((or (setq p (assq dom #$Subst#)) (setq p (assq dom sl)))
    (setq s (hasCate1 (cdr p) cat sl dom))
    (cond
      ((null (eq s 'failed)) s)
      (t (hasCateSpecial dom (cdr p) cat sl)))
    t)
   (when (not (eq sl 'failed)) (setq #$hope# t))
   'failed)))
(t
 (setq sl1
   (loop for item in sl
     when (null (containsVariables (cdr item)))
     collect item))
 (when sl1 (setq cat (subCopy cat sl1)))
 (hasCaty dom cat sl))))

---

defun constructSubst

[internl p??]
[stringimage p??]

— defun constructSubst —

(defun |constructSubst| (d)
 (let (sl (i 0))
   (setq sl (list (cons #$ d))))
 (when (listp d)
   (dolist (x (cdr d))
     (setq i (1+ i))
     (setq sl (cons (cons (internl "$" (stringimage i)) x) sl))))
 sl))

---

defun hasCateSpecial

The variable v is a pattern variable, dom is its binding under #$Subst#. We try to change dom so that it has category cat under sl. The result is a substitution list or 'failed. [eqcar p??]
defun hasCateSpecial (v dom cat sl)
(let (arg d domp nsl)
(declare (special $Integer | $QuotientField))
(cond
  ((and (cons dom) (eq (qcar dom) '|FactoredForm|)
    (cons (qcdr dom)) (eq (qcddr dom) nil))
   (setq arg (qcadr dom))
   (when (isSubDomain arg $Integer) (setq arg $Integer))
   (setq d (list '|FactoredRing| arg))
   (setq sl (hasCate arg '|Ring| (augmentSub v d sl)))
   (if (eq sl '|failed|) '
   'failed)
   (hasCaty d cat sl)))
  ((or (eqcar cat '|Field|) (eqcar cat '|DivisionRing|))
   (when (isSubDomain dom $Integer) (setq dom $Integer))
   (setq d (list |$QuotientField| dom))
   (hasCaty dom '|IntegralDomain| (augmentSub v d sl)))
  ((and (cons cat) (eq (qcar cat) '|PolynomialCategory|)
    (cons (qcdr cat)))
   (setq domp (cons '|Polynomial| (list (qcadr cat))))
   (and (or (containsVars (qcadr cat)) (canCoerceFrom dom domp))
    (hasCaty domp cat (augmentSub v domp sl))))
  ((isSubDomain dom $Integer))
   (setq nsl (hasCate $Integer cat (augmentSub v $Integer sl)))
   (if (eq nsl '|failed|)
    (hasCateSpecialNew v dom cat sl)
    (hasCaty $Integer cat nsl))
  (t
   (hasCateSpecialNew v dom cat sl))))

— defun hasCateSpecial —

defun hasCateSpecialNew

(member p 1048)
(eqcar p ??)
[defaultTargetFE p 437]
[isEqualOrSubDomain p 438]
[underDomainOf p ??]
[hasCaty p 420]
[$Integer p ??]
[$ComplexInteger p ??]
[$RationalNumber p ??]

— defun hasCateSpecialNew —

(defun |hasCateSpecialNew| (v dom cat sl)
(let (fe alg fefull d partialResult)
(declare (special |$RationalNumber| |$ComplexInteger| |$Integer|)))
(setq fe
  (member p (qcar cat)
    '((|ElementaryFunctionCategory|
      |TrigonometricFunctionCategory|
      |ArcTrigonometricFunctionCategory|
      |HyperbolicFunctionCategory|
      |ArcHyperbolicFunctionCategory|
      |PrimitiveFunctionCategory|
      |SpecialFunctionCategory|
      |Evalable|
      |CombinatorialOpsCategory|
      |TranscendentalFunctionCategory|
      |AlgebraicallyClosedFunctionSpace|
      |ExpressionSpace|
      |LiouvillianFunctionCategory|
      |FunctionSpace|))
(setq alg
  (member p (qcar cat)
    '((|RadicalCategory|
      |AlgebraicallyClosedField|))
(setq fefull
  (or fe alg (eqcar cat '|CombinatorialFunctionCategory|)))
(setq partialResult
  (cond
    ((or (eqcar dom '|Variable|) (eqcar dom '|Symbol|))
      (cond
        ((member p (car cat)
          '(|SemiGroup|
            |AbelianSemiGroup|
            |Monoid|
            |AbelianGroup|)))
        (??)))
    (??)))
    (??)
(setq d (list '|Polynomial| |$Integer|))
(|augmentSub| v d sl))
((eqcar cat '|Group|)
 (setq d (list '|Fraction| (list '|Polynomial| |$Integer|)))
 (|augmentSub| v d sl))
(ffull
 (setq d (|defaultTargetFE| dom))
 (|augmentSub| v d sl))
(t '|failed|)))
(((isEqualOrSubDomain| dom |$Integer|)
 (cond
 (fe
 (setq d (|defaultTargetFE| |$Integer|))
 (|augmentSub| v d sl))
 (alg
 (setq d '(|AlgebraicNumber|))
 (|augmentSub| v d sl))
 (t '|failed|)))
((equal (|underDomainOf| dom) |$ComplexInteger|)
 (setq d (|defaultTargetFE| |$ComplexInteger|))
 (|hasCaty| d cat (|augmentSub| v d sl)))
((and (equal dom |$RationalNumber|) alg)
 (setq d '(|AlgebraicNumber|))
 (|augmentSub| v d sl))
(ffull
 (setq d (|defaultTargetFE| dom))
 (|augmentSub| v d sl))
(t '|failed|)))
(if (eq partialResult '|failed|)
 '|failed|
 (|hasCaty| d cat partialResult)))

---

(defun defaultTargetFE

[typeIsASmallInteger p??]
 [isEqualOrSubDomain p438]
 [ifcar p??]
 [defaultTargetFE p437]
 [$FunctionalExpression p??]
 [$Integer p??]
 [$Symbol p??]
CHAPTER 17. FUNCTION SELECTION

--- defun defaultTargetFE ---

(defun defaultTargetFE (dom)
  (let (a options)
    (declare (special FunctionalExpression Integer Symbol RationalNumber))
    (setq a (car dom))
    (setq options (cdr dom))
    (cond
      ((or (and (consp a) (eq (qcar a) 'Variable)
                (consp (qcdr a)) (eq (qcddr a) nil))
        (equal a RationalNumber)
        (member (qcar a) (list (qcar Symbol) 'RationalRadicals 'Pi))
        (typeIsASmallInteger a)
        (isEqualOrSubDomain a Integer)
        (equal a '(AlgebraicNumber)))
      (if (ifcar options)
          (list FunctionalExpression (list 'Complex Integer))
        (list FunctionalExpression Integer))
      ((and (consp a) (eq (qcar a) 'Complex))
        (cons (qcdr a) (eq (qcddr a) nil))
        (defaultTargetFE (qcadr a) t))
      ((and (consp a) (consp (qcdr a)) (eq (qcddr a) nil)
        (member (qcar a) '(Polynomial RationalFunction Fraction)))
        (defaultTargetFE (qcadr a) (ifcar options))
      ((and (consp a) (equal (qcar a) FunctionalExpression))
        (cons (qcdr a) (eq (qcddr a) nil))
        a)
      ((ifcar options)
        (list FunctionalExpression (list 'Complex a)))
      (t
        (list FunctionalExpression a))))

---

defun isEqualOrSubDomain

[isEqualOrSubDomain p??]

--- defun isEqualOrSubDomain ---

(defun isEqualOrSubDomain (d1 d2)
  (or (equal d1 d2)
      (isEqualOrSubDomain d1 d2)
      (and (atom d1)))
(or (and (consp d2) (eq (qcar d2) '|Variable|)
  (consp (qcdr d2)) (eq (qcddr d2) nil)
  (equal (qcadr d2) d1))
 (and (consp d2) (eq (qcdr d2) nil)
  (equal (qcar d2) d1)))))

(and (atom d2)
 (or (and (consp d1) (eq (qcar d1) '|Variable|)
  (consp (qcdr d1)) (eq (qcddr d1) nil)
  (equal (qcadr d1) d2))
 (and (consp d1) (eq (qcdr d1) nil)
  (equal (qcar d1) d2)))))
Chapter 18

System Command Handling

The system commands are the top-level commands available in Axiom that can all be invoked
by prefixing the symbol with a closed-paren. Thus, to see they copyright you type:

)`copyright`

New commands need to be added to this table. The command invoked will be the first entry
of the pair and the “user level” of the command will be the second entry.

See:

- The “abbreviations” (19.2 p 483) command
- The “boot” (4.1 p 23) command
- The “browse” (?? p ??) command
- The “cd” (?? p ??) command
- The “clear” (23.3 p 499) command
- The “close” (24.2 p 510) command
- The “compile” (?? p ??) command
- The “copyright” (26.2 p 522) command
- The “credits” (27.3 p 525) command
- The “display” (29.2 p 535) command
- The “edit” (30.2 p 544) command
- The “fin” (31.1 p 548) command
The "frame" (32.5 p 565) command
The "help" (33.2 p 572) command
The "history" (34.4 p 582) command
The "lisp" (?? p ??) command
The "library" (66.1 p 1013) command
The "load" (38.1 p 629) command
The "ltrace" (39.1 p 632) command
The "pquit" (40.2 p 634) command
The "quit" (41.2 p 638) command
The "read" (42.1 p 642) command
The "regress" (?? p ??) command
The "savesystem" (44.1 p 650) command
The "set" (45.36 p 808) command
The "show" (46.1 p 814) command
The "spool" (?? p ??) command
The "summary" (48.1 p 830) command
The "synonym" (49.1 p 832) command
The "system" (?? p ??) command
The "tangle" (?? p ??) command
The "trace" (52.1 p 847) command
The "trademark" (26.2 p 523) command
The "undo" (53.4 p 922) command
The "what" (54.1 p 939) command
The "with" (55.1 p 947) command
The "workfiles" (56.1 p 949) command
The "zsystemdevelopment" (57.1 p 953) command
18.1 Variables Used

defvar $systemCommands

— initvars —

(defvar |$systemCommands| nil)

— postvars —

(eval-when (eval load)
  (setq |$systemCommands|
    `( |
      |abbreviations| . |compiler|  
      |boot| . |development|  
      |browse| . |development|  
      |cd| . |interpreter|  
      |clear| . |interpreter|  
      |close| . |interpreter|  
      |compiler| . |compiler|  
      |copyright| . |interpreter|  
      |credits| . |interpreter|  
      |describe| . |interpreter|  
      |display| . |interpreter|  
      |edit| . |interpreter|  
      |fin| . |development|  
      |frame| . |interpreter|  
      |help| . |interpreter|  
      |history| . |interpreter|  
      |lisp| . |development|  
      |library| . |interpreter|  
      |load| . |interpreter|  
      |ltrace| . |interpreter|  
      |pquit| . |interpreter|  
      |quit| . |interpreter|  
      |read| . |interpreter|  
      |regress| . |interpreter|  
      |savesystem| . |interpreter|  
      |set| . |interpreter|  
      |show| . |interpreter|  
      |spool| . |interpreter|  
      |summary| . |interpreter|  
      |synonym| . |interpreter|  
      |system| . |interpreter|)  
  )
defvar $syscommands

This table is used to look up a symbol to see if it might be a command.

— initvars —

(defvar $syscommands nil)

— postvars —

(eval-when (eval load)
  (setq $syscommands (mapcar #'car $systemCommands)))

defvar $noParseCommands

This is a list of the commands which have their arguments passed verbatim. Certain functions, such as the lisp function need to be able to handle all kinds of input that will not be acceptable to the interpreter.

— initvars —

(defvar $noParseCommands nil)

— postvars —
18.2 Functions

(defun handleNoParseCommands

The system commands given by the global variable $noParseCommands$ require essentially no preprocessing/parsing of their arguments. Here we dispatch the functions which implement these commands.

There are four standard commands which receive arguments

- boot
- lisp
- synonym
- system

There are six standard commands which do not receive arguments –

- quit
- fin
- pquit
- credits
- copyright
- trademark

As these commands do not necessarily exhaust those mentioned in $noParseCommands$, we provide a generic dispatch based on two conventions: commands which do not require an argument name themselves, those which do have their names prefixed by “np”. This makes it possible to dynamically define new system commands provided you handle the argument parsing.
defun Handle a top level command

(defun doSystemCommand (string)
  (let (line tok unab optionList)
    (declare (special line \$tokenCommands\ \$noParseCommands\))
    (setq string (concat "\" (expand-tabs string)))
    (setq line string)
    (processSynonyms)
    (setq string line)
    (setq string (substring string 1 nil))
    (cond
      ((string= string "") nil)
      (t (setq tok (getFirstWord string))
        (cond
          (tok
            (setq unab (unAbbreviateKeyword tok))
            (cond
              ((member unab \$noParseCommands\) (handleNoParseCommands unab string))
              (t
                (setq optionList (splitIntoOptionBlocks string))
                (cond
                  ((member unab \$tokenCommands\) (handleTokensizeSystemCommands unab optionList))
                  (t
                    (handleParsedSystemCommands unab optionList) nil)))))))
    (t nil))))

— defun doSystemCommand —
18.2. FUNCTIONS

defun Split block into option block

(stripSpaces p471)

— defun splitIntoOptionBlocks —

(defun |splitIntoOptionBlocks| (str)
(let (inString block (blockStart 0) (parenCount 0) blockList)
(dotimes (i (1- (#| str|)))
  (cond
    ((char= (elt str i) "") (setq inString (null inString)))
    (t
      (when (and (char= (elt str i) ")") (null inString))
        (incf parenCount))
      (when (and (char= (elt str i) ")") (null inString))
        (decf parenCount))
      (when
        (and (char= (elt str i) ")")
          (null inString)
          (= parenCount -1))
        (setq block (|stripSpaces| (subseq str blockStart i))))
      (setq blockList (cons block blockList))
      (setq blockStart (1+ i))
      (setq parenCount 0))))
  (setq blockList (cons (|stripSpaces| (subseq str blockStart)) blockList))
  (nreverse blockList)))

defun Tokenize a system command

dumbTokenize p467
tokTran p467
systemCommand p448

— defun handleTokenizeSystemCommands —

(defun |handleTokenizeSystemCommands| (unabr optionList)
(declare (ignore unabr))
(let (parcmd)
  (setq optionList (mapcar #'(lambda (x) (|dumbTokenize| x)) optionList))
  (setq parcmd
    (mapcar #'(lambda (opt) (mapcar #'(lambda (tok) (|tokTran| tok)) opt))
      optionList))
  (when parcmd (|systemCommand| parcmd))))

— — —
defun Handle system commands

You can type "?)?" and see trivial help information. You can type "?)? compile" and see compiler related information [selectOptionLC p479]

[helpSpad2Cmd p572]
[selectOption p479]
[commandsForUserLevel p448]
[options p??]
[e p??]
[$systemCommands p443]
[$syscommands p444]
[$CategoryFrame p??]

--- defun systemCommand ---

(defun systemCommand (cmd)
  (let ((options e op argl options fun)
        (declare (special options e $systemCommands syscommands $CategoryFrame))
    (setq op (caar cmd))
    (setq argl (cdar cmd))
    (setq options (cdr cmd))
    (setq options options)
    (setq e $CategoryFrame)
    (setq fun (selectOptionLC op syscommands 'commandError))
    (if (and argl (eq (elt argl 0) '?) (not (eq fun 'synonym)))
        (helpSpad2Cmd (cons fun nil))
      (progn
        (setq fun
          (selectOption fun (commandsForUserLevel $systemCommands)
            'commandUserLevelError))
        (funcall fun argl))))

---

defun Select commands matching this user level

The $UserLevel contains one of three values: compiler, development, or interpreter. This variable is used to select a subset of commands from the list stored in $systemCommands, representing all of the commands that are valid for this level. [satisfiesUserLevel p451]

--- defun commandsForUserLevel ---

(defun commandsForUserLevel (arg)
  (let (c)
    (dolist (pair arg)
(when (|satisfiesUserLevel| (cdr pair))
  (setq c (cons (car pair) c)))
(nreverse c)))

---

(defun No command begins with this string

[commandErrorMessage p449]

— defun commandError —

(defun |commandError| (x u)
  (|commandErrorMessage| '|command| x u))

---

defun No option begins with this string

[commandErrorMessage p449]

— defun optionError —

(defun |optionError| (x u)
  (|commandErrorMessage| '|option| x u))

---

defvar $oldline

— initvars —

(defvar $oldline nil "used to output command lines")

---

defun No command/option begins with this string

[commandAmbiguityError p452]
[sayKeyedMsg p329]
— defun commandErrorMessage —

(defun commandErrorMessage (kind x u)
  (declare (special $oldline line))
  (setq $oldline line)
  (if u
    (commandAmbiguityError kind x u)
    (progn
      (sayKeyedMsg 'S2IZ0008 (list kind x))
      (terminateSystemCommand))))

— defun Option not available at this user level —

(defun optionUserLevelError (x u)
  (userLevelErrorMessage 'option x u))

— defun Command not available at this user level —

(defun commandUserLevelError (x u)
  (userLevelErrorMessage 'command x u))

— defun Command not available error message —

(defun commandAmbiguityError (kind x u)
  (sayKeyedMsg 'S2IZ0008 (list kind x)))
defun userLevelErrorMessage

(defun userLevelErrorMessage (kind x u)
  (declare (special $UserLevel)))
  (if u
    (commandAmbiguityError kind x u)
    (progn
      (sayKeyedMsg 'S2IZ0007 (list $UserLevel kind))
      (terminateSystemCommand)))))

defun satisfiesUserLevel

(defun satisfiesUserLevel (x)
  (declare (special $UserLevel))
  (cond
    ((eq x 'interpreter) t)
    ((eq $UserLevel 'interpreter) nil)
    ((eq x 'compiler) t)
    ((eq $UserLevel 'compiler) nil)
    (t t)))

defun hasOption

(defun hasOption (al opt)
  (let ((optPname (pname opt)) found)
    (loop for pair in al do
      (when (stringPrefix? (pname (car pair)) optPname) (setq found pair))
    until found)
  found))
defun terminateSystemCommand

(tersyscommand p452)

— defun terminateSystemCommand —

(defun |terminateSystemCommand| nil (tersyscommand))

—

defun Terminate a system command

[spadThrow p??]

— defun tersyscommand —

(defun tersyscommand ()
  (fresh-line)
  (setq chr 'endoflinechr)
  (setq tok 'end_unit)
  (|spadThrow|))

—

defun commandAmbiguityError

[sayKeyedMsg p329]
[sayMSG p331]
[bright p??]
[terminateSystemCommand p452]

— defun commandAmbiguityError —

(defun |commandAmbiguityError| (kind x u)
  (|sayKeyedMsg| 's2iz0009 (list kind x))
  (dolist (a u) (|sayMSG| (cons " " (|bright| a))))
  (|terminateSystemCommand|))

—
18.2. FUNCTIONS

defun getParserMacroNames

The $pfMacros$ is a list of all of the user-defined macros. [$pfMacros$ p97]

— defun getParserMacroNames 0 —

(defun getParserMacroNames ()
 (declare (special $pfMacros$))
 (remove-duplicates (mapcar #'car $pfMacros$)))

defun clearParserMacro

Note that if a macro is defined twice this will clear the last instance. Thus:

| a => 3 |
| a => 4 |
| d macros |
| a => 4 |
| clear prop a |
| d macros |
| a => 3 |
| clear prop a |
| d macros |
| nil |

[ifcdr p??]
[assoc p??]
[remalist p??]
[$pfMacros$ p97]

— defun clearParserMacro —

(defun clearParserMacro (macro)
 (declare (special $pfMacros$))
 (when (ifcdr (assoc macro $pfMacros$))
  (setq $pfMacros$ (remalist $pfMacros$ macro))))

defun displayMacro

[isInterpMacro p??]
[sayBrightly p??]
--- defun displayMacro ---

(defun |displayMacro| (name)
  (let (|$op| m body args)
    (declare (special |$op|))
    (setq m (|isInterpMacro| name))
    (cond
      ((null m)
       (|sayBrightly|
        (cons " " (append (|bright| name)
                        (cons "is not an interpreter macro." nil))))))
    (t
     (setq |$op| (strconc "macro " (|object2String| name)))
     (setq args (car m))
     (setq body (cdr m))
     (setq args
       (cond
        ((null args) nil)
        ((null (cdr args)) (car args))
        (t (cons '|Tuple| args))))
     (|mathprint| (cons 'map (cons (cons args body) nil))))))

---

--- defun displayWorkspaceNames ---

(defun |displayWorkspaceNames| ()
  (let (pmacs names imacs)
    (setq imacs (|getInterpMacroNames|)))
18.2. FUNCTIONS

(defun getWorkspaceNames ()
  ;; getWorkspaceNames() ==
  ;;  NMSORT [n for [n,:.] in CAAR $InteractiveFrame |
  ;;  (n ^= "--macros--" and n ^= "--flags--")]

  (setq pmacs (|getParserMacroNames|))
  (|sayMessage| "Names of User-Defined Objects in the Workspace:")
  (setq names (msort (append (|getWorkspaceNames|) pmacs)))
  (if names
    (|sayAsManyPerLineAsPossible| (mapcar #'|object2String| names))
    (|sayBrightly| " * None *"))
  (setq imacs (setdifference imacs pmacs))
  (when imacs
    (|sayMessage| "Names of System-Defined Objects in the Workspace:")
    (|sayAsManyPerLineAsPossible| (mapcar #'|object2String| imacs))))
((AND (not (eq n '|--macros--|))
 (not (eq n '|--flags--|)))
 (SETQ G166322
 (CONS n G166322))))))))))

---

defun fixObjectForPrinting

The $msgdbPrims variable is set to:

(\%b \%d \%l \%i \%u \%n \%x \%ce \%rj
 \%U \%b \%d \%l \%i \%u \%n \%x \%ce \%rj)

| object2Identifier p?? |
| [member p1048] |
| [strconc p??] |
| [pname p1045] |
| [$msgdbPrims p327] |

--- defun fixObjectForPrinting ---

(defun fixObjectForPrinting (v)
 (let (vp)
   (declare (special $msgdbPrims))
   (setq vp (object2Identifier v))
   (cond
     ((eq vp '%) "\\%")
     ((member vp $msgdbPrims) (strconc "\\" (pname vp)))
     (t v))))

---

defun displayProperties, sayFunctionDeps

;displayProperties(option,1) ==
 ; $dependentAlist : local := nil
 ; $dependeeAlist : local := nil
 ; [opt,:vl]:= (1 or ['properties])
 ; imacs := getInterpMacroNames()
 ; pmacs := getParserMacroNames()
 ; macros := REMDUP append(imacs, pmacs)
 ; if vl is ['all] or null vl then
 ;   vl := MSORT append(getWorkspaceNames(),macros)
 ; if $frameMessages then sayKeyedMsg("S2IZ0065","$interpreterFrameName")
null vl =>
null $frameMessages => sayKeyedMsg("S2IZ0066",NIL)
sayKeyedMsg("S2IZ0067",[$interpreterFrameName])
interpFunctionDepAlists()
for v in vl repeat
isInternalMapName(v) => 'iterate
pl := getIProplist(v)
option = 'flags => getAndSay(v,"flags")
option = 'value => displayValue(v,getI(v,'value),nil)
option = 'condition => displayCondition(v,getI(v,"condition"),nil)
option = 'mode => displayMode(v,getI(v,'mode),nil)
option = 'type => displayType(v,getI(v,'value),nil)
option = 'properties =>
v = "--flags--" => nil
pl is [ ['cacheInfo;::;:],::;] => nil
v1 := fixObjectForPrinting(v)
sayMSG ['"Properties of",:bright prefix2String v1,":"']
null pl =>
v in pmacs =>
sayMSG " This is a user-defined macro."
displayParserMacro v
isInterpMacro v =>
sayMSG " This is a system-defined macro."
displayMacro v
sayMSG " none"
propsSeen:=nil
for [prop,:val] in pl | ^MEMQ(prop,propsSeen) and val repeat
  prop in '(alias generatedCode IS_-GENSYM mapBody localVars) =>
    nil
  prop = 'condition =>
    displayCondition(prop,val,true)
  prop = 'recursive =>
    sayMSG " This is recursive."
  prop = 'isInterpreterFunction =>
    sayMSG " This is an interpreter function."
sayFunctionDeps v where
sayFunctionDeps x ==
  if dependents := GETALIST($dependentAlist,x) then
    null rest dependents =>
    sayMSG ['" The following function or rule ",
      "depends on this:",:bright first dependents]
sayMSG
    " The following functions or rules depend on this:"
msg := ["%b"," "]
for y in dependents repeat msg := [" ",y,:msg]
sayMSG [:nreverse msg,"%d"]
if dependees := GETALIST($dependeeAlist,x) then
  null rest dependees =>
  sayMSG ['" This depends on the following function ",
    "or rule:",:bright first dependees]
CHAPTER 18. SYSTEM COMMAND HANDLING

sayMSG
" This depends on the following functions or rules:"
msg := ['"%b",""
for y in dependees repeat msg := ['" ",y,:msg]
sayMSG [:nreverse msg,'%d']
prop = 'isInterpreterRule =>
sayMSG " This is an interpreter rule."
sayFunctionDeps v
prop = 'localModemap =>
displayModemap(v,val,true)
prop = 'mode =>
displayMode(prop,val,true)
prop = 'value =>
val => displayValue(v,val,true)
sayMSG ['" 
\prop,'": 
\val]
propsSeen:= [\prop,:propsSeen]
sayKeyedMsg("S2IZ0068",[\option])
terminateSystemCommand()

[seq p??]
[getalist p??]
[exit p??]
sayMSG p331
[bright p??]
$dependeeAlist p??]
$dependentAlist p??]

— defun displayProperties,sayFunctionDeps —

(defun |displayProperties,sayFunctionDeps| (x)
(prog (dependents dependees msg)
(declare (special |$dependeeAlist| |$dependentAlist|))
(return
(seq
(if (setq dependents (getalist |$dependentAlist| x))
(seq
(if (null (cdr dependents))
(exit)

|sayMSG| (cons " The following function or rule "
 (cons "depends on this:" (|bright| (car dependents)))))
|sayMSG| " The following functions or rules depend on this:"
(setq msg (cons '\|%b| (cons " " nil)))
(do ((G166397 dependents (cdr G166397)) (y nil))
 ((or (atom G166397) (progn (setq y (car G166397)) nil)) nil)
 (seq (exit (setq msg (cons " " (cons y msg)))))
(exist (|sayMSG| (append |nreverse msg| (cons '\|%d| nil))))
nil)
(exit
(if (setq dependees (getalist |$dependeeAlist| x))

[seq p??]
[getalist p??]
[exit p??]
sayMSG p331
[bright p??]
$dependeeAlist p??]
$dependentAlist p??]
(seq
  (if (null (cdr dependees))
    (exit
      (|sayMSG| (cons " This depends on the following function "
        (cons "or rule:"  (|bright| (car dependees))))))
    (|sayMSG| " This depends on the following functions or rules:")
    (setq msg (cons '|%b| (cons " " nil)))
    (do ((G166406 dependees (cdr G166406)) (y nil))
      ((or (atom G166406) (progn (setq y (car G166406)) nil)) nil)
      (seq (exit (setq msg (cons " " (cons y msg)))))
    (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil))))))))

defun displayValue

[sayMSG p331]
[fixObjectForPrinting p456]
[pname p1045]
[objValUnwrap p??]
[objMode p??]
[displayRule p??]
[strconc p??]
[prefix2String p??]
[objMode p??]
[getdatabase p1010]
[concat p1047]
[form2String p??]
[mathprint p??]
[outputFormat p??]
[objMode p??]
[$op p??]
[$EmptyMode p??]

— defun displayValue —

(defun |displayValue| (|$op| u omitVariableNameIfTrue)
  (declare (special|$op|))
  (let (expr op rhs label labmode)
    (declare (special|$EmptyMode|))
    (if (null u)
      (|sayMSG|  
        (list '| Value of | (|fixObjectForPrinting| (pname|$op|)) ": (none)"))
      (progn
        (setq expr (|objValUnwrap| u)))
      (declare (special|$EmptyMode|))
      (let (x)
        (setq x (|getdatabase|))
        (while (not (null x))
          (setq x (cdr x))
          (setq labmode (car x)))
        (setq labmode (ev (list|$EmptyMode| labmode)))
        (setq expr (ev (list|$op| labmode)))))
    (setq expr (ev expr))
    (let (y)
      (setq y (|concat| (concat (|prefix2String| expr) (|mathprint|)))))
    (setq msg (append (nreverse msg) (cons '|%d| nil))))
    (|sayMSG| (append msg y))))

(setq msg (cons " " nil))
(while (not (null msg))
  (setq msg (append msg (cons '|%d| nil))))
(append msg (cons '|%d| nil)))

(append msg (cons '|%d| nil))

|defun displayValue|
(if (or (and (consp expr) (progn (setq op (qcar expr)) t) (eq op 'map))
   (equal (|objMode| u) |$EmptyMode|))
  (|displayRule| |$op| expr)
  (progn
   (cond
    (omitVariableNameIfTrue
     (setq rhs "): ")
    (setq label "Value (has type ")
    (t
     (setq rhs ": ")
     (setq label (strconc "Value of " (pname |$op|) ": "))))
   (setq labmode (|prefix2String| (|objMode| u)))
   (when (atom labmode) (setq labmode (list labmode)))
   (if (eq (getdatabase expr 'constructorkind) '|domain|)
     (|sayMSG| (|concat| " " label labmode rhs (|form2String| expr)))
     (|mathprint|
      (cons 'concat
        (cons label
          (append labmode
            (cons rhs
              (cons (|outputFormat| expr (|objMode| u)) nil))))))))
  nil)))))

---

defun displayType

[sayMSG p331]
[fixObjectForPrinting p456]
[pname p1045]
[prefix2String p??]
[objMode p??]
[concat p1047]
[$op p??]

— defun displayType —

(defun |displayType| (|$op| u omitVariableNameIfTrue)
  (declare (special |$op|) (ignore omitVariableNameIfTrue))
  (let (type)
    (if (null u)
      (|sayMSG|
       (list " Type of value of " (|fixObjectForPrinting| (pname |$op|))
         ": (none)"))
      (progn
       (setq type (|prefix2String| (|objMode| u)))
       (when (atom type) (setq type (list type)))
       nil)))))))
18.2. FUNCTIONS

(defun getAndSay (v prop)
  (let (val)
    (if (setq val (getI v prop))
      (sayMSG (cons 'val | | (cons val (cons '%% nil))))
      (sayMSG (cons 'none | | nil))))

(defun displayProperties
  (getInterpMacroNames p)
  (getParserMacroNames p)
  (remdup p)
  (qcdr p)
  (qcar p)
  (msort p)
  (getWorkspaceNames p)
  (sayKeyedMsg p)
  (interpFunctionDepAlists p)
  (isInternalMapName p)
  (getIProplist p)
  (getAndSay p)
  (displayValue p)
  (getI p)
  (displayCondition p)
  (displayMode p)
  (displayType p)
  (fixObjectForPrinting p))
— defun displayProperties —

(defun displayProperties (option al)
  (let (($dependentAlist| |$dependeeAlist| tmp1 opt imacs pmacs macros vl pl
        tmp2 vone prop val propsSeen)
    (declare (special |$dependentAlist| |$dependeeAlist| |$frameMessages|
                |$interpreterFrameName|))
    (setq $dependentAlist nil)
    (setq $dependeeAlist nil)
    (setq tmp1 (or al (cons '|properties| nil)))
    (setq opt (car tmp1))
    (setq vl (cdr tmp1))
    (setq imacs (|getInterpMacroNames|))
    (setq pmacs (|getParserMacroNames|))
    (setq macros (remdup (append imacs pmacs)))
    (when (or
      (and (consp vl) (eq (qcdr vl) nil) (eq (qcar vl) '|all|))
      (null vl))
      (setq vl (msort (append (|getWorkspaceNames|) macros)))))
    (when |$frameMessages|
      (|sayKeyedMsg| 'S2IZ0065 (cons |$interpreterFrameName| nil)))
    (cond
      ((null vl)
        (if (null |$frameMessages|)
            (|sayKeyedMsg| 'S2IZ0066 nil)
          (|sayKeyedMsg| 'S2IZ0067 (cons |$interpreterFrameName| nil)))
      (t
        (|interpFunctionDepAlists|)
        (do ((G166440 vl (cdr G166440)) (v nil))
            ((or (atom G166440) (progn (setq v (car G166440)) nil)) nil)
          (seq (exit

...
(cond
  ((isInternalMapName v) 'iterate)
  (t
    (setq pl (getIProplist v))
    (cond
      ((eq option 'flags)
       (getAndSay v 'flags))
      ((eq option 'value)
       (displayValue v (getI v 'value) nil))
      ((eq option 'condition)
       (displayCondition v (getI v 'condition) nil))
      ((eq option 'mode)
       (displayMode v (getI v 'mode) nil))
      ((eq option 'type)
       (displayType v (getI v 'value) nil))
      ((eq option 'properties)
       (cond
        ((eq v '--flags--)
         nil)
        ((and (consp pl)
           (progn
             (setq tmp2 (qcar pl))
             (and (consp tmp2) (eq (qcar tmp2) 'cacheInfo)))))
         nil)
        (t
         (setq vone (fixObjectForPrinting v))
         (sayMSG (cons "Properties of" (append (bright (prefix2String vone)) (cons ":" nil))))
         (cond
           ((null pl)
            (cond
              ((member v pmacs)
               (sayMSG " This is a user-defined macro.")
               (displayParserMacro v))
              ((isInternalMacro v)
               (sayMSG " This is a system-defined macro.")
               (displayMacro v))
              (t
               (sayMSG " none")))
            (t
             (setq propsSeen nil)
             (do ((G166451 pl (cdr G166451)) (G166425 nil))
               ((or (atom G166451)
                 (progn (setq G166425 (car G166451)) nil)
                 (progn
                   (setq prop (car G166425))
                   (setq val (cdr G166425))
                   G166425))}
               (do ((G166451 pl (cdr G166451)) (G166425 nil))
                 ((or (atom G166451)
                   (setq G166425 (car G166451)) nil)
                   (progn
                     (setq prop (car G166425))
                     (setq val (cdr G166425))
                     G166425))}
CHAPTER 18. SYSTEM COMMAND HANDLING

(defun displayParserMacro
  [pfPrintSrcLines p??]
  [$pfMacros p97]

  — defun displayParserMacro —

  (defun |displayParserMacro| (m)
    (let ((m (assq m |$pfMacros|)))
      (declare (special |$pfMacros|)))
      ...
18.2. FUNCTIONS

(when m (|pfPrintSrcLines| (caddr m))))

---

defun displayCondition

[bright p??]
[sayBrightly p??]
[concat p1047]
[pred2English p??]

---

defun displayCondition

(defun displayCondition (v condition giveVariableIfNil)
  (let (varPart condPart)
    (when giveVariableIfNil (setq varPart (cons '| of| (|bright| v))))
    (setq condPart (or condition '|true|))
    (|sayBrightly|
     (|concat| '| condition| varPart '|: | (|pred2English| condPart)))))))

---

defun interpFunctionDepAlists

[putalist p??]
[getalist p??]
[getFlag p??]
[$e p??]
[$dependeeAlist p??]
[$dependentAlist p??]
[$InteractiveFrame p??]

---

defun interpFunctionDepAlists

(defun interpFunctionDepAlists ()
  (let ($e)
    (declare (special $e |$dependeeAlist| |$dependentAlist| |
                          $InteractiveFrame|))
    (setq $e |$InteractiveFrame|)
    (setq |$dependentAlist| (cons (cons nil nil) nil))
    (setq |$dependeeAlist| (cons (cons nil nil) nil))
    (mapcar #'(lambda (dep)
      (let (dependee dependent)
        (setq dependee (first dep)))))
(setq dependent (second dep))
(setq |$dependentAlist|
  (putalist |$dependentAlist| dependee
    (cons dependent (getalist |$dependentAlist| dependee))))
(spadlet |$dependeeAlist|
  (putalist |$dependeeAlist| dependent
    (cons dependee (getalist |$dependeeAlist| dependent))))))
(|getFlag| '|$dependencies|)))

---

defun displayModemap

[bright p??]
[sayBrightly p??]
[concat p1047]
[formatSignature p??]

— defun displayModemap —

(defun |displayModemap| (v val giveVariableIfNil)
  (labels (g (v mm giveVariableIfNil)
    (let (local signature fn varPart prefix)
      (setq local (caar mm))
      (setq signature (cdar mm))
      (setq fn (cadr mm))
      (unless (eq local '|interpOnly|)
        (spadlet varPart (unless giveVariableIfNil (cons " of" (|bright| v))))
        (spadlet prefix
          (cons '|' Compiled function type| (append varPart (cons '|' nil))))
        (|sayBrightly| (|concat| prefix (|formatSignature| signature))))))
  (mapcar #'(lambda (x) (g v x giveVariableIfNil) val)))))

---

defun displayMode

[bright p??]
[fixObjectForPrinting p456]
[sayBrightly p??]
[concat p1047]
[prefix2String p??]
18.2. FUNCTIONS

---

<table>
<thead>
<tr>
<th>defun displayMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defun</td>
</tr>
<tr>
<td>(let (varPart)</td>
</tr>
<tr>
<td>(when mode</td>
</tr>
<tr>
<td>(unless giveVariableIfNil</td>
</tr>
<tr>
<td>(setq varPart (cons '</td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td>(</td>
</tr>
</tbody>
</table>
| (|prefix2String| mode))))))

---

defun Split into tokens delimited by spaces

[stripSpaces p471]

---

<table>
<thead>
<tr>
<th>defun dumbTokenize</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defun</td>
</tr>
<tr>
<td>(let (inString token (tokenStart 0) previousSpace tokenList)</td>
</tr>
<tr>
<td>(dotimes (i (1- (</td>
</tr>
<tr>
<td>(cond</td>
</tr>
<tr>
<td>((char= (elt str i) #&quot;) ; don’t split strings</td>
</tr>
<tr>
<td>(setq inString (null inString))</td>
</tr>
<tr>
<td>(setq previousSpace nil))</td>
</tr>
<tr>
<td>((and (char= (elt str i) #\space) (null inString))</td>
</tr>
<tr>
<td>(unless previousSpace</td>
</tr>
<tr>
<td>(setq token (</td>
</tr>
<tr>
<td>(setq tokenList (cons token tokenList))</td>
</tr>
<tr>
<td>(setq tokenStart (1+ i))</td>
</tr>
<tr>
<td>(setq previousSpace t))</td>
</tr>
<tr>
<td>(t</td>
</tr>
<tr>
<td>(setq previousSpace nil))))</td>
</tr>
<tr>
<td>(setq tokenList (cons (</td>
</tr>
</tbody>
</table>
| (nreverse tokenList))

---

defun Convert string tokens to their proper type

[isIntegerString p468]

---

<table>
<thead>
<tr>
<th>defun tokTran</th>
</tr>
</thead>
</table>
(defun |tokTran| (tok)
  (let (tmp)
    (if (stringp tok)
        (cond
            ((eql (|#| tok) 0) nil)
            ((setq tmp (|isIntegerString| tok)) tmp)
            ((char= (elt tok 0) #" ) (subseq tok 1 (1- (|#| tok))))
            (t (intern tok)))
        tok)))

---

defun Is the argument string an integer?

— defun isIntegerString 0 —

(defun |isIntegerString| (tok)
  (multiple-value-bind (int len) (parse-integer tok :junk-allowed t)
    (when (and int (= len (length tok))) int)))

---

defun Handle parsed system commands

[dumbTokenize p467]
[parseSystemCmd p469]
[tokTran p467]
[systemCommand p448]

— defun handleParsedSystemCommands —

(defun |handleParsedSystemCommands| (unabr optionList)
  (declare (ignore unabr))
  (let (restOptionList parcmd trail)
    (setq restOptionList (mapcar #'|dumbTokenize| (cdr optionList)))
    (setq parcmd (|parseSystemCmd| (car optionList)))
    (setq trail
      (mapcar #'(lambda (opt)
               (mapcar #'(lambda (tok) (|tokTran| tok)) opt)) restOptionList))
    (|systemCommand| (cons parcmd trail))))

——
18.2. FUNCTIONS

defun Parse a system command

defun parseSystemCmd (opt)
(let (spaceIndex)
(if (setq spaceIndex (search " " opt))
(list
(tokTran (stripSpaces (subseq opt 0 spaceIndex)))
(parseFromString (stripSpaces (subseq opt spaceIndex)))
(mapcar #'tokTran (dumbTokenize opt)))))

defun Get first word in a string

defun getFirstWord (string)
(let (spaceIndex)
(setq spaceIndex (search " " string))
(if spaceIndex
(stripSpaces (subseq string 0 spaceIndex))
string)))

defun Unabbreviate keywords in commands
470

CHAPTER 18. SYSTEM COMMAND HANDLING

— defun unAbbreviateKeyword —
(defun |unAbbreviateKeyword| (x)
(let (xp)
(declare (special |$systemCommands| |$currentLine| $syscommands line))
(setq xp (|selectOptionLC| x $syscommands ’|commandErrorIfAmbiguous|))
(cond
((null xp)
(setq xp ’|system|)
(setq line (concat ")system " (substring line 1 (1- (|#| line)))))
(spadlet |$currentLine| line)))
(|selectOption| xp (|commandsForUserLevel| |$systemCommands|)
’|commandUserLevelError|)))

———-

defun The command is ambiguous error
[commandAmbiguityError p452]
[$oldline p449]
[line p??]
— defun commandErrorIfAmbiguous —
(defun |commandErrorIfAmbiguous| (x u)
(declare (special $oldline line))
(when u
(setq $oldline line)
(|commandAmbiguityError| ’|command| x u)))

———[stripSpaces p471]
[nplisp p472]
[stripLisp p471]
[sayKeyedMsg p329]
[npboot p472]
[npsystem p472]
[npsynonym p473]
[member p1048]
[concat p1047]
— defun handleNoParseCommands —
(defun |handleNoParseCommands| (unab string)


18.2. FUNCTIONS

(defun Remove the spaces surrounding a string

TPDHERE: This should probably be a macro or eliminated

---

(defun stripSpaces (str)
  (string-trim '(#\space) str))

---

(defun Remove the lisp command prefix

---

(defun stripLisp (str)
(if (string= (subseq str 0 4) "lisp")
  (subseq str 4)
  str))

defun Handle the )lisp command

\[\texttt{\$ans p??}\]

---

(defun \texttt{nplisp} 0)

(defun \texttt{nplisp} (str)
  (declare (special \texttt{\$ans}))
  (setq \texttt{\$ans} (eval (read-from-string str)))
  (format t "\texttt{Value = \$\%S\%} \texttt{\$ans})")

---

defun The )boot command is no longer supported

\texttt{TPDHERE: Remove all boot references from top level}

---

(defun \texttt{npboot} 0)

(defun \texttt{npboot} (str)
  (declare (ignore str))
  (format t "The )boot command is no longer supported")

---

defun Handle the )system command

Note that unAbbreviateKeyword returns the word “system” for unknown words so we have to search for this case. This complication may never arrive in practice. \texttt{[sayKeyedMsg p329]}

---

(defun \texttt{npsystem} (unab str)
  (let (spaceIndex sysPart)
    (setq spaceIndex (search " " str))
    (cond
      ((null spaceIndex) \[\texttt{|sayKeyedMsg| 'S2IZ0080 (list str)})
      (t}
(setq sysPart (subseq str 0 spaceIndex))
(if (search sysPart (string unab))
 (obey (subseq str (1+ spaceIndex)))
 (|sayKeyedMsg| 'S2IZ0080 (list sysPart)))))))

defun Handle the )synonym command

(defun npProcessSynonym (unab str)
  (declare (ignore unab))
  (npProcessSynonym str))

defun Handle the synonym system command

(defun npProcessSynonym (str)
  (let (pair)
    (declare (special |$CommandSynonymAlist|))
    (if (= (length str) 0)
      (printSynonyms nil)
    (progn
      (setq pair (|processSynonymLine| str))
      (if |$CommandSynonymAlist|
        (putalist |$CommandSynonymAlist| (car pair) (cdr pair))
        (setq |$CommandSynonymAlist| (cons pair nil)))
      (terminateSystemCommand))))
defun printSynonyms

(defun printSynonyms (patterns)
  (prog (ls t1)
    (declare (special $CommandSynonymAlist $linelength))
    (centerAndHighlight 'System Command Synonyms $linelength (specialChar 'hbar))
    (setq ls
      (filterListOfStringsWithFn patterns
        (do ((t2 (synonymsForUserLevel $CommandSynonymAlist) (cdr t2)))
            ((atom t2) (nreverse0 t1))
          (push (cons (princ-to-string (caar t2)) (cdar t2)) t1))
        (function car))
    (printLabelledList ls "user" "synonyms" "") patterns)))

-----

defun Print a list of each matching synonym

The prefix goes before each element on each side of the list, eg, "")" [sayMessage]

(defun printLabelledList (ls label1 label2 prefix patterns)
  (let (comm syn wid)
    (if (null ls)
      (if (null patterns)
        (sayMessage (list " No " label1 "-defined " label2 " in effect."))
        (sayMessage
          "(" No ",label1 "-defined ",label2 " satisfying patterns:"))
      (printLabelledList (cons (caar ls) (cdar ls)) label1 label2 prefix patterns)))

-----

\[ \text{\texttt{|\%l| " " |\%b| ,@(append (|blankList| patterns) (list '|\%d|)))} \]
(progn
  (when patterns
    (|sayMessage|
     `(,label1 "-defined ", label2 " satisfying patterns:" |\%l| " "
     |\%b|
     ,(append (|blankList| patterns) (list '|\%d|)))))
  (do ((t1 ls (cdr t1)))
      ((atom t1) nil)
    (setq syn (caar t1))
    (setq comm (cdar t1))
    (when (string= (substring syn 0 1) "|")
      (setq syn (substring syn 1 nil))
    (when (string= syn "%i") (setq syn "%i "))
    (setq wid (max (- 30 (|entryWidth| syn)) 1))
    (|sayBrightly|
     (|concat| '|\%b| prefix syn '|\%d| (|fillerSpaces| wid ".")
     " " prefix comm))
    (|sayBrightly| ""))))

---

defvar $tokenCommands

This is a list of the commands that expect the interpreter to parse their arguments. Thus the history command expects that Axiom will have tokenized and validated the input before calling the history function.

--- initvars ---

(defvar $tokenCommands| nil)

---

--- postvars ---

(eval-when (eval load)
  (setq $tokenCommands| (|abbreviations|
    |cd|
    |clear|
    |close|
    |compiler|
    |depends|
    |display|
    |describe|
    |edit|)
defvar $InitialCommandSynonymAlist

Axiom can create “synonyms” for commands. We create an initial table of synonyms which are in common use.

— initvars —

(defun axiomVersion ()
  (declare (special *build-version* *yearweek*))
  (concatenate 'string "Axiom " *build-version* " built on " *yearweek*)))
(eval-when (eval load)
  (setq |$InitialCommandSynonymAlist|
    '(
      (?| . "what commands")
      (ap| . "what things")
      (apr| . "what things")
      (apropos| . "what things")
      (cache| . "set functions cache")
      (cl| . "clear")
      (cls| . "zsystemdevelopment )cls")
      (cms| . "system")
      (co| . "compiler")
      (d| . "display")
      (dep| . "display dependents")
      (dependents| . "display dependents")
      (e| . "edit")
      (expose| . "set expose add constructor")
      (fc| . "zsystemdevelopment )c")
      (fd| . "zsystemdevelopment )d")
      (fdt| . "zsystemdevelopment )dt")
      (fct| . "zsystemdevelopment )ct")
      (fctl| . "zsystemdevelopment )ctl")
      (fe| . "zsystemdevelopment )e")
      (fec| . "zsystemdevelopment )ec")
      (fect| . "zsystemdevelopment )ect")
      (fns| . "exec spadfn")
      (fortran| . "set output fortran")
      (h| . "help")
      (hd| . "system hypertex &")
      (kclam| . "boot clearClams ( )")
      (killcaches| . "boot clearConstructorAndLisplibCaches ( )")
      (patch| . "zsystemdevelopment )patch")
      (pause| . "zsystemdevelopment )pause")
      (prompt| . "set message prompt")
      (recurrence| . "set functions recurrence")
      (restore| . "history )restore")
      (save| . "history )save")
      (startGraphics| . "system $AXIOM/lib/viewman &")
      (startNAGLink| . "system $AXIOM/lib/nagman &")
      (stopGraphics| . "lisp (|sockSendSignal| 2 15)")
      (stopNAGLink| . "lisp (|sockSendSignal| 8 15)")
      (time| . "set message time")
      (type| . "set message type")
      (unexpose| . "set expose drop constructor")
      (up| . "zsystemdevelopment )update")
    )
  )
defvar $CommandSynonymAlist

The actual list of synonyms is initialized to be the same as the above initial list of synonyms. The user synonyms that are added during a session are pushed onto this list for later lookup.

— initvars —

(defun ncloopCommand

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

The only exception is the $include function which inserts the contents of a file inline in the input stream. This is useful for processing the read of input files. [ncloopPrefix? p479] [ncloopInclude? p621] [$systemCommandFunction p??] [$systemCommandFunction p??]

— defun ncloopCommand —
### 18.2. FUNCTIONS

```lisp
(defun |ncloopCommand| (line n)
  (let (a)
    (declare (special |$systemCommandFunction|))
    (if (setq a (|ncloopPrefix?| "include" line))
      (|ncloopInclude1| a n)
      (progn
        (funcall |$systemCommandFunction| line)
        n))))
```

---

#### defun ncloopPrefix?

If we find the prefix string in the whole string starting at position zero we return the remainder of the string without the leading prefix.

---

```lisp
(defun |ncloopPrefix?| (prefix whole)
  (when (eql (search prefix whole) 0)
    (subseq whole (length prefix))))
```

---

#### defun selectOptionLC

```lisp
(defun |selectOptionLC| (x l errorFunction)
  (|selectOption| (downcase (|object2Identifier| x)) l errorFunction))
```

---

#### defun selectOption

```lisp
(defun |selectOption| (p l errorFunction)
  (|member| p l)
  (|identp| p)
  (|stringPrefix?| p)
  (|pname| p))
```
defun selectOption
(defun selectOption (x l errorfunction)
  (let (u y)
    (cond
      (member x l) x
      (null (identp x))
      (errorfunction (funcall errorfunction x u))
      (t nil))
    (t
     (setq u
       (let (t0)
         (do ((t1 l (cdr t1)) (y nil))
          ((or (atom t1) (progn (setq y (car t1)) nil)) (nreverse0 t0))
           (if (stringPrefix? (pname x) (pname y))
             (setq t0 (cons y t0)))))
     (cond
      (and (consp u) (eq (qcdr u) nil) (progn (setq y (qcar u)) t))
      (errorfunction (funcall errorfunction x u))
      (t nil))))))
Chapter 19

)abbreviations help page Command

19.1 abbreviations help page man page

— abbreviations.help —

====================================================================
A.2. )abbreviation
====================================================================

User Level Required: compiler

Command Syntax:

- )abbreviation query [nameOrAbbrev]
- )abbreviation category abbrev fullname []{quiet}
- )abbreviation domain abbrev fullname []{quiet}
- )abbreviation package abbrev fullname []{quiet}
- )abbreviation remove nameOrAbbrev

Command Description:

This command is used to query, set and remove abbreviations for category, domain and package constructors. Every constructor must have a unique abbreviation. This abbreviation is part of the name of the subdirectory under which the components of the compiled constructor are stored. Furthermore, by issuing this command you let the system know what file to load automatically if you use a new constructor. Abbreviations must start with a letter and then be followed by up to seven letters or digits. Any letters appearing in the abbreviation must be in uppercase.
When used with the query argument, this command may be used to list the name associated with a particular abbreviation or the abbreviation for a constructor. If no abbreviation or name is given, the names and corresponding abbreviations for all constructors are listed.

The following shows the abbreviation for the constructor List:

```plaintext
>abbreviation query List
```

The following shows the constructor name corresponding to the abbreviation NNI:

```plaintext
>abbreviation query NNI
```

The following lists all constructor names and their abbreviations.

```plaintext
>abbreviation query
```

To add an abbreviation for a constructor, use this command with category, domain or package. The following add abbreviations to the system for a category, domain and package, respectively:

```plaintext
>abbreviation domain SET Set
>abbreviation category COMPCAT ComplexCategory
>abbreviation package LIST2MAP ListToMap
```

If the `)quiet` option is used, no output is displayed from this command. You would normally only define an abbreviation in a library source file. If this command is issued for a constructor that has already been loaded, the constructor will be reloaded next time it is referenced. In particular, you can use this command to force the automatic reloading of constructors.

To remove an abbreviation, the remove argument is used. This is usually only used to correct a previous command that set an abbreviation for a constructor name. If, in fact, the abbreviation does exist, you are prompted for confirmation of the removal request. Either of the following commands will remove the abbreviation VECTOR2 and the constructor name VectorFunctions2 from the system:

```plaintext
>abbreviation remove VECTOR2
>abbreviation remove VectorFunctions2
```

Also See:
- `)compile`
19.2 Functions

defun abbreviations

AbbreviationsSpad2Cmd p483

— defun abbreviations —

(defun abbreviations (l)
  (abbreviationsSpad2Cmd l))

---

defun abbreviationsSpad2Cmd

ListConstructorAbbreviations p484

Abbreviation? p??

AbbQuery p536

DlDatabase p1009

Size p1045

SayKeyedMsg p329

 MkUserConstructorAbbreviation p??

SetDatabase p1009

Seq p??

Exit p??

OpOf p??

HelpSpad2Cmd p572

SelectOptionLC p479

Qcar p??

Qcdr p??

$Options p??

— defun abbreviationsSpad2Cmd —

(defun abbreviationsSpad2Cmd (arg)
  (let (abopts quiet opt key type constructor t2 a b al)
    (declare (special $options))
    (if (null arg)
      (helpSpad2Cmd '(abbreviations))
      (progn
        (setq abopts '([query] [domain] [category] [package] [remove]))
        (setq quiet nil)
        (do ((t0 $options (cdr t0)) (t1 nil))
            ((or (atom t0)
                 (progn (setq t1 (car t0)) nil))))
      ))
(progn (progn (setq opt (car t1)) t1) nil))
(nil)
(setq opt (selectOptionLC opt '(|quiet|) '|optionError|))
(when (eq opt '|quiet|) (setq quiet t)))
(when
(and (consp arg)
(progn
(setq opt (qcar arg))
(setq al (qcdr arg))
t))
(setq key (|lop0f| (car al))))
(setq type (selectOptionLC opt abopts '|optionError|))
(cond
((eq type '|query|)
(cond
((null al) (listConstructorAbbreviations)))
((setq constructor (abbreviation? key))
(|abbQuery| constructor))
(t (|abbQuery| key))))
((eq type '|remove|)
(deldatabase key 'abbreviation))
((oddp (size al))
(|sayKeyedMsg| 's2iz0002 (list type)))
(t
(do () (nil nil)
(seq
(exit
(cond
((null al) (return '|fromLoop|)))
(t
(setq t2 al)
(setq a (car t2))
(setq b (cadr t2))
(setq al (cddr t2))
(|mkUserConstructorAbbreviation| b a type)
(setdatabase b 'abbreviation a)
(setdatabase b 'constructorkind type))))))
(unless quiet
(|sayKeyedMsg| 's2iz0001 (list a type (|opOf| b)))))))

defun listConstructorAbbreviations
[upcase p??]
[queryUserKeyedMsg p??]
[string2id-n p??]
[whatSpad2Cmd p940]
— defun listConstructorAbbreviations —

(defun listConstructorAbbreviations ()
  (let (x)
    (setq x (upcase (queryUserKeyedMsg 's2iz0056 nil)))
    (if (member (string2id-n x 1) '(Y YES))
      (progn
        (whatSpad2Cmd '(categories))
        (whatSpad2Cmd '(domains))
        (whatSpad2Cmd '(packages))
        (sayKeyedMsg 's2iz0057 nil))))

[sayKeyedMsg p329]
Chapter 20

)boot help page Command

20.1 boot help page man page

— boot.help —

====================================================================
A.3. )boot
====================================================================

User Level Required: development

Command Syntax:

- )boot bootExpression

Command Description:

This command is used by AXIOM system developers to execute expressions written in the BOOT language. For example,

)boot times3(x) == 3*x

creates and compiles the Lisp function ‘‘times3’’ obtained by translating the BOOT code.

Also See:
- )fin
- )lisp
- )set
- )system
20.2 Functions

This command is in the list of $noParseCommands 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2

\footnote{“fin” (31.1 p 548) “lisp” (?,??) “set” (45.36 p 808) “system” (?,??)}
Chapter 21

)browse help page Command

21.1 browse help page man page

— browse.help —

User Level Required: development

Command Syntax:

)browse

Command Description:

This command is used by Axiom system users to start the Axiom top level loop listening for browser connections.

21.2 Overview

The Axiom book on the help browser is a complete rewrite of the hyperdoc mechanism. There are several components that were needed to make this function. Most of the web browser components are described in bookvol11.pamphlet. This portion describes some of the design issues needed to support the interface.

The axServer command takes a port (defaulting to 8085) and a program to handle the browser interaction (defaulting to multiServ). The axServer function opens the port, constructs the
stream, and passes the stream to multiServ. The multiServ loop processes one interaction at a time.

So the basic process is that the Axiom ")browse" command opens a socket and listens for http requests. Based on the type of request (either 'GET' or 'POST') and the content of the request, which is one of:

- command - algebra request/response
- lispcall - a lisp s-expression to be evaluated
- showcall - an Axiom )show command

the multiServ function will call a handler function to evaluate the command line and construct a response. GET requests result in a new browser page. POST requests result in an inline result.

Most responses contain the fields:

- stepnum - this is the Axiom step number
- command - this is the original command from the browser
- algebra - this is the Axiom 2D algebra output
- mathml - this is the MathML version of the Axiom algebra
- type - this is the type of the Axiom result

### 21.3 Browsers, MathML, and Fonts

This work has the Firefox browser as its target. Firefox has built-in support for MathML, javascript, and XMLHttpRequests. More details are available in bookvol11.pamphlet but the very basic machinery for communication with the browser involves a dance between the browser and the multiServ function (see the axserver.spad.pamphlet).

In particular, a simple request is embedded in a web page as:

```html
<ul>
  <li>
    <input type="submit" id="p3" class="subbut"
      onclick="makeRequest('p3');" value="sin(x)" />
    <div id="ansp3"><div></div></div>
  </li>
</ul>
```

which says that this is an html “input” field of type “submit”. The CSS display class is “subbut” which is of a different color than the surrounding text to make it obvious that you can click on this field. Clickable fields that have no response text are of class “noresult”.

21.4. THE AXSERVER/MULTISERV LOOP

The javascript call to “makeRequest” gives the “id” of this input field, which must be unique in the page, as an argument. In this case, the argument is ‘p3’. The “value” field holds the display text which will be passed back to Axiom as a command.

When the result arrives the “showanswer” function will select out the mathml field of the response, construct the “id” of the html div to hold the response by concatenating the string “ans” (answer) to the “id” of the request resulting, in this case, as “ansp3”. The “showanswer” function will find this div and replace it with a div containing the mathml result.

The “makeRequest” function is:

```javascript
function makeRequest(arg) {
    http_request = new XMLHttpRequest();
    var command = commandline(arg);
    //alert(command);
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = handleResponse;
    http_request.setRequestHeader('Content-Type', 'text/plain');
    http_request.send("command="+command);
    return(false);
}
```

It contains a request to open a local server connection to Axiom, sets “handleResponse” as the function to call on reply, sets up the type of request, fills in the command field, and sends off the http request.

When a response is received, the “handleResponse” function checks for the correct reply state, strips out the important text, and calls “showanswer”.

```javascript
function handleResponse() {
    if (http_request.readyState == 4) {
        if (http_request.status == 200) {
            showanswer(http_request.responseText,'mathAns');
        } else {
            alert(‘There was a problem with the request.’+ http_request.statusText);
        }
    }
}
```

See bookvol11.pamphlet for further details.

21.4 The axServer/multiServ loop

The basic call to start an Axiom browser listener is:

```
)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV
```
This call sets the port, opens a socket, attaches it to a stream, and then calls “multiServ” with that stream. The “multiServ” function loops serving web responses to that port.

### 21.5 The `)browse` command

In order to make the whole process cleaner the function “)browse” handles the details. This code creates the command-line function for `)browse`:

The `browse` function does the internal equivalent of the following 3 command line statements:

```lisp
)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV
```

which causes Axiom to start serving web pages on port 8085.

For those unfamiliar with calling algebra from lisp there are a few points to mention.

The `loadLib` needs to be called to load the algebra code into the image. Normally this is automatic but we are not using the interpreter so we need to do this “by hand”.

Each algebra file contains a ”constructor function” which builds the domain, which is a vector, and then caches the vector so that every call to the constructor returns an EQ vector, that is, the same vector. In this case, we call the constructor `[AxiomServer]`.

The `axServer` function was mangled internally to `[AXSERV;axServer;IMV;2]`. The `multiServ` function was mangled to `[AXSERV;multiServ;SeV;3]` Note well that if you change `axserver.spad` these names might change which will generate the error message along the lines of:

```
System error:  
The function \$\vert$AXSERV;axServer;IMV;2\$\vert$ is undefined.
```

To fix this you need to look at `int/algebra/AXSERV.nrlib/code.lsp` and find the new mangled function name. A better solution would be to dynamically look up the surface names in the domain vector.

Each Axiom function expects the domain vector as the last argument. This is not obvious from the call as the interpreter supplies it. We must do that “by hand”.

We don’t call the `multiServ` function. We pass it as a parameter to the `axServer` function. When it does get called by the `SPADCALL` macro it needs to be a lisp pair whose car is the function and whose cdr is the domain vector. We construct that pair here as the second argument to `axServer`. The third, hidden, argument to `axServer` is the domain vector which we supply “by hand”.

The socket can be supplied on the command line but defaults to 8085. Axiom supplies the arguments as a list.
21.6 Variables Used

21.7 Functions

[set p808]
[loadLib p1035]
[AxiomServer p??]
[AXSERV;axServer;IMV;2 p??]

— defun browse —

(defun browse (socket)
  (let (axserv browser)
    (if socket
      (setq socket (car socket))
      (setq socket 8085))
    (setq ’(mes |auto| off))
    (setq ’(out |mathml| on))
    (loadLib ’|AxiomServer|)
    (setq axserv (|AxiomServer|))
    (setq browser
      (|AXSERV;axServer;IMV;2| socket
       (cons ’|AXSERV;multiServ;SeV;3| axserv) axserv))))

Now we have to bolt it into Axiom. This involves two lookups.

We create the lisp pair

  (|browse| . |development|)

and cons it into the $systemCommands command table. This allows the command to be executed in development mode. This lookup decides if this command is allowed. It also has the side-effect of putting the command into the $SYSCOMMANDS variable which is used to determine if the token is a command.

21.8 The server support code
Chapter 22

)cd help page Command

22.1 cd help page man page

--- cd.help ---

====================================
A.4. )cd
====================================

User Level Required: interpreter

Command Syntax:

- )cd directory

Command Description:

This command sets the AXIOM working current directory. The current directory is used for looking for input files (for )read), AXIOM library source files (for )compile), saved history environment files (for )history )restore), compiled AXIOM library files (for )library), and files to edit (for )edit). It is also used for writing spool files (via )spool), writing history input files (via )history )write) and history environment files (via )history )save), and compiled AXIOM library files (via )compile).

If issued with no argument, this command sets the AXIOM current directory to your home directory. If an argument is used, it must be a valid directory name. Except for the ‘’’ at the beginning of the command, this has the same syntax as the operating system cd command.

Also See:
o )compile

495
CHAPTER 22. )CD HELP PAGE COMMAND

- )edit
- )history
- )library
- )read
- )spool

22.2 Variables Used

22.3 Functions

1 “edit” (30.2 p 544) “history” (34.4 p 582) “library” (66.1 p 1013) “read” (42.1 p 642) “spool” (?? p ??)
Chapter 23

`)clear help page Command`

23.1 `clear help page man page`

--- clear.help ---

A.6. `)clear`

User Level Required: interpreter

Command Syntax:

- `)clear all`
- `)clear completely`
- `)clear properties all`
- `)clear properties obj1 [obj2 ...]`
- `)clear value all`
- `)clear value obj1 [obj2 ...]`
- `)clear mode all`
- `)clear mode obj1 [obj2 ...]`

Command Description:

This command is used to remove function and variable declarations, definitions and values from the workspace. To empty the entire workspace and reset the step counter to 1, issue

`)clear all`

To remove everything in the workspace but not reset the step counter, issue

497
To remove everything about the object $x$, issue

)clear properties $x$

To remove everything about the objects $x$, $y$ and $f$, issue

)clear properties $x$ $y$ $f$

The word properties may be abbreviated to the single letter ‘‘p’’.

)clear p all
)clear p $x$
)clear p $x$ $y$ $f$

All definitions of functions and values of variables may be removed by either

)clear value all
)clear v all

This retains whatever declarations the objects had. To remove definitions and values for the specific objects $x$, $y$ and $f$, issue

)clear value $x$ $y$ $f$
)clear v $x$ $y$ $f$

To remove the declarations of everything while leaving the definitions and values, issue

)clear mode all
)clear m all

To remove declarations for the specific objects $x$, $y$ and $f$, issue

)clear mode $x$ $y$ $f$
)clear m $x$ $y$ $f$

The )display names and )display properties commands may be used to see what is currently in the workspace.

The command

)clear completely

does everything that )clear all does, and also clears the internal system function and constructor caches.

Also See:
- )display
23.2 Variables Used

defvar $clearOptions

— initvars —

(defvar |$clearOptions| '(|modes| |operations| |properties| |types| |values|))

23.3 Functions

defun clear

[clearSpad2Cmd p500]

— defun clear —

(defun |clear| (1)
  (|clearSpad2Cmd| 1))

defvar $clearExcept

— initvars —

(defvar |$clearExcept| nil)

1 “display” (29.2 p 535) “history” (34.4 p 582) “undo” (53.4 p 922)
defun clearSpad2Cmd

TPDHERE: Note that this function also seems to parse out )except )completely and )scaches which don’t seem to be documented. [selectOptionLC p479]

[sayKeyedMsg p329]
[clearCmdAll p503]
[clearCmdCompletely p502]
[clearCmdSortedCaches p501]
[clearCmdExcept p504]
[clearCmdParts p505]
[updateCurrentInterpreterFrame p559]
[$clearExcept p499]
[$options p??]
[$clearOptions p499]

— defun clearSpad2Cmd —

(defun clearSpad2Cmd (l)
  (let (($clearExcept opt optlist arg)
         (declare (special $clearExcept $options $clearOptions)))
    (cond
      ($options)
      (setq $clearExcept
        (prog (t0)
          (setq t0 t)
          (return
            (do ((t1 nil (null t0))
                (t2 $options (cdr t2))
                (t3 nil))
                ((or t1
                    (atom t2)
                    (progn (setq t3 (car t2)) nil)
                    (progn (progn (setq opt (car t3)) t3) nil))
                    t0)
              (setq t0
                (and t0
                  (eq
                    (selectOptionLC opt '([except] 'optionError) '([except]))))))))
      (null l)
      (setq optlist
        (prog (t4)
          (setq t4 nil)
          (return
            (do ((t5 $clearOptions (cdr t5)) (x nil))
                ((or (atom t5) (progn (setq x (car t5)) nil)) t4)
              (setq t4 (append t4 `([%l] " " ,x))))))
        (sayKeyedMsg 's2iz0010 (list optlist))))


```
(defun clearCmdSortedCaches ()
  (let ((|$lookupDefaults| domain pair)
        (declare (special |$lookupDefaults| |$Void| |$ConstructorCache|)))
    (do ((t0 (hget |$ConstructorCache| '|SortedCache|) (cdr t0))
         (t1 nil))
        ((or (atom t0)
               (progn
                 (setq t1 (car t0))
                 (setq domain (cdr t1))
                 nil))
         nil)
     (setq pair (|compiledLookupCheck| '|clearCache| (list |$Void|) domain))
     (spadcall pair))))

(defun compiledLookupCheck
  (let ((|$lookupDefaults| domain pair)
        (declare (special |$lookupDefaults| |$Void| |$ConstructorCache|)))
    (do ((t0 (hget |$ConstructorCache| '|SortedCache|) (cdr t0))
         (t1 nil))
        ((or (atom t0)
               (progn
                 (setq t1 (car t0))
                 (setq domain (cdr t1))
                 nil))
         nil)
     (setq pair (|compiledLookupCheck| '|clearCache| (list |$Void|) domain))
     (spadcall pair))))
```
CHAPTER 23. )CLEAR HELP PAGE COMMAND

— defun compiledLookupCheck —

(defun compiledLookupCheck (op sig dollar)
  (let (fn)
    (setq fn (compiledLookup op sig dollar))
    (cond
      ((and (null fn) (eq op '^))
       (setq fn (compiledLookup '^ sig dollar)))
      ((and (null fn) (eq op '**))
       (setq fn (compiledLookup '** sig dollar)))
      (t nil))
    (cond
      (null fn)
      (keyedSystemError 'S2NR0001
        (list op (formatSignature sig) (elt dollar 0)))
      (t fn))))

defvar $functionTable

— initvars —

(defvar $functionTable nil)

——

defun clearCmdCompletely

[clearCmdAll p503]
[sayKeyedMsg p329]
clearClams p??
[clearConstructorCaches p??]
reclaim p37]
[$localExposureData p697]
[$xdatabase p??]
[$CatOfCatDatabase p??]
[$DomOfCatDatabase p??]
[$JoinOfCatDatabase p??]
[$JoinOfDomDatabase p??]
[$attributeDb p??]
[$functionTable p502]
defun clearCmdCompletely

(defvar $existingFiles p
(defvar $localExposureDataDefault p)

---

defun clearCmdCompletely

(defun clearCmdCompletely ()
  (declare (special $localExposureData $xdatabase
  $CatOfCatDatabase $DomOfCatDatabase
  $JoinOfCatDatabase $JoinOfDomDatabase
  $attributeDb $functionTable $existingFiles
  $localExposureDataDefault))
  (clearCmdAll)
  (setq $localExposureData (copy-seq $localExposureDataDefault))
  (setq $xdatabase nil)
  (setq $CatOfCatDatabase nil)
  (setq $DomOfCatDatabase nil)
  (setq $JoinOfCatDatabase nil)
  (setq $JoinOfDomDatabase nil)
  (setq $attributeDb nil)
  (setq $functionTable nil)
  (sayKeyedMsg 's2iz0013 nil)
  (clearClams)
  (clearConstructorCaches)
  (setq $existingFiles (make-hash-table :test #'equal))
  (sayKeyedMsg 's2iz0014 nil)
  (reclaim)
  (sayKeyedMsg 's2iz0015 nil))

---

defun clearCmdAll

[clearCmdSortedCaches p]
[untraceMapSubNames p]
[resetInCoreHist p]
[deleteFile p]
[histFileName p]
[updateCurrentInterpreterFrame p]
[clearMacroTable p]
[sayKeyedMsg p]
[frameRecord p]
[previousBindings p]
[variableNumberAlist p]
[InteractiveFrame p]
[useInternalHistoryTable p]
[internalHistoryTable p]
[frameMessages p]
defun clearCmdAll

(defun clearCmdAll ()
  (declare (special $frameRecord| $previousBindings| $variableNumberAlist|
    $InteractiveFrame| $useInternalHistoryTable| $internalHistoryTable|
    $frameMessages| $interpreterFrameName| $currentLine|)))

  (clearCmdSortedCaches)
  (setq $frameRecord| nil)
  (setq $previousBindings| nil)
  (setq $variableNumberAlist| nil)
  (untraceMapSubNames| /tracenames)
  (setq $InteractiveFrame| (list (list nil)))
  (resetInCoreHist)
  (when $useInternalHistoryTable|
    (setq $internalHistoryTable| nil)
  )
  (deleteFile| (histFileName|))
  (setq $IOindex| 1)
  (updateCurrentInterpreterFrame)
  (setq $currentLine| ")clear all")
  (clearMacroTable)
  (when $frameMessages|
    (sayKeyedMsg| 's2iz0011 (list $interpreterFrameName|))
    (sayKeyedMsg| 's2iz0012 nil)))

-------

defun clearMacroTable

(defun clearMacroTable ()
  (declare (special $pfMacros|))
  (setq $pfMacros| nil))

-------

defun clearCmdExcept

Clear all the options except the argument. 
[object2String| p??]
[stringPrefix| p??]
23.3. FUNCTIONS

---
defun clearCmdExcept

(defun clearCmdExcept (arg)
  (let ((opt (car arg)) (vl (cdr arg)))
    (declare (special $clearOptions!))
    (dolist (option $clearOptions)
      (unless (stringPrefix? (object2String opt) (object2String option))
        (clearCmdParts (cons option vl))))))

---
defun clearCmdParts

[selectOptionLC p479]
[rename p1045]
[types p??]
[modes p??]
[values p??]
[boot-equal p??]
[assocleft p??]
[remdup p??]
[assoc p??]
[isMap p??]
[get p??]
[exit p??]
[untraceMapSubNames p873]
[seq p??]
[recordOldValue p592]
[recordNewValue p591]
[deleteAssoc p??]
[sayKeyedMsg p329]
[getParserMacroNames p453]
[getInterpMacroNames p??]
[clearDependencies p??]
[member p1048]
[clearParserMacro p453]
[sayMessage p??]
[fixObjectForPrinting p456]
[$e p??]
[$InteractiveFrame p??]
[$clearOptions p499]
(defun clearCmdParts (arg)
  (let (|$e| (opt (car arg)) option pmacs imacs (vl (cdr arg)) p1 lm prop p2)
    (declare (special |$e| |$InteractiveFrame| |$clearOptions|))
    (setq option (|selectOptionLC| opt |$clearOptions| '|optionError|))
    (setq option (intern (pname option)))
    (setq option
      (case option
        (|types| '|mode|)
        (|modes| '|mode|)
        (|values| '|value|)
        (t option)))
    (if (null vl)
      (|sayKeyedMsg| 's2iz0055 nil)
      (progn
        (setq pmacs (|getParserMacroNames|))
        (setq imacs (|getInterpreterMacroNames|))
        (cond
          ((boot-equal vl '|all|)
            (setq vl (assocleft (caar |$InteractiveFrame|)))
            (setq vl (remdup (append vl pmacs))))
          (setq |$e| |$InteractiveFrame|)
          (do ((t0 vl (cdr t0)) (x nil))
              ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
            (|clearDependencies| x t)
            (when (and (eq option '|properties|) (|member| x pmacs))
              (|clearParserMacro| x))
            (when (and (eq option '|properties|)
                        (|member| x imacs)
                        (null (|member| x pmacs)))
              (|sayMessage| (cons "You cannot clear the definition of the system-defined macro "
                        (cons (|fixObjectForPrinting| x)
                        (cons (intern "." "BOOT") nil)))))
    (cond
      ((setq p1 (|assoc| x (caar |$InteractiveFrame|)))
       (cond
        ((|isMap| x)
         (seq
          (setq lm
            (|get| x '|localModemap| |$InteractiveFrame|))
          (cond
            ((consp lm)
             (exit (|untraceMapSubNames| (cons (cadar lm) nil))))))))
      (dolist (p2 (cdr p1)))))
(setq prop (car p2))
  (|recordOldValue| x prop (cdr p2))
  (|recordNewValue| x prop nil))
(setf (caar |$InteractiveFrame|)
  (|deleteAssoc| x (caar |$InteractiveFrame|))))
((setq p2 (|assoc| option (cdr p1)))
  (|recordOldValue| x option (cdr p2))
  (|recordNewValue| x option nil)
  (rplacd p2 nil))))
nil))
Chapter 24

)close help page Command

24.1 close help page man page

— close.help —

====================================================================
A.5. )close
====================================================================

User Level Required: interpreter

Command Syntax:

- )close
  - )close )quietly

Command Description:

This command is used to close down interpreter client processes. Such processes are started by HyperDoc to run AXIOM examples when you click on their text. When you have finished examining or modifying the example and you do not want the extra window around anymore, issue

)close

to the AXIOM prompt in the window.

If you try to close down the last remaining interpreter client process, AXIOM will offer to close down the entire AXIOM session and return you to the operating system by displaying something like

This is the last AXIOM session. Do you want to kill AXIOM?
Type "y" (followed by the Return key) if this is what you had in mind. Type "n" (followed by the Return key) to cancel the command.

You can use the )quietly option to force AXIOM to close down the interpreter client process without closing down the entire AXIOM session.

Also See:
- )quit
- )pquit

24.2 Functions

defun queryClients

Returns the number of active scratchpad clients [sockSendInt p??]
[sockGetInt p??]
[$SessionManager p??]
[$QueryClients p??]

--- defun queryClients ---

(defun |queryClients| ()
  (declare (special |$SessionManager| |$QueryClients|))
  ([sockSendInt |$SessionManager| |$QueryClients|])
  ([sockGetInt |$SessionManager| |$QueryClients|]))

---

defun close

[throwKeyedMsg p??]
[sockSendInt p??]
[closeInterpreterFrame p562]
[selectOptionLC p479]
[upcase p??]
[queryUserKeyedMsg p??]
[string2id-n p??]

---

1 "quit" (41.2 p 638) "pquit" (40.2 p 634)
24.2. FUNCTIONS

(defun close (args)
  (declare (ignore args))
  (let (numClients opt fullopt quiet x)
    (declare (special |$SpadServer| |$SessionManager| |$CloseClient|
                   |$currentFrameNum| |$options|))
    (if (null |$SpadServer|)
        (|throwKeyedMsg| 's2iz0071 nil))
    (progn
      (setq numClients (|queryClients|))
      (cond
        (> numClients 1)
          (|sockSendInt| |$SessionManager| |$CloseClient|)
          (|sockSendInt| |$SessionManager| |$currentFrameNum|)
          (|closeInterpreterFrame| nil)
        (t
          (do ((t0 |$options| (cdr t0)) (t1 nil))
                ((or (atom t0)
                     (progn (setq t1 (car t0)) nil)
                     (progn (progn (setq opt (car t1)) t1) nil))
               nil)
            (setq fullopt (|selectOptionLC| opt '|quiet| '|optionError|))
            (unless quiet (setq quiet (eq fullopt '|quiet|)))
            (cond
              (quiet
                (|sockSendInt| |$SessionManager| |$CloseClient|)
                (|sockSendInt| |$SessionManager| |$currentFrameNum|)
                (|closeInterpreterFrame| nil)
              (t
                (setq x (upcase (|queryUserKeyedMsg| 's2iz0072 nil)))
                (when (member (string2id-n x 1) '(yes y)) (bye))))))))
  ___
CHAPTER 24. )CLOSE HELP PAGE COMMAND
Chapter 25

)compile help page Command

25.1 compile help page man page

— compile.help —

====================================================================
A.7. )compile
====================================================================

User Level Required: compiler

Command Syntax:

- )compile
- )compile fileName
- )compile fileName.spad
- )compile directory/fileName.spad
- )compile fileName )quiet
- )compile fileName )noquiet
- )compile fileName )break
- )compile fileName )nobreak
- )compile fileName )library
- )compile fileName )nolibrary
- )compile fileName )vartrace
- )compile fileName )constructor nameOrAbbrev

Command Description:

You use this command to invoke the AXIOM library compiler. This compiles files with file extension .spad with the AXIOM system compiler. The command first looks in the standard system directories for files with extension .spad.
Should you not want the )library command automatically invoked, call )compile with the )nolibrary option. For example,

)compile mycode )nolibrary

By default, the )library system command exposes all domains and categories it processes. This means that the AXIOM interpreter will consider those domains and categories when it is trying to resolve a reference to a function. Sometimes domains and categories should not be exposed. For example, a domain may just be used privately by another domain and may not be meant for top-level use. The )library command should still be used, though, so that the code will be loaded on demand. In this case, you should use the )nolibrary option on )compile and the )noexpose option in the )library command. For example,

)compile mycode.spad )nolibrary
)library mycode )noexpose

Once you have established your own collection of compiled code, you may find it handy to use the )dir option on the )library command. This causes )library to process all compiled code in the specified directory. For example,

)library )dir /u/jones/as/quantum

You must give an explicit directory after )dir, even if you want all compiled code in the current working directory processed.

)library )dir .

You can compile category, domain, and package constructors contained in files with file extension .spad. You can compile individual constructors or every constructor in a file.

The full filename is remembered between invocations of this command and )edit commands. The sequence of commands

)compile matrix.spad
)edit
)compile

will call the compiler, edit, and then call the compiler again on the file matrix.spad. If you do not specify a directory, the working current directory (see description of command )cd ) is searched for the file. If the file is not found, the standard system directories are searched.

If you do not give any options, all constructors within a file are compiled. Each constructor should have an )abbreviation command in the file in which it is defined. We suggest that you place the )abbreviation commands at the top of the file in the order in which the constructors are defined. The list of
25.2. FUNCTIONS

commands serves as a table of contents for the file.

The )library option causes directories containing the compiled code for each constructor to be created in the working current directory. The name of such a directory consists of the constructor abbreviation and the .NRLIB file extension. For example, the directory containing the compiled code for the MATRIX constructor is called MATRIX.NRLIB. The )nolibrary option says that such files should not be created.

The )vartrace option causes the compiler to generate extra code for the constructor to support conditional tracing of variable assignments. (see description of command )trace). Without this option, this code is suppressed and one cannot use the )vars option for the trace command.

The )constructor option is used to specify a particular constructor to compile. All other constructors in the file are ignored. The constructor name or abbreviation follows )constructor. Thus either

)compile matrix.spad )constructor RectangularMatrix

or

)compile matrix.spad )constructor RMATRIX

compiles the RectangularMatrix constructor defined in matrix.spad.

The )break and )nobreak options determine what the compiler does when it encounters an error. )break is the default and it indicates that processing should stop at the first error. The value of the )set break variable then controls what happens.

Also See:
  o )abbreviation
  o )edit
  o )library

25.2 Functions

defvar $/editfile

— initvars —

1 “abbreviation” (?? p ??) “edit” (30.2 p 544) “library” (66.1 p 1013)
(defvar /editfile nil)
Chapter 26

)copyright help page Command

26.1 copyright help page man page

— copyright.help —

The term Axiom, in the field of computer algebra software, along with AXIOM and associated images are common-law trademarks. While the software license allows copies, the trademarks may only be used when referring to this project.

Axiom is distributed under terms of the Modified BSD license. Axiom was released under this license as of September 3, 2002. Source code is freely available at: http://savannah.nongnu.org/projects/axiom
Copyrights remain with the original copyright holders. Use of this material is by permission and/or license. Individual files contain reference to these applicable copyrights. The copyright and license statements are collected here for reference.

Portions Copyright (c) 2003- The Axiom Team

The Axiom Team is the collective name for the people who have contributed to this project. Where no other copyright statement is noted in a file this copyright will apply.

Portions Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

517
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of The Numerical ALgorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1989-95 GROUPE BULL

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL GROUPE BULL BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of GROUPE BULL shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from GROUPE BULL.

Portions Copyright (C) 2002, Codemist Ltd. All rights reserved.
acn@codemist.co.uk
CCL Public License 1.0
======================

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

(3) Neither the name of Codemist nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

(4) If you distribute a modified form or either source or binary code
(a) you must make the source form of these modification available to Codemist;
(b) you grant Codemist a royalty-free license to use, modify or redistribute your modifications without limitation;
(c) you represent that you are legally entitled to grant these rights and that you are not providing Codemist with any code that violates any law or breaches any contract.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1995-1997 Eric Young (eay@mincom.oz.au)
All rights reserved.

This package is an SSL implementation written by Eric Young (eay@mincom.oz.au).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are heared to. The following conditions
apply to all code found in this distribution, be it the RC4, RSA,
\lhash, DES, etc., code; not just the SSL code. The SSL documentation
included with this distribution is covered by the same copyright terms
except that the holder is Tim Hudson (tjh@mincom.oz.au).

Copyright remains Eric Young’s, and as such any Copyright notices in
the code are not to be removed.
If this package is used in a product, Eric Young should be given attribution
as the author of the parts of the library used.
This can be in the form of a textual message at program startup or
in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software
   must display the following acknowledgement:
   "This product includes cryptographic software written by
    Eric Young (eay@mincom.oz.au)"
The word 'cryptographic' can be left out if the routines from the library
being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from
   the apps directory (application code) you must include an acknowledgement:
   "This product includes software written by Tim Hudson (tjh@mincom.oz.au)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ‘‘AS IS’’ AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

The licence and distribution terms for any publically available version or
derivative of this code cannot be changed. i.e. this code cannot simply be
copied and put under another distribution licence
[including the GNU Public Licence.]

Portions Copyright (C) 1988 by Leslie Lamport.

Portions Copyright (c) 1998 Free Software Foundation, Inc.
Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, distribute with modifications, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE ABOVE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
 DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name(s) of the above copyright
holders shall not be used in advertising or otherwise to promote the
sale, use or other dealings in this Software without prior written
authorization.

Portions Copyright 1989-2000 by Norman Ramsey. All rights reserved.

Noweb is protected by copyright. It is not public-domain
software or shareware, and it is not protected by a "copyleft"
agreement like the one used by the Free Software Foundation.

Noweb is available free for any use in any field of endeavor. You may
redistribute noweb in whole or in part provided you acknowledge its
source and include this COPYRIGHT file. You may modify noweb and
create derived works, provided you retain this copyright notice, but
the result may not be called noweb without my written consent.

You may sell noweb if you wish. For example, you may sell a CD-ROM
including noweb.

You may sell a derived work, provided that all source code for your
derived work is available, at no additional charge, to anyone who buys
your derived work in any form. You must give permission for said
source code to be used and modified under the terms of this license.
You must state clearly that your work uses or is based on noweb and
that noweb is available free of charge. You must also request that
bug reports on your work be reported to you.

Portions Copyright (c) 1987 The RAND Corporation. All rights reserved.
CHAPTER 26. COPYRIGHT HELP PAGE COMMAND


All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Portions Copyright (c) Renaud Rioboo and the University Paris 6.

Portions Copyright (c) 2003-2010 Jocelyn Guidry

Portions Copyright (c) 2001-2010 Timothy Daly

26.2 Functions

defun copyright

[obey p??]
[concat p1047]
[getenviron p29]

— defun copyright —

(defun |copyright| ()
 (obey (concat "cat " (getenviron "AXIOM") "/doc/spadhelp/copyright.help")))

———
defun trademark

— defun trademark 0 —

(defun |trademark| ()
(format t "The term Axiom, in the field of computer algebra software, "~%")
(format t "along with AXIOM and associated images are common-law "~%")
(format t "trademarks. While the software license allows copies, the "~%")
(format t "trademarks may only be used when referring to this project "~%"))

This command is in the list of $noParseCommands 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2
Chapter 27

)credits help page Command

27.1 credits help page man page

27.2 Variables Used

27.3 Functions

defun credits

[credits p525]

— defun credits 0 —

(defun credits ()
  (declare (special credits))
  (mapcar #'(lambda (x) (princ x) (terpri)) creditlist))

— —

This command is in the list of $noParseCommands 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2
Chapter 28

)describe help page Command

28.1 describe help page man page

— describe.help —

====================================================================
)describe
====================================================================

User Level Required: interpreter

Command Syntax:

- )describe categoryName
- )describe domainName
- )describe packageName

Command Description:

This command is used to display the comments for the operation, category, domain or package. The comments are part of the algebra source code.

The commands

)describe <categoryName> [internal]
)describe <domainName> [internal]
)describe <packageName> [internal]

will show a properly formatted version of the "Description:" keyword from the comments in the algebra source for the category, domain, or package requested.
If 'internal' is requested, then the internal format of the domain or package is described. Categories do not have an internal representation.

---

defvar $describeOptions

The current value of $describeOptions is

---

initvars

---

(defvar $describeOptions '(|category| |domain| |package|))

---

28.2 Functions

defun Print comment strings from algebra libraries

This trivial function satisfies the standard pattern of making a user command match the name of the function which implements the command. That command immediately invokes a “Spad2Cmd” version. [describespad2cmd p?]

---

defun describe

---

(defun |describe| (l)
(describeSpad2Cmd l))

---

defun describeSpad2Cmd

The describe command prints cleaned-up comment strings from the algebra libraries. It can print strings associated with a category, domain, package, or by operation. This implements command line options of the form:

)describe categoryName [internal]
)describe domainName [internal]
)describe packageName [internal]

The describeInternal function will either call the “dc” function to describe the internal representation of the argument or it will print a cleaned up version of the text for the
"Description" keyword in the Category, Domain, or Package source code.  

(defun describeSpad2Cmd (l)
  (labels ((fullname (arg)
             "Convert abbreviations to the full constructor name"
             (let ((abb (getdatabase arg 'abbreviation)))
               (if abb arg (getdatabase arg 'constructor)))))
    (describeInternal (cdp internal?)
      (if internal?
          (progn
            (unless (eq (getdatabase cdp 'constructorkind) '|category|) (|dc| cdp))
            (showdatabase cdp)
            (mapcar #'(lambda (x) (if (stringp x) (cleanline x)))
              (flatten (car (getdatabase (fullname cdp) 'documentation))))))
          (let ((|$e| |$EmptyEnvironment|) (opt (second l)))
            (declare (special |$e| |$EmptyEnvironment| $describeOptions))
            (if (and (consp l) (not (eq opt '?)))
              (describeInternal (first l) (second l))
              (|sayMessage|
                (append
                  '(" describe keyword arguments are")
                  (mapcar #'(lambda (x) (format nil "" a" a x)) $describeOptions)
                  (format nil "" or abbreviations thereof"))))))
  (let ((|$e| |$EmptyEnvironment|) (opt (second l)))
    (declare (special |$e| |$EmptyEnvironment| $describeOptions))
    (if (and (consp l) (not (eq opt '?)))
      (describeInternal (first l) (second l))
      (|sayMessage|
        (append
          '(" describe keyword arguments are")
          (mapcar #'(lambda (x) (format nil "" a" a x)) $describeOptions)
          (format nil "" or abbreviations thereof"))))))

(defun cleanline (line)
  (labels ((replaceInLine (thing other line)
              (do ((mark (search thing line) (search thing line))
                   (null mark) line)

---

---

---
(setq line
  (concatenate 'string (subseq line 0 mark) other
  (subseq line (+ mark (length thing)))))

(removeFromLine (thing line) (replaceInLine thing "" line))

(removeKeyword (str line)
  (do ((mark (search str line) (search str line)))
      ((null mark) line)
    (let (left point mid right)
      (setq left (subseq line 0 mark))
      (setq point (search "}" line :start2 mark))
      (setq mid (subseq line (+ mark (length str)) point))
      (setq right (subseq line (+ point 1))))
    (setq line (concatenate 'string left mid right))))

(addSpaces (str line)
  (do ((mark (search str line) (search str line)) (cnt))
      ((null mark) line)
    (let (left point mid right)
      (setq left (subseq line 0 mark))
      (setq point (search "}" line :start2 mark))
      (setq mid (subseq line (+ mark (length str)) point))
      (if (setq cnt (parse-integer mid :junk-allowed t))
        (setq mid (make-string cnt :initial-element #\ ))
        (setq mid "")
      )
      (setq right (subseq line (+ point 1))))
    (setq line (concatenate 'string left mid right))))

(splitAtNewline (line)
  (do ((mark (search "~%" line) (search "~%" line)) (lines))
      ((null mark)
        (push " " lines)
        (push line lines)
      (nreverse lines))
    (push (subseq line 0 mark) lines)
    (setq line (subseq line (+ mark 2)))))

(wrapOneLine (line margin result)
  (if (null line)
    (nreverse result)
  (if (< (length line) margin)
    (wrapOneLine nil margin (append (list line) result))
  (let (oneline spill aspace)
    (setq aspace (position #\space (subseq line 0 margin) :from-end t))
    (setq oneline (string-trim '(#\space) (subseq line 0 aspace)))
    (setq spill (string-trim '(#\space) (subseq line aspace)))
    (wrapOneLine spill margin (append (list oneline) result))))))

(reflowParagraph (line)
28.2. FUNCTIONS

(defun flatten (x)
(labels (
  (rec (x acc)
    (cond
      ((null x) acc)
      ((atom x) (cons x acc))
      (t (rec (car x) (rec (cdr x) acc)))))
  (rec x nil)))

---

defun flatten

--- defun flatten 0 ---

(defun flatten (x)
(labels (
  (rec (x acc)
    (cond
      ((null x) acc)
      ((atom x) (cons x acc))
      (t (rec (car x) (rec (cdr x) acc)))))
  (rec x nil)))

---
Chapter 29

)display help page Command

29.1 display help page man page

— display.help —

====================================================================
A.8. )display
====================================================================

User Level Required: interpreter

Command Syntax:

- )display all
- )display properties
- )display properties all
- )display properties [obj1 [obj2 ...] ]
- )display value all
- )display value [obj1 [obj2 ...] ]
- )display mode all
- )display mode [obj1 [obj2 ...] ]
- )display names
- )display operations opName

Command Description:

This command is used to display the contents of the workspace and signatures of functions with a given name. (A signature gives the argument and return types of a function.)

The command
lists the names of all user-defined objects in the workspace. This is useful if you do not wish to see everything about the objects and need only be reminded of their names.

The commands

)display all
)display properties
)display properties all

all do the same thing: show the values and types and declared modes of all variables in the workspace. If you have defined functions, their signatures and definitions will also be displayed.

To show all information about a particular variable or user functions, for example, something named d, issue

)display properties d

To just show the value (and the type) of d, issue

)display value d

To just show the declared mode of d, issue

)display mode d

All modemaps for a given operation may be displayed by using )display operations. A modemap is a collection of information about a particular reference to an operation. This includes the types of the arguments and the return value, the location of the implementation and any conditions on the types. The modemap may contain patterns. The following displays the modemaps for the operation FromcomplexComplexCategory:

) d op complex

Also See:
- ) clear
- ) history
- ) set
- ) show
- ) what

---

1 “clear” (23.3 p 499) “history” (34.4 p 582) “set” (45.36 p 808) “show” (46.1 p 814) “what” (54.1 p 939)
29.2. FUNCTIONS

defvar $displayOptions

The current value of $displayOptions is
   — initvars —
   (defvar $displayOptions
     '(|abbreviations| |all| |macros| |modes| |names| |operations|
      |properties| |types| |values|))

29.2 Functions

defun display

This trivial function satisfies the standard pattern of making a user command match the
name of the function which implements the command. That command immediatly invokes
a “Spad2Cmd” version. [displayspad2cmd p??]

   — defun display —

   (defun |display| (l)
     (displaySpad2Cmd l))

displaySpad2Cmd

We process the options to the command and call the appropriate display function. There
are really only 4 display functions. All of the other options are just subcases.

There is a slight mismatch between the $displayOptions list of symbols and the options this
command accepts so we have a cond branch to clean up the option variable. This allows for
the options to be plural.

If we fall all the way thru we use the $displayOptions list to construct a list of strings for
the sayMessage function and tell the user what options are available. [abbQuery p536]
   [opOf p??]
   [listConstructorAbbreviations p484]
   [displayOperations p537]
   [displayMacros p538]
   [displayWorkspaceNames p454]
   [displayProperties p461]
   [selectOptionLC p479]
(defun displaySpad2Cmd (l)
  (let ((|$e| |$EmptyEnvironment|) (opt (car l)) (vl (cdr l)) option)
    (declare (special |$e| |$EmptyEnvironment| |$displayOptions|))
    (if (and (consp l) (not (eq opt '?)))
      (progn
        (setq option (|selectOptionLC| opt |$displayOptions| '|optionError|))
        (cond
          ((eq option '|all|)
           (setq l (list '|properties|)))
          (setq option '|properties|))
          ((or (eq option '|modes|) (eq option '|types|))
           (setq l (cons '|type| vl)))
          (setq option '|type|)))
          ((eq option '|values|)
           (setq l (cons '|value| vl)))
          (setq option '|value|))))
    (cond
      ((eq option '|abbreviations|)
       (if (null vl)
         (|listConstructorAbbreviations|)
         (dolist (v vl) (|abbQuery| (|opOf| v)))))
      ((eq option '|operations|) (|displayOperations| vl))
      ((eq option '|macros|) (|displayMacros| vl))
      ((eq option '|names|) (|displayWorkspaceNames|))
      (t (|displayProperties| option l))))
    (|sayMessage|)
    (append
     '(" display keyword arguments are")
     (mapcar #'(lambda (x) (format nil "~% ~a" x)) |$displayOptions|)
     (format nil "~% or abbreviations thereof"))))
  )

---

defun abbQuery

[getdatabase p1010]
[sayKeyedMsg p329]

---

---

---
29.2. FUNCTIONS

(let (abb)
  (cond
    ((setq abb (getdatabase x 'abbreviation))
     (|sayKeyedMsg| 's2iz0001 (list abb (getdatabase x 'constructorkind) x)))
    ((setq abb (getdatabase x 'constructor))
     (|sayKeyedMsg| 's2iz0001 (list x (getdatabase abb 'constructorkind) abb)))
    (t
     (|sayKeyedMsg| 's2iz0003 (list x)))))

defun displayOperations

This function takes a list of operation names. If the list is null we query the user to see if they want all operations printed. Otherwise we print the information for the requested symbols. [reportOpSymbol p??] [yesanswer p537] [sayKeyedMsg p329]

— defun displayOperations —

(defun displayOperations (l)
  (if l
      (dolist (op l) (|reportOpSymbol| op))
      (if (yesanswer)
          (dolist (op (|allOperations|)) (|reportOpSymbol| op))
          (|sayKeyedMsg| 's2iz0059 nil))))

— defun yesanswer —

(defun yesanswer ()
  (member
    (string2id-n (upcase (|queryUserKeyedMsg| 's2iz0058 nil)) 1) '(y yes)))
defun displayMacros

(getInterpMacroNames p)
(getParserMacroNames p)
(remdup p)
(sayBrightly p)
(member p)
(displayParserMacro p)
(seq p)
(exit p)
(displayMacro p)

— defun displayMacros —

(defun displayMacros (names)
  (let (imacs pmacs macros first)
    (setq imacs (getInterpMacroNames))
    (setq pmacs (getParserMacroNames))
    (if names
      (setq macros names)
      (setq macros (append imacs pmacs)))
    (setq macros (remdup macros))
    (cond
      ((null macros) (sayBrightly " There are no Axiom macros."))
      (t
       (setq first t)
       (do ((t0 macros (cdr t0)) (macro nil))
           ((or (atom t0) (progn (setq macro (car t0)) nil)) nil)
        (seq
         (exit
          (cond
           ((member macro pmacs)
            (cond
             (first (sayBrightly
               (cons '|%l| (cons "User-defined macros:" nil)))
               (setq first nil)))
             (displayParserMacro macro))
           (member macro imacs) 'iterate)
           (t (sayBrightly)
             (cons " "
               (cons '|%b|
                 (cons macro
                   (cons '|%d| (cons " is not a known Axiom macro." nil))))))))))))
        (setq first t)
        (do ((t1 macros (cdr t1)) (macro nil))
            ((or (atom t1) (progn (setq macro (car t1)) nil)) nil)
          (seq
            (expr "")
            (cons (cons (cons (car macro) (cons (case
              (member macro imacs)
              (iterate) 
              (member macro pmacs)
              (iterate) 
              (t (sayBrightly)
                (cons " "
                  (cons '|%b|
                    (cons macro
                      (cons '|%d| (cons " is not a known Axiom macro." nil))))))))))))
            (setq first t)
            (do ((t2 macros (cdr t2)) (macro nil))
                ((or (atom t2) (progn (setq macro (car t2)) nil)) nil)
              (seq
                (expr "")
                (cons (cons (car macro) (cons (case
                  (member macro imacs)
                  (iterate) 
                  (member macro pmacs)
                  (iterate) 
                  (t (sayBrightly)
                    (cons " "
                      (cons '|b|
                        (cons (case
                          (member macro imacs)
                          (iterate) 
                          (member macro pmacs)
                          (iterate) 
                          (t (sayBrightly)
                            (cons " "
                              (cons '|d|
                                (cons (case
                                  (member macro imacs)
                                  (iterate) 
                                  (member macro pmacs)
                                  (iterate) 
                                  (t (sayBrightly)
                                    (cons " "
                                      (cons '|d|
                                        (cons (case
                                          (member macro imacs)
                                          (iterate) 
                                          (member macro pmacs)
                                          (iterate) 
                                          (t (sayBrightly)
                                            (cons " "
                                              (cons '|d|
                                                (cons (case
                                                  (member macro imacs)
                                                  (iterate) 
                                                  (member macro pmacs)
                                                  (iterate) 
                                                  (t (sayBrightly)
                                                    (cons " "
                                                      (cons '|d|
                                                        (cons (case
                                                          (member macro imacs)
                                                          (iterate) 
                                                          (member macro pmacs)
                                                          (iterate) 
                                                          (t (sayBrightly)
                                                            (cons " "
                                                              (cons '|d|
                                                                (cons (case
                                                                  (member macro imacs)
                                                                  (iterate) 
                                                                  (member macro pmacs)
                                                                  (iterate) 
                                                                  (t (sayBrightly)
                                                                    (cons " "
                                                                      (cons '|d|
                                                                        (cons (case
                                                                          (member macro imacs)
                                                                          (iterate) 
                                                                          (member macro pmacs)
                                                                          (iterate) 
                                                                          (t (sayBrightly)
                                                                            (cons " "
                                                                              (cons '|d|
                                                                                (cons (case
                                                                                  (member macro imacs)
                                                                                  (iterate) 
                                                                                  (member macro pmacs)
                                                                                  (iterate) 
                                                                                  (t (sayBrightly)
                                                                                    (cons " "
                                                                                      (cons '|d|
                                                                                        (cons (case
                                                                                          (member macro imacs)
                                                                                          (iterate) 
                                                                                          (member macro pmacs)
                                                                                          (iterate) 
                                                                                          (t (sayBrightly)
                                                                                            (cons " "
                                                                                              (cons '|d|
                                                                                               (cons (case
                                                                                                 (member macro imacs)
                                                                                                 (iterate) 
                                                                                                 (member macro pmacs)
                                                                                                 (iterate) 
                                                                                                 (t (sayBrightly)
                                                                                                   (cons " "
                                                                                                     (cons '|d|
                                                                                                       (cons (case
                                                                                                         (member macro imacs)
                                                                                                         (iterate) 
                                                                                                         (member macro pmacs)
                                                                                                         (iterate) 
                                                                                                         (t (sayBrightly)
                                                                                                           (cons " "
                                                                                                              (cons '|d|
                                                                                                                (cons (case
                                                                                                                  (member macro imacs)
                                                                                                                  (iterate) 
                                                                                                                  (member macro pmacs)
                                                                                                                  (iterate) 
                                                                                                                  (t (sayBrightly)
                                                                                                                    (cons " "
                                                                                                                      (cons '|d|
                                                                                                                       (cons (case
                                                                                                                         (member macro imacs)
                                                                                                                         (iterate) 
                                                                                                                         (member macro pmacs)
                                                                                                                         (iterate) 
                                                                                                                         (t (sayBrightly)
                                                                                                                          (cons " "
                                                                                                                              (cons '|d|
                                                                                                                               (cons (case
                                                                                                                                  (member macro imacs)
                                                                                                                                  (iterate) 
                                                                                                                                  (member macro pmacs)
                                                                                                                                  (iterate) 
                                                                                                                                  (t (sayBrightly)
                                                                                                                                    (cons " "
                                                                                                                                      (cons '|d|
                                                                                                                                       (cons (case
                                                                                                                                         (member macro imacs)
                                                                                                                                         (iterate) 
                                                                                                                                         (member macro pmacs)
                                                                                                                                         (iterate) 
                                                                                                                                         (t (sayBrightly)
                                                                                                                                         (cons " "
                                                                                                                                            (cons '|d|
                                                                                                                                             (cons (case
                                                                                                                                                (member macro imacs)
                                                                                                                                                (iterate) 
                                                                                                                                                (member macro pmacs)
                                                                                                                                                (iterate) 
                                                                                                                                                (t (sayBrightly)
                                                                                                                                                (cons " "
                                                                                                                                                    (cons '|d|
                                                                                                                                                        (cons (case
                                                                                                                                                           (member macro imacs)
                                                                                                                                                           (iterate) 
                                                                                                                                                           (member macro pmacs)
                                                                                                                                                           (iterate) 
                                                                                                                                                           (t (sayBrightly)
                                                                                                                                                           (cons " "
                                                                                                                                                               (cons '|d|
                                                                                                                                                                 (cons (case
                                                                                                                                                                     (member macro imacs)
                                                                                                                                                                     (iterate) 
                                                                                                                                                                     (member macro pmacs)
                                                                                                                                                                     (iterate) 
                                                                                                                                                                     (t (sayBrightly)
                                                                                                                                                                     (cons " "
                                                                                                                                                                             (cons '|d|
                                                                                                                                                                               (cons (case
                                                                                                                                                                                   (member macro imacs)
                                                                                                                                                                                   (iterate) 
                                                                                                                                                                                   (member macro pmacs)
                                                                                                                                                                                   (iterate) 
                                                                                                                                                                                   (t (sayBrightly)
                                                                                                                                                                                   (cons " "
                                                                                                                                                                                        (cons '|d|
                                                                                                                                                                                            (cons (case
                                                                                                                                                                                                (member macro imacs)
                                                                                                                                                                                                (iterate) 
                                                                                                                                                                                                (member macro pmacs)
                                                                                                                                                                                                (iterate) 
                                                                                                                                                                                                (t (sayBrightly)
                                                                                                                                                                                                (cons " "
                                                                                                                                                                                                    (cons '|d|
                                                                                                                                                                                                      (cons (case
                                                                                                                                                                                                        (member macro imacs)
                                                                                                                                                                                                        (iterate) 
                                                                                                                                                (member macro pmacs)
                                                                                                                                                (iterate) 
                                                                                (t (sayBrightly)
                                                                                    (cons " "
                                                                                                      (cons '|d|
                                                                                                       (cons (case
                                                                                                          (member macro imacs)
                                                                                                          (iterate) 
                                                                                                          (member macro pmacs)
                                                                                                          (iterate) 
                                          (t (sayBrightly)
                                            (cons " "
                                              (cons '|b|
                                                (cons macro
                                                  (cons '|d| (cons " is not a known Axiom macro." nil)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
(exit
  (cond
   ((|member| macro imacs)
    (cond
     ((|member| macro pmacs) '|iterate|)
      (t
       (cond
        (first
         (|sayBrightly|
          (cons '|%l|
            (cons "System-defined macros:" nil))) (setq first nil)))
        (|displayMacro| macro))))
    ((|member| macro pmacs) '|iterate|))))
nil))))

defun sayExample

This function expects 2 arguments, the documentation string and the name of the operation. It searches the documentation string for ++X lines. These lines are examples lines for functions. They look like ordinary ++ comments and t into the ordinary comment blocks. So, for example, in the plot.spad.pamphlet file we find the following function signature:

```
plot: (F -> F,R) -> %
   ++ plot(f,a..b) plots the function \spad{f(x)}
   ++ on the interval \spad{[a,b]}.
   ++
   ++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
   ++X plot(fp,-1.0..1.0)$PLOT
```

This function splits out and prints the lines that begin with ++X.

A minor complication of printing the examples is that the lines have been processed into internal compiler format. Thus the lines that read:

```
++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
++X plot(fp,-1.0..1.0)$PLOT
```

are actually stored as one long line containing the example lines

```
"\indented{1}{plot(\spad{f},\{a..\spad{b}\}) plots the function \spad{f(x)}
\spad{f(x)}) \indented{1}{on the interval \spad{[a,\{b\}]}.)
\blankline
\spad{X} fp:=(t:DFLOAT):DFLOAT +-> sin(\spad{t})
\spad{X} plot(\spad{fp},\{\spad{-1}.0..1.0}\$PLOT"
```
So when we have an example line starting with \texttt{++X}, it gets converted to the compiler to \texttt{\spad{X}}. So each example line is delimited by \texttt{\spad{X}}.

The compiler also removes the newlines so if there is a subsequent \texttt{\spad{X}} in the docstring then it implies multiple example lines and we loop over them, splitting them up at the delimiter.

If there is only one then we clean it up and print it. 

\begin{verbatim}
| defun sayExample |
| (defun sayExample (docstring)
| (let (line point)
| (when (setq point (search "\spad{X}" docstring))
| (setq line (subseq docstring (+ point 8)))
| (do ((mark (search "\spad{X}" line) (search "\spad{X}" line)))
| ((null mark))
| (princ (cleanupLine (subseq line 0 mark))))
| (|sayNewLine|)
| (setq line (subseq line (+ mark 8))))
| (princ (cleanupLine line))
| (|sayNewLine|)
| (|sayNewLine|)))
\end{verbatim}

---

defun cleanupLine

This function expects example lines in internal format that has been partially processed to remove the prefix. Thus we get lines that look like:

\begin{verbatim}
fp:=(t:DFLOAT):DFLOAT +-> sin(\spad{t})
plot(\spad{fp},{}\spad{-1}.0..1.0)\$PLOT
\end{verbatim}

It removes all instances of \texttt{\{} and \texttt{\}} and unwraps the \texttt{\spad{}} call, leaving only the argument. We return lines that look like:

\begin{verbatim}
fp:=(t:DFLOAT):DFLOAT +-> sin(t)
plot(fp,-1.0..1.0)$PLOT
\end{verbatim}

which is hopefully exactly what the user wrote.

The compiler inserts \texttt{\{} as a space so we remove it. We remove all of the \texttt{\}} characters. We remove all of the \texttt{\spad{\ldots}} delimiters which will occur around other \texttt{\spad{}} variables. Technically we should search recursively for the matching delimiter rather than the next brace but the problem does not arise in practice.

---

defun cleanupLine 0 ---
(defun cleanupLine (line)
  (do ((mark (search "{}" line) (search "{" line)))
     ((null mark))
    (setq line
       (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 2)))))
  (do ((mark (search "\"" line) (search "\\" line)))
     ((null mark))
    (setq line
       (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 1)))))
  (do ((mark (search "spad{" line) (search "spad{" line)))
     ((null mark))
    (let (left point mid right)
      (setq left (subseq line 0 mark))
      (setq point (search "}" line :start mark))
      (setq mid (subseq line (+ mark 5) point))
      (setq right (subseq line (+ point 1))))
    (setq line (concatenate 'string left mid right))))
line)
Chapter 30

)edit help page Command

30.1 edit help page man page

— edit.help —

====================================================================
A.9. )edit
====================================================================

User Level Required: interpreter

Command Syntax:

- )edit [filename]

Command Description:

This command is used to edit files. It works in conjunction with the )read and )compile commands to remember the name of the file on which you are working. By specifying the name fully, you can edit any file you wish. Thus

)edit /u/julius/matrix.input

will place you in an editor looking at the file /u/julius/matrix.input. By default, the editor is vi, but if you have an EDITOR shell environment variable defined, that editor will be used. When AXIOM is running under the X Window System, it will try to open a separate xterm running your editor if it thinks one is necessary. For example, under the Korn shell, if you issue

export EDITOR=emacs

then the emacs editor will be used by )edit.
If you do not specify a file name, the last file you edited, read or compiled will be used. If there is no ‘‘last file’’ you will be placed in the editor editing an empty unnamed file.

It is possible to use the )system command to edit a file directly. For example,

```plaintext
)system emacs /etc/rc.tcpip
```
calls emacs to edit the file.

Also See:
- )system
- )compile
- )read

---

### 30.2 Functions

**defun edit**

[editSpad2Cmd p544]

---

```plaintext
(defun |edit| (l) (|editSpad2Cmd| l))
```

---

**defun editSpad2Cmd**

[pathname p1042]
[pathnameDirectory p1041]
[pathnameType p1040]
[FINDFILE p??]
[pathnameName p1040]
[editFile p545]
[updateSourceFiles p546]
[/editfile p515]

---

1 “system” (?? p ??) “read” (42.1 p 642)
30.2. FUNCTIONS

— defun editSpad2Cmd —

(defun |editSpad2Cmd| (l)
(let (olddir filetypes ll rc)
(declare (special /editfile))
 (setq l (cond ((null l) /editfile) (t (car l))))
 (setq l (|pathname| l))
 (setq olddir (|pathnameDirectory| l))
 (setq filetypes
   (cond
    ((|pathnameType| l) (list (|pathnameType| l)))
    ((eq |$UserLevel| '|interpreter|) '("input" "INPUT" "spad" "SPAD"))
    ((eq |$UserLevel| '|compiler|) '("input" "INPUT" "spad" "SPAD"))
    (t '("input" "INPUT" "spad" "SPAD" "boot" "BOOT"
      "lisp" "LISP" "meta" "META")))
 (setq ll
   (cond
    ((string= olddir "")
     (|pathname| ($findfile (|pathnameName| l) filetypes)))
    (t l)))
 (setq l (|pathname| ll))
 (setq /editfile l)
 (setq rc (|editFile| l))
 (|updateSourceFiles| l)
 rc))

defun Implement the )edit command

[|strconc p??|]
[|namestring p1040|]
[|pathname p1042|]
[|obey p??|]

— defun editFile —

(defun |editFile| (file)
(cond
  ((member (intern "WIN32" (find-package 'keyword)) *features*)
   (obey (strconc "notepad " (|namestring| (|pathname| file)))))
  (t
   (obey
    (strconc "$AXIOM/lib/SPADEDIT " (|namestring| (|pathname| file)))))
  ))
defun updateSourceFiles

(defun updateSourceFiles (arg)
 (declare (special $sourceFiles))
 (setq arg (pathname arg))
 (setq arg (pathname (list (pathnameName arg) (pathnameType arg) "*")))
 (when (and (makeInputFilename arg)
               (member (pathnameTypeId arg) '(boot lisp meta)))
             (setq $sourceFiles (insert arg $sourceFiles))
 arg)

— defun updateSourceFiles —

(defun updateSourceFiles (arg)
 (declare (special $sourceFiles))
 (setq arg (pathname arg))
 (setq arg (pathname (list (pathnameName arg) (pathnameType arg) "*")))
 (when (and (makeInputFilename arg)
               (member (pathnameTypeId arg) '(boot lisp meta)))
             (setq $sourceFiles (insert arg $sourceFiles))
 arg)

——
Chapter 31

)fin help page Command

31.1 fin help page man page

— fin.help —

====================================================================
A.10. )fin
====================================================================

User Level Required: development

Command Syntax:

- )fin

Command Description:

This command is used by AXIOM developers to leave the AXIOM system and return to the underlying Lisp system. To return to AXIOM, issue the "'(spad)'" function call to Lisp.

Also See:
o )pquit
o )quit

\footnote{\textcopyright
d p 634) “quit” (d p 638)}
defun Exit from the interpreter to lisp

[spad-reader p??]
[eof p??]

— defun fin 0 —

(defun |fin| ()
  (setq *eof* t)
  (throw 'spad_reader nil))

31.2 Functions

This command is in the list of $noParseCommands$ 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2
Chapter 32

)frame help page Command

32.1 frame help page man page

--- frame.help ---

====================================================================
A.11. )frame
====================================================================

User Level Required: interpreter

Command Syntax:

- )frame new frameName
- )frame drop [frameName]
- )frame next
- )frame last
- )frame names
- )frame import frameName [objectName1 [objectName2 ...]]
- )set message frame on | off
- )set message prompt frame

Command Description:

A frame can be thought of as a logical session within the physical session that you get when you start the system. You can have as many frames as you want, within the limits of your computer’s storage, paging space, and so on. Each frame has its own step number, environment and history. You can have a variable named a in one frame and it will have nothing to do with anything that might be called a in any other frame.

Some frames are created by the HyperDoc program and these can have pretty
strange names, since they are generated automatically. To find out the names of all frames, issue

)frame names

It will indicate the name of the current frame.

You create a new frame “quark” by issuing

)frame new quark

The history facility can be turned on by issuing either )set history on or )history on. If the history facility is on and you are saving history information in a file rather than in the AXIOM environment then a history file with filename quark.axh will be created as you enter commands. If you wish to go back to what you were doing in the “initial” frame, use

)frame next

or

)frame last

to cycle through the ring of available frames to get back to “initial”.

If you want to throw away a frame (say “quark”), issue

)frame drop quark

If you omit the name, the current frame is dropped.

If you do use frames with the history facility on and writing to a file, you may want to delete some of the older history files. These are directories, so you may want to issue a command like rm -r quark.axh to the operating system.

You can bring things from another frame by using )frame import. For example, to bring the f and g from the frame “quark” to the current frame, issue

)frame import quark f g

If you want everything from the frame “quark”, issue

)frame import quark

You will be asked to verify that you really want everything.

There are two )set flags to make it easier to tell where you are.

)set message frame on | off
32.2 VARIABLES USED

will print more messages about frames when it is set on. By default, it is off.

)set message prompt frame

will give a prompt that looks like

initial (1) ->

when you start up. In this case, the frame name and step make up the prompt.

Also See:
  o )history
  o )set

32.2 Variables Used

The frame mechanism uses several dollar variables.

Primary variables

Primary variables are those which exist solely to make the frame mechanism work.

The $interpreterFrameName contains a symbol which is the name of the current frame in use.

The $interpreterFrameRing contains a list of all of the existing frames. The first frame on the list is the “current” frame. When AXIOMsys is started directly there is only one frame named “initial”.

If the system is started under sman (using the axiom shell script, for example), there are two frames, “initial” and “frame0”. In this case, “frame0” is the current frame. This can cause subtle problems because functions defined in the axiom initialization file (.axiom.input) will be defined in frame “initial” but the current frame will be “frame0”. They will appear to be undefined. However, if the user does ”)frame next” they can switch to the “initial” frame and see the functions correctly defined.

The $frameMessages variable controls when frame messages will be displayed. The variable is initially NIL. It can be set on (T) or off (NIL) using the system command:

)set message frame on | off

---

1 “history” (34.4 p 582) “set” (45.36 p 808)
Setting frame messages on will output a line detailing the current frame after every output is complete.

**Used variables**

The frame collects and uses a few top level variables. These are: $InteractiveFrame, $IOindex, $HiFiAccess, $HistList, $HistListLen, $HistListAct, $HistRecord, $internalHistoryTable, and $localExposureData.

These variables can also be changed by the frame mechanism when the user requests changing to a different frame.

### 32.3 Data Structures

**Frames and the Interpreter Frame Ring**

Axiom has the notion of “frames”. A frame is a data structure which holds all the vital data from an Axiom session. There can be multiple frames and these live in a top-level variable called $interpreterFrameRing. This variable holds a circular list of frames. The parts of a frame and their initial, default values are:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$interpreterFrameName</td>
<td>a string, named on creation</td>
</tr>
<tr>
<td>$InteractiveFrame</td>
<td>(list (list nil))</td>
</tr>
<tr>
<td>$IOindex</td>
<td>an integer, 1</td>
</tr>
<tr>
<td>$HiFiAccess</td>
<td>$HiFiAccess, see the variable description</td>
</tr>
<tr>
<td>$HistList</td>
<td>$HistList, see the variable description</td>
</tr>
<tr>
<td>$HistListLen</td>
<td>$HistListLen, see the variable description</td>
</tr>
<tr>
<td>$HistListAct</td>
<td>$HistListAct, see the variable description</td>
</tr>
<tr>
<td>$HistRecord</td>
<td>$HistRecord, see the variable description</td>
</tr>
<tr>
<td>$internalHistoryTable</td>
<td>nil</td>
</tr>
<tr>
<td>$localExposureData</td>
<td>a copy of $localExposureData</td>
</tr>
</tbody>
</table>

### 32.4 Accessor Functions

These could be macros but we wish to export them to the API code in the algebra so we keep them as functions.

**0th Frame Component – frameName**

```lisp
defun frameName
  ; defun frameName 0
```

---
(defun frameName (frame)
  (car frame))

1st Frame Component – frameInteractive

— defun frameInteractive 0 —

(defun frameInteractive (frame)
  (nth 1 frame))

2nd Frame Component – frameIOIndex

— defun frameIOIndex 0 —

(defun frameIOIndex (frame)
  (nth 2 frame))

3rd Frame Component – frameHiFiAccess

— defun frameHiFiAccess 0 —

(defun frameHiFiAccess (frame)
  (nth 3 frame))

4th Frame Component – frameHistList

— defun frameHistList 0 —
(defun frameHistList (frame)
  (nth 4 frame))

5th Frame Component – frameHistListLen

— defun frameHistListLen 0 —

(defun frameHistListLen (frame)
  (nth 5 frame))

6th Frame Component – frameHistListAct

— defun frameHistListAct 0 —

(defun frameHistListAct (frame)
  (nth 6 frame))

7th Frame Component – frameHistRecord

— defun frameHistRecord 0 —

(defun frameHistRecord (frame)
  (nth 7 frame))

8th Frame Component – frameHistoryTable

— defun frameHistoryTable 0 —
32.5. FUNCTIONS

32.5.1 Functions

Initializing the Interpreter Frame Ring

Now that we know what a frame looks like we need a function to initialize the list of frames. This function sets the initial frame name to “initial” and creates a list of frames containing an empty frame. This list is the interpreter frame ring and is not actually circular but is managed as a circular list.

As a final step we update the world from this frame. This has the side-effect of resetting all the important global variables to their initial values.

```
(defun initializeInterpreterFrameRing ()
  "Initializing the Interpreter Frame Ring"
  (declare (special $interpreterFrameName $interpreterFrameRing))
  (setq $interpreterFrameName 'initial)
  (setq $interpreterFrameRing
    (list (emptyInterpreterFrame $interpreterFrameName)
      (updateFromCurrentInterpreterFrame $interpreterFrameName)))
  nil)
```

---

9th Frame Component – frameExposureData

```
(defun frameExposureData 0)
  (nth 9 frame))
```

---

32.5 Functions

Initializing the Interpreter Frame Ring

Now that we know what a frame looks like we need a function to initialize the list of frames. This function sets the initial frame name to “initial” and creates a list of frames containing an empty frame. This list is the interpreter frame ring and is not actually circular but is managed as a circular list.

As a final step we update the world from this frame. This has the side-effect of resetting all the important global variables to their initial values.

```
(defun initializeInterpreterFrameRing ()
  "Initializing the Interpreter Frame Ring"
  (declare (special $interpreterFrameName $interpreterFrameRing))
  (setq $interpreterFrameName 'initial)
  (setq $interpreterFrameRing
    (list (emptyInterpreterFrame $interpreterFrameName)
      (updateFromCurrentInterpreterFrame $interpreterFrameName)))
  nil)
```

---
CHAPTER 32.  )FRAME HELP PAGE COMMAND

Creating a List of all of the Frame Names

This function simply walks across the frame in the frame ring and returns a list of the name of each frame. [interpreterFrameRing p??]

— defun frameNames 0 —

(defun |frameNames| ()
  "Creating a List of all of the Frame Names"
  (declare (special $interpreterFrameRing))
  (mapcar #'frameName $interpreterFrameRing))

——

Get Named Frame Environment (aka Interactive)

If the frame is found we return the environment portion of the frame otherwise we construct an empty environment and return it. The initial values of an empty frame are created here. This function returns a single frame that will be placed in the frame ring. [frameInteractive p??]

— defun frameEnvironment —

(defun |frameEnvironment| (fname)
  "Get Named Frame Environment (aka Interactive)"
  (let ((frame (|findFrameInRing| fname)))
    (if frame
      (frameInteractive frame)
      (list (list nil))))))

——

Create a new, empty Interpreter Frame

[HiFiAccess p733]
[HistList p??]
[HistListLen p??]
[HistListAct p??]
[HistRecord p??]
[localExposureDataDefault p696]

— defun emptyInterpreterFrame 0 —

(defun |emptyInterpreterFrame| (name)
"Create a new, empty Interpreter Frame"
(declare (special |$HiFiAccess| |$HistList| |$HistListLen| |$HistListAct|
 |$HistRecord| |$localExposureDataDefault|))
(list name ; frame name
 (list (list nil)) ; environment
 1 ; $IOindex
|$HiFiAccess|
|$HistList|
|$HistListLen|
|$HistListAct|
|$HistRecord|
nil ; $internalHistoryTable
(copy-seq |$localExposureDataDefault|)); |$localExposureData|

Collecting up the Environment into a Frame

We can collect up all the current environment information into one frame element with this
call. It creates a list of the current values of the global variables and returns this as a frame
element.

[$interpreterFrameName p??]
[$InteractiveFrame p??]
[$IOindex p10]
[$HiFiAccess p733]
[$HistList p??]
[$HistListLen p??]
[$HistListAct p??]
[$HistRecord p??]
[$internalHistoryTable p??]
[$localExposureData p697]

— defun createCurrentInterpreterFrame 0 —

(defun |createCurrentInterpreterFrame| ()
 "Collecting up the Environment into a Frame"
(declare (special |$interpreterFrameName| |$InteractiveFrame| |$IOindex|
 |$HiFiAccess| |$HistList| |$HistListLen| |$HistListAct| |$HistRecord|
 |$internalHistoryTable| |$localExposureData|))
(list
 |$interpreterFrameName|
 |$InteractiveFrame|
 |$IOindex|
 |$HiFiAccess|
 |$HistList|
 |$HistListLen|
Update from the Current Frame

The frames are kept on a circular list. The first element on that list is known as “the current frame”. This will initialize all of the interesting interpreter data structures from that frame.

(defun updateFromCurrentInterpreterFrame ()
"Update from the Current Frame"
(let (tmp1)
  (declare (special $interpreterFrameRing $interpreterFrameName $InteractiveFrame $IOindex $HiFiAccess $HistList $HistListLen $HistListAct $HistRecord $internalHistoryTable $localExposureData $frameMessages))
  (setq tmp1 (first $interpreterFrameRing)))

|sayMessage| (" Current interpreter frame is called", #(|bright| |$interpreterFrameName|)))

---

**Find a Frame in the Frame Ring by Name**

Each frame contains its name as the 0th element. We simply walk all the frames and if we find one we return it. [boot-equal p??]
[frameName p552]
[$interpreterFrameRing p??]

---

<table>
<thead>
<tr>
<th>defun findFrameInRing 0</th>
</tr>
</thead>
</table>

(defun |findFrameInRing| (name) "Find a Frame in the Frame Ring by Name"
(let (result)
  (declare (special |$interpreterFrameRing|))
  (dolist (frame |$interpreterFrameRing|)
    (when (boot-equal (frameName frame) name)
      (setq result frame)))
  result))

---

**Update the Current Interpreter Frame**

This function collects the normal contents of the world into a frame object, places it first on the frame list, and then sets the current values of the world from the frame object. [createCurrentInterpreterFrame p557]
[updateFromCurrentInterpreterFrame p558]
[$interpreterFrameRing p??]

---

<table>
<thead>
<tr>
<th>defun updateCurrentInterpreterFrame</th>
</tr>
</thead>
</table>

(defun |updateCurrentInterpreterFrame| () "Update the Current Interpreter Frame"
  (declare (special |$interpreterFrameRing|))
  (rplaca |$interpreterFrameRing| ![createCurrentInterpreterFrame!!])
  ![updateFromCurrentInterpreterFrame!!])

---
Move to the next Interpreter Frame in Ring

This function updates the current frame to make sure all of the current information is recorded. If there are more frame elements in the list then this will destructively move the current frame to the end of the list, that is, assume the frame list reads (1 2 3) this function will destructively change it to (2 3 1). [updateFromCurrentInterpreterFrame p558]

[interpreterFrameRing p??]

---

(defun nextInterpreterFrame

(defun nextInterpreterFrame ()
 "Move to the next Interpreter Frame in Ring"
 (declare (special $interpreterFrameRing))
 (when (cdr $interpreterFrameRing)
   (setq $interpreterFrameRing
     (nconc (cdr $interpreterFrameRing) (list (car $interpreterFrameRing))))
   (updateFromCurrentInterpreterFrame)))

---

Change to the Named Interpreter Frame

[updateCurrentInterpreterFrame p559]
[findFrameInRing p559]
[nremove p??]
[updateFromCurrentInterpreterFrame p558]
[interpreterFrameRing p??]

---

defun changeToNamedInterpreterFrame

(defun changeToNamedInterpreterFrame (name)
 "Change to the Named Interpreter Frame"
 (let (frame)
   (declare (special $interpreterFrameRing))
   (updateCurrentInterpreterFrame)
   (setq frame (findFrameInRing name))
   (when frame
     (setq $interpreterFrameRing
       (cons frame (nremove $interpreterFrameRing frame)))
     (updateFromCurrentInterpreterFrame))))
32.5. FUNCTIONS

Move to the previous Interpreter Frame in Ring

(defun previousInterpreterFrame ()
  "Move to the previous Interpreter Frame in Ring"
  (let ((tmp1 l b)
         (declare (special $interpreterFrameRing))
         (updateCurrentInterpreterFrame))
    (when (cdr $interpreterFrameRing)
      (setq tmp1 (reverse $interpreterFrameRing))
      (setq l (car tmp1))
      (setq b (nreverse (cdr tmp1)))
      (setq $interpreterFrameRing (nconc (cons l nil) b))
      (updateFromCurrentInterpreterFrame)))

Add a New Interpreter Frame

(defun addNewInterpreterFrame (name)
  "Add a New interpreter Frame"
  (declare (special $interpreterFrameRing))
  (if (null name)
      (throwKeyedMsg 's2iz0018 nil) ; you must provide a name for new frame
      (progn
        (updateCurrentInterpreterFrame)
        (dolist (f $interpreterFrameRing)
          (when (boot-equal name (frameName f)) ; existing frame with same name
Close an Interpreter Frame

[framename p??]
[throwKeyedMsg p??]
[$erase p??]
[makeHistFileName p579]
[updateFromCurrentInterpreterFrame p558]
[$interpreterFrameRing p??]
[$interpreterFrameName p??]

— defun closeInterpreterFrame —

(defun closeInterpreterFrame (name)
  "Close an Interpreter Frame"
  (declare (special $interpreterFrameRing|$interpreterFrameName|))
  (let ((ifr found)
         (if (null (cdr $interpreterFrameRing|)))
          (if (and name (not (equal name $interpreterFrameName|)))
              (|throwKeyedMsg| 's2iz0020 ; 1 frame left. not the correct name.
                 (cons $interpreterFrameName| nil))
              (|throwKeyedMsg| 's2iz0021 nil)) ; only 1 frame left, not closed
              (progn
                (if (null name)
                    (setq $interpreterFrameRing| (cdr $interpreterFrameRing|))
                    (progn
                      (setq found nil)
                      (setq ifr nil)
                      (dolist (f $interpreterFrameRing|)
                        (if (or found (not (equal name (frameName f))))
                            (setq ifr (cons f ifr)))
                        (setq found t))
                    (if (null found)
                        (|throwKeyedMsg| 's2iz0022 (cons name nil))
                        (progn
                          ($erase (|makeHistFileName| name))
                          (setq $interpreterFrameRing| (nreverse ifr))))))
              (|updateFromCurrentInterpreterFrame|))))
)
Display the Frame Names

[明亮 p??]
[frameName p??]
[sayKeyedMsg p329]
[$interpreterFrameRing p??]

— defun displayFrameNames —

(defun displayFrameNames ()
"Display the Frame Names"
(declare (special $interpreterFrameRing))
(let (t1)
  (setq t1
    (mapcar #'(lambda (f) `(|%l| " " ,@(|bright| (frameName f))))
      $interpreterFrameRing))
  (|sayKeyedMsg| 's2iz0024 (list (apply #'append t1)))))

Import items from another frame

[成员 p1048]
[frameNames p556]
[throwKeyedMsg p??]
[boot-equal p??]
[frameName p??]
[frameEnvironment p556]
[upcase p??]
[queryUserKeyedMsg p??]
[string2id-n p??]
[importFromFrame p563]
[sayKeyedMsg p329]
[clearCmdParts p505]
[seq p??]
[exit p??]
[putHist p590]
[get p??]
[getalist p??]
[$interpreterFrameRing p??]

— defun importFromFrame —
(defun importFromFrame (args)
  "Import items from another frame"
  (prog (temp1 fname fenv x v props vars plist prop val m)
    (declare (special $interpreterFrameRing))
    (when (and args (atom args)) (setq args (cons args nil)))
    (if (null args)
      (throwKeyedMsg 'S2IZ0073 nil) ; missing frame name
      (progn
        (setq temp1 args)
        (setq fname (car temp1))
        (setq args (cdr temp1))
        (cond
          ((null (member fname (frameNames)))
            (throwKeyedMsg 'S2IZ0074 (cons fname nil))) ; not frame name
          ((boot-equal fname (frameName (car $interpreterFrameRing)))
            (throwKeyedMsg 'S2IZ0075 NIL)) ; cannot import from curr frame
          (t
            (setq fenv (frameEnvironment fname))
            (cond
              ((null args)
                (setq x
                  (upcase (queryUserKeyedMsg 'S2IZ0076 (cons fname nil))))
                ; import everything?
                (cond
                  ((member (string2id-n x 1) '(y yes))
                    (setq vars nil)
                    (do ((tmp0 (caar fenv) (cdr tmp0)) (tmp1 nil))
                      ((or (atom tmp0)
                          (progn (setq tmp1 (car tmp0)) nil)
                          (progn
                            (progn
                              (setq v (car tmp1))
                              (setq props (cdr tmp1))
                              (progn
                                (setq m (car tmp3))
                                (progn
                                  (setq m (car tmp3))
                                  (setq m (car tmp3))
                                  nil)
                              (setq vars (cons m vars))))
                          (t (setq vars (cons v vars))))))))
                  (t nil)))))
      (importFromFrame (cons fname vars))))
    (t
The top level frame command

[frameSpad2Cmd p566]

— defun frame —

(defun |frame| (l)
  "The top level frame command"
  (frameSpad2Cmd l))
The top level frame command handler

(defun frameSpad2Cmd (args)
  "The top level frame command handler"
  (let (frameArgs arg a)
    (declare (special $options))
    (setq frameArgs '(|drop| |import| |last| |names| |new| |next|))
    (cond
      ($options
       (throwKeyedMsg 'S2IZ0016 ; frame command does not take options
                     (cons "frame" nil)))
      ((null args) (helpSpad2Cmd (cons 'frame nil)))
      (t
       (setq arg (selectOptionLC (car args) frameArgs 'optionError))
       (setq args (cdr args))
       (when (and (consp args)
                  (eq (qcdr args) nil)
                  (progn (setq a (qcar args)) t))
         (setq args a))
       (when (atom args) (setq args (object2Identifier args)))
       (case arg
         (|drop| (if (and args (consp args))
                    (throwKeyedMsg 'S2IZ0017 ; not a valid frame name
                                    (cons args nil))
                    (closeInterpreterFrame args))))
  )
)
32.6. FRAME FILE MESSAGES

32.6.1 Frame File Messages

--- Frame File Messages ---

S2IZ0016
The %1b system command takes arguments but no options.

S2IZ0017
%1b is not a valid frame name

S2IZ0018
You must provide a name for the new frame.

S2IZ0019
You cannot use the name %1b for a new frame because an existing frame already has that name.

S2IZ0020
There is only one frame active and therefore that cannot be closed. Furthermore, the frame name you gave is not the name of the current frame. The current frame is called %1b.

S2IZ0021
The current frame is the only active one. Issue %b )clear all %d to clear its contents.

S2IZ0022
There is no frame called %1b and so your command cannot be processed.

S2IZ0024
The names of the existing frames are: %1 %1
The current frame is the first one listed.

S2IZ0073
%b )frame import %d must be followed by the frame name. The names of objects in that frame can then optionally follow the frame name. For example,

%ceon %b )frame import calculus %d %ceoff
imports all objects in the %b calculus %d frame, and

%ceon %b )frame import calculus epsilon delta %d %ceoff
imports the objects named \%b epsilon \%d and \%b delta \%d from the frame \%b calculus \%d.
Please note that if the current frame contained any information about objects with these names, then that information would be cleared before the import took place.

S2IZ0074
You cannot import anything from the frame \%1b because that is not the name of an existing frame.

S2IZ0075
You cannot import from the current frame (nor is there a need!).

S2IZ0076
User verification required:
do you really want to import everything from the frame \%1b?
If so, please enter \%b y \%d or \%b yes \%d:

S2IZ0077
On your request, AXIOM will not import everything from frame \%1b.

S2IZ0078
Import from frame \%1b is complete. Please issue \%b )display all \%d if you wish to see the contents of the current frame.

S2IZ0079
AXIOM cannot import \%1b from frame \%2b because it cannot be found.
Chapter 33

)help help page Command

33.1 help help page man page

— help.help —

====================================================================
| help | help |
|-------|
====================================================================

User Level Required: interpreter

Command Syntax:

- )help
- )help commandName
- )help syntax

Command Description:

This command displays help information about system commands. If you issue

)help

then this very text will be shown. You can also give the name or abbreviation
of a system command to display information about it. For example,

)help clear

will display the description of the )clear system command.

The command
\texttt{)}help syntax

will give further information about the Axiom language syntax.

All this material is available in the AXIOM User Guide and in HyperDoc. In HyperDoc, choose the Commands item from the Reference menu.

\begin{verbatim}
A.1. Introduction

System commands are used to perform AXIOM environment management. Among the commands are those that display what has been defined or computed, set up multiple logical AXIOM environments (frames), clear definitions, read files of expressions and commands, show what functions are available, and terminate AXIOM.

Some commands are restricted: the commands

\texttt{)}set userlevel interpreter
\texttt{)}set userlevel compiler
\texttt{)}set userlevel development

set the user-access level to the three possible choices. All commands are available at development level and the fewest are available at interpreter level. The default user-level is interpreter. In addition to the \texttt{)}set command (discussed in description of command \texttt{)}set ) you can use the HyperDoc settings facility to change the user-level. Click on [Settings] here to immediately go to the settings facility.

Each command listing begins with one or more syntax pattern descriptions plus examples of related commands. The syntax descriptions are intended to be easy to read and do not necessarily represent the most compact way of specifying all possible arguments and options; the descriptions may occasionally be redundant.

All system commands begin with a right parenthesis which should be in the first available column of the input line (that is, immediately after the input prompt, if any). System commands may be issued directly to AXIOM or be included in .input files.

A system command argument is a word that directly follows the command name and is not followed or preceded by a right parenthesis. A system command option follows the system command and is directly preceded by a right parenthesis. Options may have arguments: they directly follow the option. This example may make it easier to remember what is an option and what is an argument:

\texttt{)}syscmd arg1 arg2 )opt1 opt1arg1 opt1arg2 )opt2 opt2arg1 ...
In the system command descriptions, optional arguments and options are enclosed in brackets ("[") and "]"). If an argument or option name is in italics, it is meant to be a variable and must have some actual value substituted for it when the system command call is made. For example, the syntax pattern description

\( \text{read fileName [quietly]} \)

would imply that you must provide an actual file name for fileName but need not use the \( \text{quietly} \) option. Thus

\( \text{read matrix.input} \)

is a valid instance of the above pattern.

System command names and options may be abbreviated and may be in upper or lower case. The case of actual arguments may be significant, depending on the particular situation (such as in file names). System command names and options may be abbreviated to the minimum number of starting letters so that the name or option is unique. Thus

\( \text{s Integer} \)

is not a valid abbreviation for the \( \text{set} \) command, because both \( \text{set} \) and \( \text{show} \) begin with the letter \"s\". Typically, two or three letters are sufficient for disambiguating names. In our descriptions of the commands, we have used no abbreviations for either command names or options.

In some syntax descriptions we use a vertical line \"|\" to indicate that you must specify one of the listed choices. For example, in

\( \text{set output fortran on | off} \)

only on and off are acceptable words for following boot. We also sometimes use \"...\" to indicate that additional arguments or options of the listed form are allowed. Finally, in the syntax descriptions we may also list the syntax of related commands.

====================================================================
Other help topics
====================================================================

Available help topics are:

- abbreviations
- assignment
- blocks
- browse
- boot
- cd
- clear
- clef
- close
- collection
- compile
- describe
- display
- edit
- fin
- for
- frame
- help
- history
- if
- iterate
- leave
- library
- lisp
- load
- ltrace
- parallel
- pquit
- quit
- read
- repeat
- savesystem
- set
- show
- spool
- suchthat
Available algebra help topics are:

33.2 Functions

The top level help command

```lisp
(defun help (l)
  "The top level help command"
  (helpSpad2Cmd l))
```

The top level help command handler

```lisp
(defun helpSpad2Cmd (args)
  "The top level help command handler"
  (unless (newHelpSpad2Cmd args)
    (sayKeyedMsg 's2iz0025 (cons args nil))))
```

defun newHelpSpad2Cmd

```lisp
(defun newHelpSpad2Cmd (makeInputFilename obey concat namestring)
```

33.2. FUNCTIONS

(defun newHelpSpad2Cmd (args)
  (let (sarg arg narg helpfile filestream line)
    (declare (special $syscommands |$useFullScreenHelp|))
    (when (null args) (setq args (list '?)))
    (if (> (|#| args) 1)
      (|sayKeyedMsg| 's2iz0026 nil)
      (progn
        (setq sarg (pname (car args)))
        (cond
          ((string= sarg "?") (setq args (list '|help|)))
          ((string= sarg ">%") (setq args (list '|history|)))
          ((string= sarg "%%%") (setq args (list '|history|)))
          (t nil))
        (setq arg (|selectOptionLC| (car args) $syscommands nil))
        (cond ((null arg) (setq arg (car args))))
        (setq narg (pname arg))
        (cond
          ((null (setq helpfile (makeInputFilename (list narg "help")))) nil)
          ($useFullScreenHelp|
            (obey (concat "$AXIOM/lib/SPADEDIT " (|namestring| helpfile))) t)
          (t
            (setq filestream (make-instream helpfile))
            (do ((line (|read-line| filestream nil) (|read-line| filestream nil))
                 (null line) (shut filestream))
                (say line)))))
  )
)
Chapter 34

)history help page Command

34.1  history help page man page

— history.help —

====================================================================
A.13. )history
====================================================================

User Level Required: interpreter

Command Syntax:

- )history )on
- )history )off
- )history )write historyInputFileName
- )history )show [n] [both]
- )history )save savedHistoryName
- )history )restore [savedHistoryName]
- )history )reset
- )history )change n
- )history )memory
- )history )file
- %
- %%(n)
- )set history on | off

Command Description:

The history facility within AXIOM allows you to restore your environment to
that of another session and recall previous computational results. Additional
commands allow you to review previous input lines and to create an .input
file of the lines typed to AXIOM.

AXIOM saves your input and output if the history facility is turned on (which is the default). This information is saved if either of

)setup history on
)history on

has been issued. Issuing either

)setup history off
)history off

will discontinue the recording of information.

Whether the facility is disabled or not, the value of % in AXIOM always refers to the result of the last computation. If you have not yet entered anything, % evaluates to an object of type Variable('%). The function %% may be used to refer to other previous results if the history facility is enabled. In that case, %%(n) is the output from step n if n > 0. If n < 0, the step is computed relative to the current step. Thus %%(−1) is also the previous step, %%(−2), is the step before that, and so on. If an invalid step number is given, AXIOM will signal an error.

The environment information can either be saved in a file or entirely in memory (the default). Each frame (description of command)frame has its own history database. When it is kept in a file, some of it may also be kept in memory for efficiency. When the information is saved in a file, the name of the file is of the form FRAME.axh where ‘FRAME’ is the name of the current frame. The history file is placed in the current working directory (see description of command )cd). Note that these history database files are not text files (in fact, they are directories themselves), and so are not in human-readable format.

The options to the )history command are as follows:

)change n
    will set the number of steps that are saved in memory to n. This option only has effect when the history data is maintained in a file. If you have issued )history )memory (or not changed the default) there is no need to use )history )change.

)on
    will start the recording of information. If the workspace is not empty, you will be asked to confirm this request. If you do so, the workspace will be cleared and history data will begin being saved. You can also turn the facility on by issuing )set history on.

)off
    will stop the recording of information. The )history )show command will
not work after issuing this command. Note that this command may be issued
to save time, as there is some performance penalty paid for saving the
environment data. You can also turn the facility off by issuing )set
history off.

)file
indicates that history data should be saved in an external file on disk.

)memory
indicates that all history data should be kept in memory rather than
saved in a file. Note that if you are computing with very large objects
it may not be practical to kept this data in memory.

)reset
will flush the internal list of the most recent workspace calculations so
that the data structures may be garbage collected by the underlying Lisp
system. Like )history )change, this option only has real effect when
history data is being saved in a file.

)restore [savedHistoryName]
completely clears the environment and restores it to a saved session, if
possible. The )save option below allows you to save a session to a file
with a given name. If you had issued )history )save jacobi the command
)history )restore jacobi would clear the current workspace and load the
contents of the named saved session. If no saved session name is
specified, the system looks for a file called last.axh.

)save savedHistoryName
is used to save a snapshot of the environment in a file. This file is
placed in the current working directory (see description of command )cd
). Use )history )restore to restore the environment to the state
preserved in the file. This option also creates an input file containing
all the lines of input since you created the workspace frame (for
example, by starting your AXIOM session) or last did a )clear all or
)clear completely.

)show [n] [both]
can show previous input lines and output results. )show will display up
to twenty of the last input lines (fewer if you haven’t typed in twenty
lines). )show n will display up to n of the last input lines. )show both
will display up to five of the last input lines and output results. )show
n both will display up to n of the last input lines and output results.

)write historyInputFile
creates an .input file with the input lines typed since the start of the
session/frame or the last )clear all or )clear completely. If
historyInputFileName does not contain a period ("."") in the filename,
.input is appended to it. For example, )history )write chaos and )history
)write chaos.input both write the input lines to a file called
chaos.input in your current working directory. If you issued one or more
undo commands, )history )write eliminates all input lines backtracked over as a result of )undo. You can edit this file and then use )read to have AXIOM process the contents.

Also See:
- )frame
- )read
- )set
- )undo

---

History recording is done in two different ways:

- all changes in variable bindings (i.e. previous values) are written to $HistList, which is a circular list
- all new bindings (including the binding to %) are written to a file called histFileName() one older session is accessible via the file $oldHistFileName()

### 34.2 Initialized history variables

The following global variables are used:

- $HistList, $HistListLen and $HistListAct which is the actual number of “undoable” steps
- $HistRecord collects the input line, all variable bindings and the output of a step, before it is written to the file histFileName().
- $HistAccess is a flag, which is reset by )history )off

The result of step n can be accessed by %n, which is translated into a call of fetchOutput(n).

The updateHist is called after every interpreter step. The putHist function records all changes in the environment to $HistList and $HistRecord.

**defvar $oldHistoryFileName**

--- initvars ---

(defvar $oldHistoryFileName '"vm/370 filename name component")

---

1 “frame” (32.5 p 565) “read” (42.1 p 642) “set” (45.36 p 808) “undo” (53.4 p 922)
34.3 DATA STRUCTURES

defvar $historyFileType

    — initvars —

(defvar $historyFileType '|axh| "vm/370 filename type component")

---

defvar $historyDirectory

    — initvars —

(defvar $historyDirectory 'A "vm/370 filename disk component")

---

defvar $useInternalHistoryTable

    — initvars —

(defvar $useInternalHistoryTable t "t means keep history in core")

---

34.3 Data Structures

34.4 Functions

defun makeHistFileName

[makePathname p1042]

    — defun makeHistFileName —

(defun makeHistFileName (fname)
  (makePathname fname $historyFileType $historyDirectory)))

---
defun oldHistFileName

[makeHistFileName p579]
[$oldHistoryFileName p578]

— defun oldHistFileName —

(defun |oldHistFileName| ()
  (declare (special |$oldHistoryFileName|))
  (|makeHistFileName| |$oldHistoryFileName|))

——

defun histFileName

[makeHistFileName p579]
[$interpreterFrameName p??]

— defun histFileName —

(defun |histFileName| ()
  (declare (special |$interpreterFrameName|))
  (|makeHistFileName| |$interpreterFrameName|))

——

defun histInputFileName

[makePathname p1042]
[$interpreterFrameName p??]
[$historyDirectory p579]

— defun histInputFileName —

(defun |histInputFileName| (fn)
  (declare (special |$interpreterFrameName| |$historyDirectory|))
  (if (null fn)
      (|makePathname| |$interpreterFrameName| 'input |$historyDirectory|)
    (|makePathname| fn 'input |$historyDirectory|)))

——
defun initHist

(defun initHist ()
    (let (oldFile newFile)
        (declare (special $useInternalHistoryTable |$HiFiAccess|))
        (if $useInternalHistoryTable
            (initHistList)
            (progn
                (setq oldFile (oldHistFileName))
                (setq newFile (histFileName))
                (histFileErase oldFile)
                (when (makeInputFilename newFile) (replaceFile oldFile newFile))
                (setq $HiFiAccess t)
                (initHistList))))

— defun initHist —

defun initHistList

(defun initHistList ()
    (let (li)
        (declare (special $HistListLen $HistList $HistListAct $HistRecord))
        (setq $HistListLen 20)
        (setq $HistList (list nil))
        (setq li $HistList)
        (do ((i 1 (1+ i)))
            (> i $HistListLen) nil)
        (setq li (cons nil li)))

— defun initHistList —
The top level history command

(defun history (l)
  "The top level history command"
  (declare (special |$options|))
  (if (or l (null |$options|))
    (|sayKeyedMsg| 's2ih0006 nil) ; syntax error
    (|historySpad2Cmd|)))

The top level history command handler
(defun historySpad2Cmd ()
"The top level history command handler"
(let (histOptions opts opt optargs x)
 (declare (special $options $HiFiAccess $IOindex)))
 (setq histOptions
 '([on] [off] [yes] [no] [change] [reset] [restore] [write] [save] [show] [file] [memory]))
 (setq opts
 (prog (tmp1)
  (setq tmp1 nil)
  (return
   (do ((tmp2 $options (cdr tmp2)) (tmp3 nil))
    ((or (atom tmp2)
       (progn
        (setq tmp3 (car tmp2))
        nil)
       (progn
        (progn
         (setq opt (car tmp3))
         (setq optargs (cdr tmp3))
         tmp3)
        nil))
    (nreverse0 tmp1))
    (setq tmp1
      (cons
       (cons
        (cons
         ([selectOptionLC histOptions 'optionError] optargs)
        tmp1))))))
 (do ((tmp4 opts (cdr tmp4)) (tmp5 nil))
 (or (atom tmp4)
    (progn
     (setq tmp5 (car tmp4))
     nil)
    (progn
     (progn
      (setq opt (car tmp5))
      (setq optargs (cdr tmp5))
      tmp5)
     nil))
   nil)
 (seq)
(exit
 (cond
  ((|member| opt '((|on| |yes|)))
   (cond
    (|$HiFiAccess|
     (|sayKeyedMsg| 'S2IH0007 nil)) ; history already on
    ((eq $IDindex 1)
     (setq $HiFiAccess t)
     (|initHistList|)
     (|sayKeyedMsg| 'S2IH0008 nil)) ; history now on
    (t
     (setq x ; really want to turn history on?
       (upcase (|queryUserKeyedMsg| 'S2IH0009 nil)))
     (cond
      ((member (string2id-n x 1) '(Y YES))
       (|histFileErase| (|histFileName|))
       (setq $HiFiAccess t)
       (setq $options nil)
       (|clearSpad2Cmd| '(all))
       (|sayKeyedMsg| 'S2IH0008 nil)) ; history now on
       (|initHistList|))
      (t
       (|sayKeyedMsg| 'S2IH0010 nil)))))) ; history still off
  ((|member| opt '((|off| |no|)))
   (cond
    ((null $HiFiAccess)
     (|sayKeyedMsg| 'S2IH0011 nil)) ; history already off
    (t
     (setq $HiFiAccess nil)
     (|disableHist|)
     (|sayKeyedMsg| 'S2IH0012 nil)))) ; history now off
  ((eq opt '|file|) (|setHistoryCore| nil))
  ((eq opt '|memory|) (|setHistoryCore| t))
  ((eq opt '|reset|) (|resetInCoreHist|))
  ((eq opt '|save|) (|saveHistory| optargs))
  ((eq opt '|show|) (|showHistory| optargs))
  ((eq opt '|change|) (|changeHistListLen| (car optargs)))
  ((eq opt '|restore|) (|restoreHistory| optargs))
  ((eq opt '|write|) (|writeInputLines| optargs 1))))
  'done))

(defun setHistoryCore
  We case on the inCore argument value

  If history is already on and is kept in the same location as requested (file or memory)
then complain.

If history is not in use then start using the file or memory as requested. This is done by simply setting the $useInternalHistoryTable to the requested value, where T means use memory and NIL means use a file. We tell the user.

If history should be in memory, that is inCore is not NIL, and the history file already contains information we read the information from the file, store it in memory, and erase the history file. We modify $useInternalHistoryTable to T to indicate that we’re maintaining the history in memory and tell the user.

Otherwise history must be on and in memory. We erase any old history file and then write the in-memory history to a new file.
CHAPTER 34. )HISTORY HELP PAGE COMMAND

((not (eql |$IOindex| 0))
  (setq l (length (rkeyids (|histFileName|))))
  (do ((i 1 (1+ i)))
      (> i l) nil)
    (setq vec (unwind-protect (|readHiFi| i) (|disableHist|)))
    (setq |$internalHistoryTable|
      (cons (cons i vec) |$internalHistoryTable|)))
  (|histFileErase| (|histFileName|)))
(setq |$useInternalHistoryTable| t)
(|sayKeyedMsg| 'S2I0032 nil)) ; use memory history
(t
  (setq |$HiFiAccess| nil)
  (|histFileErase| (|histFileName|))
  (setq str
    (rdefiostream
      (cons
        '(mode . output)
        (cons
          (cons 'file (|histFileName|)
            nil)))))
  (do ((tmp0 (reverse |$internalHistoryTable|) (cdr tmp0))
       (tmp1 nil)
       (or (atom tmp0)
          (progn
            (setq tmp1 (car tmp0))
            nil))
       (progn
         (progn
           (setq n (car tmp1))
           (setq rec (cdr tmp1))
           (setq i nil))
         nil)))
  nil)
  (spadrwrite (|object2Identifier| n) rec str))
(rshut str)
(setq |$HiFiAccess| t)
(setq |$internalHistoryTable| nil)
(setq |$useInternalHistoryTable| nil)
(|sayKeyedMsg| 's2ih0031 nil))) ; use file history

defvar $underbar

Also used in the output routines.

— initvars —

(defvar underbar "_")
defun writeInputLines

[sayKeyedMsg p329]
[throwKeyedMsg p??]
[size p1045]
[spadddifference p??]
[concat p1047]
[stringp p??]
[readHiFi p580]
[histInputFileName p580]
[histFileErase p617]
[defi ostream p982]
[namestring p1040]
[shut p982]
[underbar p586]
[$HiFiAccess p733]
[$IOindex p10]

— defun writeInputLines —

(defun writeInputLines (fn initial)
  (let (maxn breakChars vecl k svec done n lineList file inp)
    (declare (special underbar $HiFiAccess $IOindex))
    (cond
      ((null $HiFiAccess) (sayKeyedMsg 's2ih0013 nil)) ; history is not on
      ((null fn) (throwKeyedMsg 's2ih0038 nil)) ; missing file name
      (t
       (setq maxn 72)
       (setq breakChars (cons '|| (cons '+ nil)))
       (do ((tmp0 (spadddifference $IOindex 1))
            (i initial (+ i 1)))
           (> i tmp0) nil)
       (setq vecl (car (readHiFi i))
            (when (stringp vecl) (setq vecl (cons vecl nil)))
            (dolist (vec vecl)
              (setq n (size vec))
              (do ()
               ((null (> n maxn)) nil)
               (setq done nil)
               (do ((j i (1+ j))
                    ((or (> j maxn) (null (null done))) nil)
                    (setq k (spadddifference (1+ maxn) j))
                    (when (member (elt vec k) breakChars)
                      (do ((j (1+ j) (1+ j))))
                      (null (null done)))))))
          (null (null done)) nil)
    (when (null done) nil)
    (when (null (null done)) nil))
)}
(setq svec (concat (substring vec 0 (1+ k)) underbar))
(setq lineList (cons svec lineList))
(setq done t)
(setq vec (substring vec (1+ k) nil))
(setq n (size vec))
(when done (setq n 0)))
(setq lineList (cons svec lineList)))))
(setq file (\|histInputFileName\| fn))
(\|histFileErase\| file)
(setq inp
(defiostream
 (cons
   '(mode . output)
   (cons (cons 'file file) nil)) 255 0))
(dolist (x (\|removeUndoLines\| (nreverse lineList)))
 (write-line x inp))
(cond
 (not (eq fn '\|redo\|))
   (\|sayKeyedMsg\| 's2ih0014 ; edit this file to see input lines
     (list (\|namestring\| file))))
 (shut inp)
 nil)))

defun resetInCoreHist

\|$HistListAct\| p??
\|$HistListLen\| p??
\|$HistList\| p??

defun resetInCoreHist —

(defun \|resetInCoreHist\| ()
 (declare (special \|$HistListAct\| \|$HistListLen\| \|$HistList\|))
 (setq \|$HistListAct\| 0)
 (do ((i 1 (1+ i)))
   (> i \|$HistListLen\|) nil)
 (setq \|$HistList\| (cdr \|$HistList\|))
 (rplaca \|$HistList\| nil)))
defun changeHistListLen

[sayKeyedMsg p329]
[spaddifference p??]
[$HistListLen p??]
[$HistList p??]
[$HistListAct p??]

— defun changeHistListLen —

(defun changeHistListLen (n)
  (let ((dif l)
    (declare (special $HistListLen $HistList $HistListAct))
  (if (null (integerp n))
    (sayKeyedMsg 's2ih0015 (list n)) ; only positive integers
    (progn
      (setq dif (spaddifference n $HistListLen))
      (setq $HistListLen n)
      (setq 1 (cdr $HistList))
      (cond
        (> dif 0)
        (do ((i 1 (1+ i)))
            (> i dif) nil)
          (setq 1 (cons nil l)))
        (minusp dif)
        (do ((tmp0 (spaddifference dif))
            (i 1 (1+ i)))
            (> i tmp0) nil)
          (setq 1 (cdr l)))
      (cond
        (> $HistListAct n) (setq $HistListAct n))
      (t nil))))
  (rplacd $HistList 1)
  'done)))))

defun updateHist

[startTimingProcess p??]
[updateInCoreHist p590]
[writeHiFi p602]
[disableHist p603]
[updateCurrentInterpreterFrame p559]
[stopTimingProcess p??]
[IOindex p10]
[$HiFiAccess p733]
--- defun updateHist ---

(defun updateHist ()
  (declare (special $IOindex $HiFiAccess $HistRecord $mkTestInputStack $currentLine))
  (when $IOindex
    (startTimingProcess 'history)
    (updateInCoreHist)
  (when $HiFiAccess
    (unwind-protect (writeHiFi) (disableHist))
    (setq $HistRecord nil))
  (incf $IOindex)
  (updateCurrentInterpreterFrame)
  (setq $mkTestInputStack nil)
  (setq $currentLine nil)
  (stopTimingProcess 'history))

---

defun updateInCoreHist

(defun updateInCoreHist ()
  (declare (special $HistList $HistListLen $HistListAct))
  (setq $HistList (cdr $HistList))
  (rplaca $HistList nil)
  (when (> $HistListLen $HistListAct)
    (setq $HistListAct (1+ $HistListAct)))

---

defun putHist

(defun putHist ()
  (declare (special $HistList $HistListLen $HistListAct))
  (setq $HistList (cdr $HistList))
  (rplaca $HistList nil)
  (when (> $HistListLen $HistListAct)
    (setq $HistListAct (1+ $HistListAct))))
---

defun putHist

(defun putHist (x prop val e)
  (declare (special $HiFiAccess))
  (when (null (eq x 'x)) (recordOldValue x prop (get x prop e))
  (when $HiFiAccess (recordNewValue x prop val))
  (putIntSymTab x prop val e))

---

defun recordNewValue

(defun recordNewValue (x prop val)
  (startTimingProcess 'history)
  (recordNewValue0 x prop val
  (stopTimingProcess 'history))

---

defun recordNewValue0

(defun recordNewValue0 (x prop val)
  (let (p1 p2 p)
    (declare (special $HistRecord))
    (if (setq p1 (assq x $HistRecord))
      (if (setq p2 (assq prop (cdr p1)))
        (rplacd p2 val)
        (rplacd p1 (cons (cons prop val) (cdr p1))))
      (progn
        (setq p (cons x (list (cons prop val))))))
defun recordOldValue

(defun recordOldValue (x prop val)
  (startTimingProcess '|history|)
  (recordOldValue0 x prop val)
  (stopTimingProcess '|history|))

---

defun recordOldValue0

(defun recordOldValue0 (x prop val)
  (let ((p1 p)
         (declare (special |$HistList|))
         (when (setq p1 (assq x (car |$HistList|)))
           (when (null (assq prop (cdr p1)))
             (rplacd p1 (cons (cons prop val) (cdr p1))))
           (setq p (cons x (list (cons prop val))))
           (rplaca |$HistList| (cons p (car |$HistList|))))))

---

defun undoInCore

(defun undoInCore
  (undoChanges p)
  (spadddifference p)
  (readHiFi p))
34.4. FUNCTIONS

[disableHist p603]
[assq p1050]
[sayKeyedMsg p329]
[putHist p590]
[updateHist p589]
[$HistList p??]
[$HistListLen p??]
[$IOindex p10]
[$HiFiAccess p733]
[$InteractiveFrame p??]

— defun undoInCore —

(defun undoInCore (n)
  (let (li vec p p1 val)
    (declare (special $HistList $HistListLen $IOindex $HiFiAccess
       $InteractiveFrame))
    (setq li $HistList)
    (do ((i n (+ i 1)))
      (> i $HistListLen) nil)
      (setq li (cdr li)))
    (undoChanges li)
    (setq n (spaddifference (spaddifference $IOindex n) 1))
    (and
      (> n 0)
      (if $HiFiAccess
         (progn
           (setq vec (cdr (unwind-protect (readHiFi n) (disableHist))))
           (setq val
             (and
              (setq p (assq '% vec))
              (setq p1 (assq 'value (cdr p)))
              (cdr p1)))
             ((sayKeyedMsg 's2ih0019 (cons n nil)))) ; no history file
             (setq $InteractiveFrame (putHist '% 'value val $InteractiveFrame))
             (updateHist))))

——

defun undoChanges

[boot-equal p??]
[undoChanges p593]
[putHist p590]
[$HistList p??]
[$InteractiveFrame p??]
--- defun undoChanges ---

(defun undoChanges (li)
  (let (x)
    (declare (special $HistList $InteractiveFrame))
    (when (null (boot-equal (cdr li) $HistList)) (undoChanges (cdr li))
      (dolist (p1 (car li))
        (setq x (car p1))
        (dolist (p2 (cdr p1))
          (putHist x (car p2) (cdr p2) $InteractiveFrame)))))

---

defun undoFromFile

[seq p??]
[exit p??]
[recordOldValue p592]
[recordNewValue p591]
[readHiFi p601]
[disableHist p603]
[putHist p590]
[assq p1050]
[updateHist p589]
[$InteractiveFrame p??]
[$HiFiAccess p733]

--- defun undoFromFile ---

(defun undoFromFile (n)
  (let (varl prop vec x p p1 val)
    (declare (special $InteractiveFrame $HiFiAccess))
    (do ((tmp0 (caar $InteractiveFrame) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
            (progn (setq tmp1 (car tmp0)) nil)
            (progn (setq x (car tmp1))
                    (setq varl (cdr tmp1))
                    (setq tmp1) val)
            nil)
      nil)
    (exit
     (do ((tmp2 varl (cdr tmp2)) (p nil))
         nil
         (seq
          (progn
           (setq x (car tmp1))
           (setq varl (cdr tmp1))
           (setq tmp1) val)
          nil)
       nil)))
34.4. FUNCTIONS

((or (atom tmp2) (progn (setq p (car tmp2)) nil)) nil)

(seq
  (exit
    (progn
      (setq prop (car p))
      (setq val (cdr p))
      (when val
        (progn
          (when (null (eq x '%))
            (|recordOldValue| x prop val))
          (when |$HiFiAccess|
            (|recordNewValue| x prop val))
          (rplacd p nil)))))))

(do ((i 1 (1+ i)))
    (> i n) nil)
  (setq vec
    (unwind-protect (cdr (|readHiFi| i)) (|disableHist|)))

(do ((tmp3 vec (cdr tmp3)) (p1 nil))
    ((or (atom tmp3) (progn (setq p1 (car tmp3)) nil)) nil)
  (setq x (car p1))

(do ((tmp4 (cdr p1) (cdr tmp4)) (p2 nil))
    ((or (atom tmp4) (progn (setq p2 (car tmp4)) nil)) nil)
  (setq |$InteractiveFrame|
    (|putHist| x (car p2) (CDR p2) |$InteractiveFrame|)))

(setq val
  (and
    (setq p (assq '% vec))
    (setq p1 (assq '|value| (cdr p)))
    (cdr p1)))

(setq |$InteractiveFrame| (|putHist| '% '|value| val |$InteractiveFrame|))

(deffunction saveHistory
  (sayKeyedMsg p329)
  (makeInputFilename p983)
  (histFileName p580)
  (throwKeyedMsg p22)
  (makeHistFileName p579)
  (histInputFileName p580)
  (writeInputLines p587)
  (histFileErase p617)
  (rdefiostream p22)
  (spadrwrite0 p604)
  (object2Identifier p22)
defun saveHistory (fn)
  (let ((|$seen| savefile inputfile saveStr n rec val)
   (declare (special |$seen| |$HiFiAccess| |$useInternalHistoryTable| |
   |$internalHistoryTable|))
   (setq |$seen| (make-hash-table :test #'eq))
   (cond
     ((null |$HiFiAccess|)
      (|sayKeyedMsg| 's2ih0016 nil)) ; the history file is not on
     ((and (null |$useInternalHistoryTable|)
             (null (makeInputFilename (|histFileName|))))
      (|sayKeyedMsg| 's2ih0022 nil)) ; no history saved yet
     ((null fn)
      (|throwKeyedMsg| 's2ih0037 nil)) ; need to specify a history filename
     (t
      (setq savefile (|makeHistFileName| fn))
      (setq inputfile (|histInputFileName| fn))
      (|writeInputLines| fn 1)
      (|histFileErase| savefile)
      (when |$useInternalHistoryTable|
        (setq saveStr
          (rdefiostream
            (cons '(mode . output)
              (cons (cons 'file savefile) nil))))
        (do ((tmp0 (reverse |$internalHistoryTable|) (cdr tmp0))
             (tmp1 nil))
            ((or (atom tmp0)
                  (progn (setq tmp1 (car tmp0)) nil)
                  (progn
                    (setq n (car tmp1))
                    (setq rec (cdr tmp1))
                    tmp1)
                  nil)
             nil)
        (setq val (spadrwrite0 (|object2Identifier| n) rec saveStr))
        (when (eq val '|writifyFailed|)
          (|sayKeyedMsg| 's2ih0035 ; can't save the value of step
            (list n inputfile))))
        (rshut saveStr))
      (|sayKeyedMsg| 's2ih0018 ; saved history file is...}

---

CHAPTER 34. HISTORY HELP PAGE COMMAND

[rshut p??]
namestring p1040
|$seen p??
|$HiFiAccess p733
|$useInternalHistoryTable p579
|$internalHistoryTable p??]
(cons (list 'namestring 'savefile) nil)))

defun restoreHistory

(let ((options fnq) restfile curfile l oldInternal vec line x a)
  (declare (special options internalHistoryTable HiFiAccess e
       useInternalHistoryTable InteractiveFrame oldHistoryFileName))
  (cond
    ((null fn) (setq fnq oldHistoryFileName)))
    ((and (consp fn)
       (eq (qcdr fn) nil)

---

(defun restoreHistory (fn)
  (let ((options fnq) restfile curfile l oldInternal vec line x a)
    (declare (special options internalHistoryTable HiFiAccess e
        useInternalHistoryTable InteractiveFrame oldHistoryFileName))
    (cond
      ((null fn) (setq fnq oldHistoryFileName)))
      ((and (consp fn)
        (eq (qcdr fn) nil)
(progn
  (setq fnq (qcar fn))
  t)
  (identp fnq))
(setq fnq fnq))
(t (throwKeyedMsg 's2ih0023 (cons fnq nil))) ; invalid filename
(setq restfile (makeHistFileName fnq))
(if (null (makeInputFilename restfile))
  (sayKeyedMsg 's2ih0024 ; file does not exist
    (cons (namestring restfile) nil))
(progn
  (setq $options nil)
  (clearSpad2Cmd '(all))
  (setq curfile (histFileName))
  (histFileErase curfile)
  ($copy restfile curfile)
  (setq l (length (rkeyids curfile)))
  (setq $HiFiAccess t)
  (setq oldInternal $useInternalHistoryTable)
  (setq $useInternalHistoryTable nil)
  (when oldInternal (setq $internalHistoryTable nil))
  (do ((i 1 (1+ i)))
    (< i l) nil)
  (setq vec (unwind-protect (readHiFi i) (disableHist)))
  (when oldInternal
    (setq $internalHistoryTable
      (cons (cons i vec) $internalHistoryTable)))
  (setq line (car vec))
  (dolist (p1 (cdr vec))
    (setq x (car p1))
    (do ((tmp1 (cdr p1) (cdr tmp1)) (p2 nil))
      (or (atom tmp1) (setq p2 (car tmp1)) nil)
      (setq $InteractiveFrame
        (putHist x
          (car p2) (cdr p2) $InteractiveFrame))))
  (updateInCoreHist))
(progn
  (setq $e ($InteractiveFrame))
  (do ((tmp2 (caar $InteractiveFrame) (cdr tmp2)) (tmp3 nil))
    (or (atom tmp2)
      (progn
        (setq tmp3 (car tmp2))
        nil)
      (progn
        (setq a (car tmp3))
        tmp3
        nil))
    nil)
  (when (get a 'localModemap $InteractiveFrame)
    (rempropI a 'localModemap))
(rempropI a 'localVars)
(rempropI a 'mapBody))
(setq $IOindex (1+ l))
(setq $useInternalHistoryTable oldInternal)
(sayKeyedMsg 'S2IH0025 ; workspace restored
 (cons (namestring restfile) nil))
(clearCmdSortedCaches))

_____________________

defun setIOindex

[$IOindex p10]

defun |setIOindex|

(defun |setIOindex| (n)
  (declare (special |$IOindex|))
  (setq |$IOindex| n))

_____________________

defun showInput

[tab p??]
[readHiFi p601]
[disableHist p603]
[sayMSG p331]

defun |showInput|

(defun |showInput| (mini maxi)
  (let (vec l)
    (do ((|ind| mini (+ |ind| 1)))
        ((> |ind| maxi) nil)
      (setq vec (unwind-protect (|readHiFi| |ind|) (|disableHist|)))
      (cond
        ((> 10 |ind|) (tab 2))
        ((> 100 |ind|) (tab 1))
        (t nil))
      (setq l (car vec))
      (if (stringp l)
        (sayMSG (list " [" |ind| "] " (car vec)))
        (progn
          ...))

_____________________

(setq ln (car tmp0)) nil)  
(|sayMSG| (list " " ln)))))))

---

defun showInOut

[assq p1050]  
[spadPrint p??]  
[objValUnwrap p??]  
[objMode p??]  
[readHiFi p601]  
[disableHist p603]  
[sayMSG p331]

--- defun showInOut ---

defun |showInOut| (mini maxi)
(let (vec Alist triple)
  (do ((ind mini (+ ind 1)))
    ((> ind maxi) nil)
    (setq vec (unwind-protect (|readHiFi| ind) (|disableHist|)))
    (|sayMSG| (cons (car vec) nil))
    (cond
      ((setq Alist (assq '% (cdr vec)))
       (setq triple (cdr (assq '|value| (cdr Alist))))
       (setq $IOindex ind)
       (|spadPrint| (|objValUnwrap| triple) (|objMode| triple)))))))

---

defun fetchOutput

[boot-equal p??]  
[spaddifference p??]  
[getI p??]  
[throwKeyedMsg p??]  
[readHiFi p601]  
[disableHist p603]  
[assq p1050]

--- defun fetchOutput ---
(defun |fetchOutput| (n)
 (let (vec Alist val)
    (cond
      ((and (boot-equal n (spaddifference 1)) (setq val (|getI| '%' '|value|)))
       val)
      (|$HiFiAccess|
       (setq n
         (cond
          ((minusp n) (+ |$IOindex| n))
          (t n)))
       (cond
        ((>= n |$IOindex|)
         (|throwKeyedMsg| 'S2IH0001 (cons n nil))) ; no step n yet
        (> 1 n)
         (|throwKeyedMsg| 's2ih0002 (cons n nil))) ; only nonzero steps
        (t
         (setq vec (unwind-protect (|readHiFi| n) (|disableHist|))))
         (cond
          (setq Alist (assq '%' (cdr vec)))
          (cond
           ((setq val (cdr (assq '|value| (cdr Alist))))
            val)
           (t
            (|throwKeyedMsg| 's2ih0003 (cons n nil))))))) ; no step value
        (t (|throwKeyedMsg| 's2ih0004 nil)))) ; history not on

---

Read the history file using index n

[assoc p??]
[keyedSystemError p??]
[qcdr p??]
[rdefostream p??]
[histFileName p580]
[spadrread p605]
[object2Identifier p??]
[rshut p??]
[$useInternalHistoryTable p579]
[$internalHistoryTable p??]

— defun readHiFi —

(defun |readHiFi| (n)
  "Read the history file using index n"
  (let (pair HiFi vec)
(declare (special |$useInternalHistoryTable| |$internalHistoryTable|))
(if |$useInternalHistoryTable|
  (progn
    (setq pair (assoc n |$internalHistoryTable|))
    (if (atom pair)
      (keyedSystemError 's2ih0034 nil) ; missing element
      (setq vec (qcdr pair)))
    (progn
      (setq HiFi
        (rdefiostream
          (cons
            '(mode . input)
            (cons
              (cons 'file (histFileName)) nil))))
      (setq vec (spadrread (object2Identifier n) HiFi))
      (rshut HiFi))
    vec))

—— defun writeHiFi ——

(defun |writeHiFi| ()
  "Writes information of the current step to history file"
  (let (HiFi)
    (declare (special |$useInternalHistoryTable| |$internalHistoryTable|
      |$IOindex| |$HistRecord| |$currentLine|))
    (if |$useInternalHistoryTable|
      (setq |$internalHistoryTable|
        (cons
          (cons |$IOindex|
            (cons |$currentLine| |$HistRecord|))
          |$internalHistoryTable|))
      (progn
..."
34.4. FUNCTIONS

(defun disableHist ()
"Disable history if an error occurred"
(declare (special $HiFiAccess))
(cond
((null $HiFiAccess)
  (histFileErase (histFileName)))
(t nil)))

(defun writeHistModesAndValues ()
(let (a x)
  (declare (special $InteractiveFrame))
  (do ((tmp0 (caar $InteractiveFrame) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
            (progn
              (setq tmp1 (car tmp0))
              nil))
   (cons (cons (cons 'file (histFileName)) nil))
   (spadrwrite (cons $currentLine $HistRecord) HiFi)
   (rshut HiFi))))))

Disable history if an error occurred

[histFileErase p617]
[histFileName p580]
[$HiFiAccess p733]

---

(defun |disableHist| ()
"Disable history if an error occurred"
(declare (special |$HiFiAccess|))
(cond
((null |$HiFiAccess|)
  (histFileErase (histFileName)))
(t nil)))

---

defun writeHistModesAndValues

[get p590]
[putHist p590]
[$InteractiveFrame p590]

---

(defun |writeHistModesAndValues| ()
(let (a x)
  (declare (special |$InteractiveFrame|))
  (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
            (progn
              (setq tmp1 (car tmp0))
              nil))
   (cons (cons (cons 'file (histFileName)) nil))
   (spadrwrite (cons |$currentLine| |$HistRecord|) HiFi)
   (rshut HiFi)))))))
34.5 Lisplib output transformations

Lisplib output transformations

Some types of objects cannot be saved by LISP/VM in lisplib. These functions transform an object to a writable form and back.

---

(defun spadrwrite0

[|safeWritify p606|]
[|rwrite p604|]

---

defun spadrwrite0

(defun spadrwrite0 (vec item stream)
  (let (val)
    (setq val (|safeWritify| item))
    (if (eq val '|writifyFailed|)
      val
      (progn
        (|rwrite| vec val stream)
        item))))

---

(defun Random write to a stream

[rwrite p604]
[|pname p1045|]
[|identp p1046|]
— defun rwrite —

(defun |rwrite| (key val stream)
  (when (identp key) (setq key (pname key)))
  (|rwrite| key val stream))

———

defun spadrwrite

[spadrwrite0 p604] [throwKeyedMsg p??]

— defun spadrwrite —

(defun spadrwrite (vec item stream)
  (let (val)
    (setq val (spadrwrite0 vec item stream))
    (if (eq val '|writifyFailed|)
      (|throwKeyedMsg| 's2ih0036 nil) ; cannot save value to file
       item)))

———

defun spadrread

[dewritify p615] [rread p605]

— defun spadrread —

(defun spadrread (vec stream)
  (|dewritify| (|rread| vec stream nil)))

———

defun Random read a key from a stream

RREAD takes erroval to return if key is missing [rread p605] [identp p1046] [pname p1045]

— defun rread —
(defun |rread| (key rstream errorval)
  (when (identp key) (setq key (pname key)))
  (rread key rstream errorval))

---

defun unwritable?

(defun unwritable? (ob)
  (cond
   ((or (consp ob) (vecp ob)) nil)
   ((or (compiled-function-p ob) (hash-table-p ob)) t)
   ((or (placep ob) (readtablep ob)) t)
   ((floatp ob) t)
   (t nil)))

---

defun writifyComplain

Create a full isomorphic object to be saved in a lisplib. Note that dewritify(writify(x)) preserves UEQUALity of hashtables. HASHTABLEs go both ways. READTABLEs cannot presently be transformed back. [sayKeyedMsg p329] [$writifyComplained p??]

(defun writifyComplain (s)
  (declare (special |$writifyComplained|))
  (unless |$writifyComplained|
    (setq |$writifyComplained| t)
    (|sayKeyedMsg| 's2ih0027 (list s))) ; cannot save value

---

defun safeWritify

([writifyTag p??]
 [writify p610]
— defun safeWritify —

(defun safeWritify (ob)
  (catch 'writifyTag (writify ob)))

———

defun writify,writifyInner

[writifyTag p??]
[seq p??]
[exit p??]
[hget p1044]
[qcar p??]
[qcdr p??]
[spadClosure? p611]
[writify,writifyInner p607]
[hput p1044]
[qrplaca p??]
[qrplacd p??]
[veep p??]
[isDomainOrPackage p875]
[mkEvalable p913]
[devaluate p??]
[qvmaxindex p??]
[qsetvelt p??]
[qvelt p??]
[constructor? p??]
[hkeys p1044]
[hashtable-class p??]
[placep p??]
[boot-equal p??]
[$seen p??]
[$NonNullStream p611]
[$NullStream p611]

— defun writify,writifyInner —

(defun writify,writifyInner (ob)
  (prog (e name tmp1 tmp2 tmp3 x qcar qcdr d n keys nob)
    (declare (special |$seen| |$NonNullStream| |$NullStream|))
    (return
     (seq
      (when (null ob) (exit nil)))

(when (setq e (hget |$seen| ob)) (exit e))
(when (consp ob)
  (exit
   (seq
    (setq qcar (qcar ob))
    (setq qcdr (qcdr ob))
    (when (setq name (|spadClosure?| ob))
      (exit
       (seq
        (setq d (|writify,writifyInner| (qcdr ob)))
        (setq nob
          (cons 'writified!!
            (cons 'spadclosure
              (cons d (cons name nil))))))
        (hput |$seen| ob nob)
        (hput |$seen| nob nob)
        (exit nob))))
  (when
    (and
      (consp ob)
      (eq (qcar ob) 'lambda-closure)
      (progn
        (setq tmp1 (qcdr ob))
        (and (consp tmp1)
          (progn
            (setq tmp2 (qcdr tmp1))
            (and
              (consp tmp2)
              (progn
                (setq tmp3 (qcdr tmp2))
                (and (consp tmp3)
                  (progn
                    (setq x (qcar tmp3))
                    t)))))))))
  (exit
   (throw '|writifyTag| '|writifyFailed|)))
  (setq nob (cons qcar qcdr))
  (hput |$seen| ob nob)
  (hput |$seen| nob nob)
  (setq qcar (|writify,writifyInner| qcar))
  (setq qcdr (|writify,writifyInner| qcdr))
  (qrplaca nob qcar)
  (qrplacd nob qcdr)
  (exit nob))))
(when (vecp ob)
  (exit
   (seq
    (when (|isDomainOrPackage| ob)
      (setq d (|mkEvalable| (|devaluate| ob)))
      (setq nob (list 'writified!! 'devaluated (|writify,writifyInner| d))))
(hput $seen ob nob)
(hput $seen nob nob)
(exit nob))
(setq n (qvmaxindex ob))
(setq nob (make-array (1+ n))
(hput $seen ob nob)
(hput $seen nob nob)
(do ((i 0 (=! i)))
   ((> i n) nil)
   (qsetvelt nob i (|writify,writifyInner| (qvelt ob i)))
  )
)(exit nob))
(when (eq ob 'writified!!)
  (exit
   (cons 'writified!! (cons 'self nil)))
)(when (|constructor?| ob)
  (exit ob))
(when (compiled-function-p ob)
  (exit
   (throw '|writifyTag| '|writifyFailed|)))
(when (hash-table-p ob)
  (setq nob (cons 'writified!! nil))
  (hput $seen ob nob)
  (hput $seen nob nob)
  (setq keys (hkeys ob))
  (qrplacd nob
   (cons
    'hashtable
    (cons
     (hashtable-class ob)
     (cons
      (|writify,writifyInner| keys)
      (cons
       (prog (tmp0)
          (setq tmp0 nil)
          (return
           (do ((tmp1 keys (cdr tmp1)) (k nil))
              ((or (atom tmp1)
               (progn
                (setq k (car tmp1))
                nil))
               (nreverse0 tmp0))
               (setq tmp0
               (cons (|writify,writifyInner| (hget ob k)) tmp0))))
               (nil))))))
  (exit nob))
(when (placep ob)
  (setq nob (cons 'writified!! (cons 'place nil))
  (hput $seen ob nob)
  (hput $seen nob nob)
  (exit nob))
(when (readtablep ob)
  (exit
   (throw '|writifyTag| '|writifyFailed|)))
(when (stringp ob)
  (exit
   (seq
    (when (eq ob |$NullStream|)
      (exit
       (cons 'writified!! (cons 'nullstream nil))))
    (when (eq ob |$NonNullStream|)
      (exit
       (cons 'writified!! (cons 'nonnullstream nil))))
    (exit ob)))))
(when (floatp ob)
  (exit
   (seq
    (when (boot-equal ob (read-from-string (princ-to-string ob)))
      (exit ob))
    (exit
     (cons 'writified!!
      (cons 'float
       (cons ob
        (multiple-value-list (integer-decode-float ob))))))))
  (exit ob))))

---

(defun writify
  (defun |writify| (ob)
    (let (|$seen| |$writifyComplained|)
      (declare (special |$seen| |$writifyComplained|))
      (if (null (|ScanOrPairVec| (|function| |unwritable?|) ob))
        ob
        (progn
          (setq |$seen| (make-hash-table :test #'eq))
          (setq |$writifyComplained| nil)
          (|writify,writifyInner| ob))))

---

(defun writify
  |defun writify |
defun spadClosure?

[ qcar p?? ]
[ bpiname p?? ]
[ qcdr p?? ]
[ vecp p?? ]

(defun spadClosure? (ob)
  (let (fun name vec)
    (setq fun (qcar ob))
    (if (null (setq name (bpiname fun)))
      nil
    (progn
      (setq vec (qcdr ob))
      (if (null (vecp vec))
        nil
        name)))))

defvar $NonNullStream

— initvars —

(defvar |$NonNullStream| "NonNullStream")

defvar $NullStream

— initvars —

(defvar |$NullStream| "NullStream")
defun dewritify,dewritifyInner

| seq p?? | exit p?? |
| hget p1044 | intp p?? |
| gensymmer p?? | error p?? |
| poundsign p?? | hput p1044 |
| [dewritify,dewritifyInner p612] | concat p1047 |
| vmread p?? | [make-instream p981] |
| spaddifference p?? | [qcar p??] |
| [qcdr p??] | [qrplaca p??] |
| [qrplacd p??] | [vecp p??] |
| [qvmaxindex p??] | [qsetvelt p??] |
| [qvelt p??] | [$seen p??] |
| [$NonNullStream p611] | [$NonNullStream p611] |

— defun dewritify,dewritifyInner —

(defun |dewritify,dewritifyInner| (ob)
  (prog (e type oname f vec name tmp1 signif expon sign fval qcar qcdr n nob)
    (declare (special |$seen| |$NullStream| |$NonNullStream|))
    (return
      (seq
        (when (null ob)
          (exit nil))
        (when (setq e (hget |$seen| ob))
          (exit e))
        (when (and (consp ob) (eq (car ob) 'writified!!))
          (exit
            (seq
              (setq type (elt ob 1))
              (when (eq type 'self)
                (exit 'writified!!))
              (when (eq type 'bpi)
                (exit
                  (seq

                )
              ))
            ))
          )
        ))
      ))
    )
  )
)
(setq oname (elt ob 2))
(setq f
  (seq
    (when (integerp oname) (exit (eval (gensymmer oname)))))
    (exit (symbol-function oname)))
  (when (null (compiled-function-p f))
    (exit (|error| "A required BPI does not exist.
")))
  (when (and (> (|#| ob) 3) (not (equal (sxhash f) (elt ob 3)))))
    (exit (|error| "A required BPI has been redefined.
")))
  (hput |$seen| ob f)
  (exit f))))

(when (eq type 'hashtable)
  (exit
    (seq
      (setq nob (make-hash-table :test #'equal))
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (do ((tmp0 (elt ob 3) (cdr tmp0))
          (k nil)
          (tmp1 (elt ob 4) (cdr tmp1))
          (e nil))
          ((or (atom tmp0)
               (progn
                 (setq k (car tmp0))
                 nil)
              (atom tmp1)
              (progn
                (setq e (car tmp1))
                nil))
        nil)
        nil)
      (seq
        (exit
          (hput nob (|dewritify,dewritifyInner| k)
            (|dewritify,dewritifyInner| e))))
      (exit nob)))))

(when (eq type 'devaluated)
  (exit
    (seq
      (setq nob (eval (|dewritify,dewritifyInner| (elt ob 2))))
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (exit nob)))))

(when (eq type 'spadclosure)
  (exit
    (seq
      (setq vec (|dewritify,dewritifyInner| (elt ob 2)))
      (setq name (ELT ob 3))
      (when (null (fboundp name))
        (exit
          (|error|)
(concat "undefined function: " (symbol-name name))))
(setq nob (cons (symbol-function name) vec))
(hput |$seen| ob nob)
(hput |$seen| nob nob)
(exit nob))))

(when (eq type 'place)
  (exit
   (seq
    (setq nob (vmread (make-instream nil)))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (exit nob))))

(when (eq type 'readtable)
  (exit (|error| "Cannot de-writify a read table."))) 

(when (eq type 'nullstream)
  (exit |$NullStream|))

(when (eq type 'nonnullstream)
  (exit |$NonNullStream|))

(when (eq type 'float)
  (exit
   (seq
    (progn
     (setq tmp1 (cddr ob))
     (setq fval (car tmp1))
     (setq signif (cadr tmp1))
     (setq expon (caddr tmp1))
     (setq sign (cadddr tmp1))
     tmp1)
     (setq fval (scale-float (float signif fval) expon))
     (when (minusp sign)
       (exit (spadddifference fval)))
     (exit fval)))
     (exit (|error| "Unknown type to de-writify."))) ))

(when (consp ob)
  (exit
   (seq
    (setq qcar (qcar ob))
    (setq qcdr (qcdr ob))
    (setq nob (cons qcar qcdr))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (qrplaca nob (|dewritify,dewritifyInner| qcar))
    (qrplacd nob (|dewritify,dewritifyInner| qcdr))
    (exit nob))))

(when (vecp ob)
  (exit
   (seq
    (setq n (qvmaxindex ob))
    (setq nob (make-array (1+ n)))
    (hput |$seen| ob nob)
    (setq nob (cons (symbol-function name) vec))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (exit nob))))
34.5. LISPLIB OUTPUT TRANSFORMATIONS

```lisp
(defun dewritify (ob)
  (let (|$seen|
        (declare (special |$seen|))
        (if (null (|ScanOrPairVec| #'(lambda (a) (eq a 'writified!!)) ob))
            ob
            (progn
              (setq |$seen| (make-hash-table :test #'eq))
              (|dewritify,dewritifyInner| ob))))))

——

defun ScanOrPairVec,ScanOrInner

[|ScanOrPairVecAnswer p616|
  [function p??]
  [dewritify,dewritifyInner p612]
  [$seen p??]

—— defun ScanOrPairVec,ScanOrInner ——
```
(defun |ScanOrPairVec,ScanOrInner| (f ob)
  (declare (special |$seen|))
  (when (hget |$seen| ob) nil)
  (when (consp ob)
    (hput |$seen| ob t)
    (|ScanOrPairVec,ScanOrInner| f (qcar ob))
    (|ScanOrPairVec,ScanOrInner| f (qcdr ob)))
  (when (vecp ob)
    (hput |$seen| ob t)
    (do ((tmp0 (spaddifference (|#| ob) 1)) (i 0 (1+ i)))
        (> i tmp0) nil)
      (|ScanOrPairVec,ScanOrInner| f (elt ob i)))
  (when (funcall f ob) (throw '|ScanOrPairVecAnswer| t))
  nil)

---

defun ScanOrPairVec

[ScanOrPairVecAnswer p??]
[ScanOrPairVec,ScanOrInner p615]
[$seen p??]

   — defun ScanOrPairVec —

(defun |ScanOrPairVec| (f ob)
  (let (|$seen|)
    (declare (special |$seen|))
    (setq |$seen| (make-hash-table :test #'eq))
    (catch '|ScanOrPairVecAnswer| (|ScanOrPairVec,ScanOrInner| f ob))))

---

defun gensymInt

[gensymp p??]
[error p??]
[pname p1045]
[charDigitVal p617]

   — defun gensymInt —

(defun |gensymInt| (g)
  (let (p n)
    (if (null (gensymp g))

(|error| "Need a GENSYM")
(progn
  (setq p (pname g))
  (setq n 0)
  (do ((tmp0 (spaddifference (|#| p) 1)) (i 2 (1+ i)))
      ((> i tmp0) nil)
    (setq n (+ (* 10 n) (|charDigitVal| (elt p i)))))
  n)))

defun charDigitVal

[spaddifference p??]
[error p??]

— defun charDigitVal —

(defun |charDigitVal| (c)
  (let (digits n)
    (setq digits "0123456789")
    (setq n (spaddifference 1))
    (do ((tmp0 (spaddifference (|#| digits) 1)) (i 0 (1+ i)))
        ((or (> i tmp0) (null (minusp n))) nil)
      (if (char= c (elt digits i))
        (setq n i)
        nil))
    (if (minusp n)
      (|error| "Character is not a digit")
      n)))

— defun histFileErase —

(defun histFileErase (file)
  (when (probe-file file) (delete-file file)))
34.6 History File Messages

— History File Messages —

S2IH0001
You have not reached step %1b yet, and so its value cannot be supplied.
S2IH0002
Cannot supply value for step %1b because 1 is the first step.
S2IH0003
Step %1b has no value.
S2IH0004
The history facility is not on, so you cannot use %b %k %d .
S2IH0006
You have not used the correct syntax for the %b history %d command.
Issue %b )help history %d for more information.
S2IH0007
The history facility is already on.
S2IH0008
The history facility is now on.
S2IH0009
Turning on the history facility will clear the contents of the workspace.
Please enter %b y %d or %b yes %d if you really want to do this:
S2IH0010
The history facility is still off.
S2IH0011
The history facility is already off.
S2IH0012
The history facility is now off.
S2IH0013
The history facility is not on, so the .input file containing your user input cannot be created.
S2IH0014
Edit %b %1 %d to see the saved input lines.
S2IH0015
The argument %b n %d for %b )history )change n must be a nonnegative integer and your argument, %1b , is not one.
S2IH0016
The history facility is not on, so no information can be saved.
S2IH0018
The saved history file is %1b .
S2IH0019
There is no history file, so value of step %1b is undefined.
S2IH0022
No history information had been saved yet.
S2IH0023
%1b is not a valid filename for the history file.
S2IH0024
History information cannot be restored from %1b because the file does not exist.
S2IH0025
The workspace has been successfully restored from the history file %1b.
S2IH0026
The history facility command %1b cannot be performed because the history facility is not on.
S2IH0027
A value containing a %1b is being saved in a history file or a compiled input file INLIB. This type is not yet usable in other history operations. You might want to issue %b history off %d
S2IH0029
History information is already being maintained in an external file (and not in memory).
S2IH0030
History information is already being maintained in memory (and not in an external file).
S2IH0031
When the history facility is active, history information will be maintained in a file (and not in an internal table).
S2IH0032
When the history facility is active, history information will be maintained in memory (and not in an external file).
S2IH0034
Missing element in internal history table.
S2IH0035
Can’t save the value of step number %1b. You can re-generate this value by running the input file %2b.
S2IH0036
The value specified cannot be saved to a file.
S2IH0037
You must specify a file name to the history save command
S2IH0038
You must specify a file name to the history write command
Chapter 35

)include help page Command

35.1 include help page man page

— include.help —

User Level Required: interpreter

Command Syntax:

)include filename

Command Description:

The )include command can be used in .input files to place the contents of another file inline with the current file. The path can be an absolute or relative pathname.

35.2 Functions

defun ncloopInclude1

[ncloopIncFileName p622]
[ncloopInclude p622]

— defun ncloopInclude1 —
(defun ncloopInclude1 (name n)
  (let (a)
    (if (setq a (ncloopIncFileName name))
        (ncloopInclude a n)
        n)))

---

**Returns the first non-blank substring of the given string**

(defun ncloopIncFileName (string)
  "Returns the first non-blank substring of the given string"
  (let (fn)
    (unless (setq fn (incFileName string))
      (write-line (concat string " not found")))
    fn))

---

**Open the include file and read it in**

The ncloopInclude0 function is part of the parser and lives in int-top.boot. (ncloopInclude0 p71)

(defun ncloopInclude (name n)
  "Open the include file and read it in"
  (with-open-file (st name) (ncloopInclude0 st name n)))

---

**Return the include filename**

Given a string we return the first token from the string which is the first non-blank substring. (incBiteOff p623)

(defun incFileName p622)
[concat p1047]

— defun ncloopIncFileName —

— defun ncloopInclude —

— defun incFileName —
(defun incFileName (x)
  "Return the include filename"
  (car (incBiteOff x)))

---

Return the next token

Takes a sequence and returns the a list of the first token and the remaining string characters. If there are no remaining string characters the second string is of length 0. Effectively it "bites off" the first token in the string. If the string only 0 or more blanks it returns nil.

— defun incBiteOff —

(defun incBiteOff (x)
  "Return the next token"
  (let (blank nonblank)
    (setq x (string x))
    (when (setq nonblank (position #\space x :test-not #'char=))
      (setq blank (position #\space x :start nonblank))
      (if blank
        (list (subseq x nonblank blank) (subseq x blank))
        (list (subseq x nonblank) ""))))

---
Chapter 36

)library help page Command

36.1 library help page man page

— library.help —

====================================================================
A.14. )library
====================================================================

User Level Required: interpreter

Command Syntax:

- )library libName1 [libName2 ...]
- )library )dir dirName
- )library )only objName1 [objlib2 ...]
- )library )noexpose

Command Description:

This command replaces the )load system command that was available in AXIOM releases before version 2.0. The )library command makes available to AXIOM the compiled objects in the libraries listed.

For example, if you )compile dopler.spad in your home directory, issue )library dopler to have AXIOM look at the library, determine the category and domain constructors present, update the internal database with various properties of the constructors, and arrange for the constructors to be automatically loaded when needed. If the )noexpose option has not been given, the constructors will be exposed (that is, available) in the current frame.

If you compiled a file you will have an NRLIB present, for example,
DOPLER.NRLIB, where DOPLER is a constructor abbreviation. The command )library DOPLER will then do the analysis and database updates as above.

To tell the system about all libraries in a directory, use )library )dir dirName where dirName is an explicit directory. You may specify ‘.’ as the directory, which means the current directory from which you started the system or the one you set via the )cd command. The directory name is required.

You may only want to tell the system about particular constructors within a library. In this case, use the )only option. The command )library dopler )only Test1 will only cause the Test1 constructor to be analyzed, autoloaded, etc..

Finally, each constructor in a library are usually automatically exposed when the )library command is used. Use the )noexpose option if you not want them exposed. At a later time you can use )set expose add constructor to expose any hidden constructors.

Note for AXIOM beta testers: At various times this command was called )local and )with before the name )library became the official name.

Also See:
  o )cd
  o )compile
  o )frame
  o )set

---

1 “cd” (?? p ??) “frame” (32.5 p 565) “set” (45.36 p 808)
Chapter 37

)lisp help page Command

37.1  lisp help page man page

— lisp.help —

====================================================================
A.15. )lisp
====================================================================

User Level Required:  development

Command Syntax:

- )lisp [lispExpression]

Command Description:

This command is used by AXIOM system developers to have single expressions evaluated by the Lisp system on which AXIOM is built. The lispExpression is read by the Lisp reader and evaluated. If this expression is not complete (unbalanced parentheses, say), the reader will wait until a complete expression is entered.

Since this command is only useful for evaluating single expressions, the )fin command may be used to drop out of AXIOM into Lisp.

Also See:
o )system
o )boot
o )fin
37.2 Functions

This command is in the list of \texttt{\$noParseCommands} 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function \texttt{handleNoParseCommands} 18.2

\footnote{“system” (77 p 77) “boot” (4.1 p 23) “fin” (31.1 p 548)}
Chapter 38

)load help page Command

38.1 load help page man page

— load.help —

====================================================================
A.16. )load
====================================================================

User Level Required: interpreter

Command Description:

This command is obsolete. Use )library instead.

——-

defun The )load command (obsolete)

We keep this command around in case anyone has the original Axiom book. [sayKeyedMsg p329]

— defun load —

(defun |load| |ignore|
 (declare (ignore ignore))
 (|sayKeyedMsg| 'S2IU0003 nil))

——-

629
Chapter 39

)ltrace help page Command

39.1 ltrace help page man page

— ltrace.help —

====================================================================
A.17. )ltrace
====================================================================

User Level Required: development

Command Syntax:

This command has the same arguments as options as the )trace command.

Command Description:

This command is used by AXIOM system developers to trace Lisp or BOOT
functions. It is not supported for general use.

Also See:
o )boot
o )lisp
o )trace

---

1

1 “boot” (4.1 p 23) “lisp” (?? p ??) “trace” (52.1 p 847)
defun The top level )ltrace function

[trace p$47$

— defun ltrace —

(defun |ltrace| (arg) (|trace| arg))

39.2 Variables Used

39.3 Functions
Chapter 40

)pquit help page Command

40.1 pquit help page man page

— pquit.help —

====================================================================
| pquit.help |
====================================================================

A.18. )pquit
====================================================================

User Level Required: interpreter

Command Syntax:
- )pquit

Command Description:

This command is used to terminate AXIOM and return to the operating system. Other than by redoing all your computations or by using the )history )restore command to try to restore your working environment, you cannot return to AXIOM in the same state.

)pquit differs from the )quit in that it always asks for confirmation that you want to terminate AXIOM (the ‘‘p’’ is for ‘‘protected’’). When you enter the )pquit command, AXIOM responds

Please enter y or yes if you really want to leave the interactive environment and return to the operating system:

If you respond with y or yes, you will see the message

You are now leaving the AXIOM interactive environment.
Issue the command axiom to the operating system to start a new session.

and AXIOM will terminate and return you to the operating system (or the environment from which you invoked the system). If you responded with something other than y or yes, then the message

You have chosen to remain in the AXIOM interactive environment.

will be displayed and, indeed, AXIOM would still be running.

Also See:
  o  )fin
  o  )history
  o  )close
  o  )quit
  o  )system

40.2 Functions

The top level pquit command

[pquitSpad2Cmd p634]

— defun pquit —

(defun pquit ()
 "The top level pquit command"
 (|pquitSpad2Cmd|))

The top level pquit command handler

[quitSpad2Cmd p638]
[$quitCommandType p800]

— defun pquitSpad2Cmd —

(defun pquitSpad2Cmd ()

1 “fin” (31.1 p 548) “history” (34.4 p 582) “close” (24.2 p 510) “quit” (41.2 p 638) “system” (?? p ??)
"The top level pquit command handler"
(let ((|$quitCommandType| '|protected|))
 (declare (special |$quitCommandType|))
 (|quitSpad2Cmd|)))

This command is in the list of $noParseCommands 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2
Chapter 41

)quit help page Command

41.1  quit help page man page

— quit.help —

====================================================================
A.19. )quit
====================================================================

User Level Required: interpreter

Command Syntax:

- )quit
- )set quit protected | unprotected

Command Description:

This command is used to terminate AXIOM and return to the operating system. Other than by redoing all your computations or by using the )history )restore command to try to restore your working environment, you cannot return to AXIOM in the same state.

)quit differs from the )pquit in that it asks for confirmation only if the command

)set quit protected

has been issued. Otherwise, )quit will make AXIOM terminate and return you to the operating system (or the environment from which you invoked the system).

The default setting is )set quit protected so that )quit and )pquit behave in

637
the same way. If you do issue

)set quit unprotected

we suggest that you do not (somehow) assign )quit to be executed when you
press, say, a function key.

Also See:
o )fin
o )history
o )close
o )pquit
o )system

1

41.2 Functions

The top level quit command

[quitSpad2Cmd p638]

— defun quit —

(defun quit ()
 "The top level quit command"
 (quitSpad2Cmd))

The top level quit command handler

[upcase p??]
[queryUserKeyedMsg p??]
[string2id-n p??]
[leaveScratchpad p639]
[sayKeyedMsg p329]
[tersyscommand p452]
[$quitCommandType p800]

— defun quitSpad2Cmd —

1 “fin” (31.1 p 548) “history” (34.4 p 582) “close” (24.2 p 510) “pquit” (40.2 p 634) “system” (?? p ??)
41.2. FUNCTIONS

(defun |quitSpad2Cmd| ()
"The top level quit command handler"
(declare (special |$quitCommandType|))
(if (eq |$quitCommandType| '|protected|)
(let (x)
  (setq x (upcase (|queryUserKeyedMsg| 's2iz0031 nil)))
  (when (member (string2id-n x 1) '(y yes)) (|leaveScratchpad|))
  (|sayKeyedMsg| 's2iz0032 nil)
  (tersyscommand))
  (|leaveScratchpad|)))

---

Leave the Axiom interpreter

— defun leaveScratchpad —

(defun |leaveScratchpad| ()
"Leave the Axiom interpreter"
(bye))

---

This command is in the list of $noParseCommands 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2
Chapter 42

)read help page Command

42.1 read help page man page

— read.help —

====================================================================
A.20. )read
====================================================================

User Level Required: interpreter

Command Syntax:

- )read [fileName]
- )read [fileName] [)quiet] [)ifthere]

Command Description:

This command is used to read .input files into AXIOM. The command

)read matrix.input

will read the contents of the file matrix.input into AXIOM. The ‘‘.input’’
file extension is optional. See the AXIOM User Guide index for more
information about .input files.

This command remembers the previous file you edited, read or compiled. If you
do not specify a file name, the previous file will be read.

The )ifthere option checks to see whether the .input file exists. If it does
not, the )read command does nothing. If you do not use this option and the
file does not exist, you are asked to give the name of an existing .input
file.

The )quiet option suppresses output while the file is being read.

Also See:
- )compile
- )edit
- )history

---

defun The )read command

[readSpad2Cmd p642]

---

---

defun Implement the )read command

[selectOptionLC p479]
[optionError p449]
[pathname p1042]
[pathnameTypeld p1041]
[makePathname p1042]
[pathnameName p1040]
[mergePathnames p1041]
[findfile p??]
[throwKeyedMsg p??]
[namestring p1040]
[upcase p??]
[member p1048]
[/read p644]
[$InteractiveMode p22]
[$findfile p??]
[$UserLevel p807]
[$options p??]
[/editfile p515]

---

1 “edit” (30.2 p 544) “history” (34.4 p 582)
— defun readSpad2Cmd —

(defun readSpad2Cmd (arg)
  (prog (|$InteractiveMode| fullopt ifthere quiet ef devFTs fileTypes
          ll ft upft fs)
    (declare (special |$InteractiveMode| |findfile| |$UserLevel| |$options|
              /editfile))
    (setq |$InteractiveMode| t)
    (dolist (opt |$options|)
      (setq fullopt
            (|selectOptionLC| (caar opt) '|quiet| |test| |ifthere|) '|optionError|))
      (cond
        ((eq fullopt '|ifthere|) (setq ifthere t))
        ((eq fullopt '|quiet|) (setq quiet t)))
    (setq ef (|pathname| /editfile))
    (when (eq (|pathnameTypeId| ef) 'spad)
      (setq ef (|makePathname| (|pathnameName| ef) "* " ""))
    (if arg
        (setq arg (|mergePathnames| (|pathname| arg) ef))
        (setq arg ef))
    (setq devFTs ("input" "INPUT" "boot" "BOOT" "lisp" "LISP"))
    (setq fileTypes
      (cond
        ((eq |$UserLevel| '|interpreter|) ("input" "INPUT"))
        ((eq |$UserLevel| '|compiler|) ("input" "INPUT"))
        (t devFTs))
    (setq ll ($findfile arg fileTypes))
    (unless ll
      (if ifthere
          (return nil)
          (|throwKeyedMsg| 'S2IL0003 (list (|namestring| arg))))))
    (setq ll (|pathname| ll))
    (setq ft (|pathnameType| ll))
    (setq upft (upcase ft))
    (cond
      ((null (|member| upft fileTypes))
        (setq fs (|namestring| arg))
        (if (|member| upft devFTs)
          (|throwKeyedMsg| 'S2IZ0033 (list fs))
          (|throwKeyedMsg| 'S2IZ0034 (list fs)))
      (t
        (setq /editfile ll)
        (when (string= upft "BOOT") (setq |$InteractiveMode| nil))
        (/read ll quiet))))
defun /read

[/read /rf (vol9)]
[/read /rq (vol9)]
[/editfile p515]

---

(defun /read (l q)
  (declare (special /editfile))
  (setq /editfile l)
  (cond
    (q (/rq))
    (t (/rf))
  )
  (flag |boot-NewKEY| 'key)
  (!terminateSystemCommand!)
  (!spadPrompt!))

---
Chapter 43

)regress help page Command

43.1  regress help page man page

— regress.help —

====================================================================
A.18. )regress
====================================================================

User Level Required:  interpreter

Command Syntax:

- )regress fileName

Command Description:

The regress command will run the regress function that was compiled as part of the lisp image build process. This function expects an input filename, possibly containing a path prefix.

If the filename contains a period then we consider it a fully formed filename, otherwise we append ‘’.output’’, which is the default file extension.

)regress matrix
)regress matrix.output
)regress /path/to/file/matrix
)regress /path/to/file/matrix.output

will test the contents of the file matrix.output.
The idea behind regression testing is to check that the results we currently get match the results we used to get. In order to do that we create input files with a special comment format that contains the prior results. These are easy to create as all you need to do is run the Axiom function, capture the results, and turn them input specially formed comments using the -- comment.

A regression file caches the result of an Axiom function so we can automate the testing process. It is a file of many tests, each with their own output.

The regression file format uses the Axiom -- comment syntax to keep a copy of the expected output from an Axiom command. This expected output is compared character by character against the actual output.

The regression file is broken into numbered blocks, delimited by a --S for the beginning and a --E for the end. The total number of blocks is also given so missing or failed tests also raise an error.

There are 4 special kinds of -- comments in regression files:

--S n of M  this is test n of M tests in this file
--E n  this marks the end of test n
--R any output  this marks the actual expected output line
--I any output  this line is compared but ignored

A regression test file looks like:

```axiom
)set break resume
)set message type off
)clear all

--S 1 of 3
2+3
---R this is the exact Axiom output
---R (1) 5
---E 1

--S 2 of 3
2+3
---R this should fail to match
---R (2) 7
---E 2

--S 3 of 3
2+3
---R this fails to match but we
---I (3) 7 use --I to ignore this line
---E 3
```
We can now run this file with

)read foo.input

Note that when this file is run it will create a spool file called "foo.output" because of the lines:

)spool foo.output
)spool

The "foo.output" file contains the console image of the result. It will look like:

)set message type off
)clear all

--S 1 of 3
2+3

(1) 5
--R
--R (1) 5
--E 1

--S 2 of 3
2+3

(2) 5
--R
--R (2) 7
--E 2

--S 3 of 3
2+3

(3) 5
--R
--I (3) 7
--E 3

)spool

This "foo.output" file can now be checked using the )regress command.

When we run the )regress foo.output we see;

testing foo
passed foo 1 of 3
CHAPTER 43: )REGRESS HELP PAGE COMMAND

MISMATCH
expected: " (2) 7"
got: " (2) 5"
FAILED foo 2 of 2
passed foo 3 of 3
regression result FAILED 1 of 3 stanzas file foo

Tests either pass or fail. A passing test generates the message:

passed foo 1 of 3

A failing test will give a reversed printout of the expected vs actual output as well as a FAILED message, as in:

MISMATCH
expected: " (2) 7"
got: " (2) 5"
FAILED foo 2 of 3

The last line of output is a summary:

regression result FAILED 1 of 3 stanzas file foo

-----

— defun regress —

(defun regress (arg)
  (let (($InteractiveMode| namestring dot1 outfile (extension "output"))
    (declare (special $InteractiveMode|))
    (setq $InteractiveMode| t)
    (setq namestring (symbol-name (car arg)))
    (setq dot1 (position "." namestring))
    (unless dot1
      (setq outfile (concatenate 'string (subseq namestring 0) "." extension)))
    (if (probe-file outfile)
      (regress outfile)
      (format t (concatenate 'string outfile " file not found"))))

-----
Chapter 44

)\savesystem help page Command

44.1 \savesystem help page man page

--- \savesystem.help ---

A.8. \savesystem

User Level Required: interpreter

Command Syntax:

- \savesystem filename

Command Description:

This command is used to save an AXIOM image to disk. This creates an executable file which, when started, has everything loaded into it that was there when the image was saved. Thus, after executing commands which cause the loading of some packages, the command:

\savesystem /tmp/savesys

will create an image that can be restarted with the UNIX command:

axiom -ws /tmp/savesys
This new system will not need to reload the packages and domains that were already loaded when the system was saved.

There is currently a restriction that only systems started with the command "AXIOMsys" may be saved.

---

defun The )savesystem command

[helpSpad2Cmd p572]
[spad-save p989]

— defun savesystem —

(defun |savesystem| (arg)
  (if (or (not (eql (cons |#| arg) 1)) (null (symbolp (car arg))))
      (|helpSpad2Cmd| '(|savesystem|))
      (spad-save (symbol-name (car arg)))))

---
Chapter 45

)set help page Command

45.1 set help page man page

— set.help —

====================================================================
A.21. )set
====================================================================

User Level Required: interpreter

Command Syntax:

- )set
- )set label1 [... labelN]
- )set label1 [... labelN] newValue

Command Description:

The )set command is used to view or set system variables that control what
messages are displayed, the type of output desired, the status of the history
facility, the way AXIOM user functions are cached, and so on. Since this
collection is very large, we will not discuss them here. Rather, we will show
how the facility is used. We urge you to explore the )set options to
familiarize yourself with how you can modify your AXIOM working environment.
There is a HyperDoc version of this same facility available from the main
HyperDoc menu. Click [here] to go to it.

The )set command is command-driven with a menu display. It is
tree-structured. To see all top-level nodes, issue )set by itself.

)set
Variables with values have them displayed near the right margin. Subtrees of selections have "..." displayed in the value field. For example, there are many kinds of messages, so issue )set message to see the choices.

)set message

The current setting for the variable that displays whether computation times are displayed is visible in the menu displayed by the last command. To see more information, issue

)set message time

This shows that time printing is on now. To turn it off, issue

)set message time off

As noted above, not all settings have so many qualifiers. For example, to change the )quit command to being unprotected (that is, you will not be prompted for verification), you need only issue

)set quit unprotected

Also See: o )quit

1

45.2 Overview

This section contains tree of information used to initialize the )set command in the interpreter. The current list is:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile</td>
<td>Library compiler options</td>
<td>...</td>
</tr>
<tr>
<td>breakmode</td>
<td>execute break processing on error</td>
<td>break</td>
</tr>
<tr>
<td>expose</td>
<td>control interpreter constructor exposure</td>
<td>...</td>
</tr>
<tr>
<td>functions</td>
<td>some interpreter function options</td>
<td>...</td>
</tr>
<tr>
<td>fortran</td>
<td>view and set options for FORTRAN output</td>
<td>...</td>
</tr>
<tr>
<td>kernel</td>
<td>library functions built into the kernel for efficiency</td>
<td>...</td>
</tr>
<tr>
<td>hyperdoc</td>
<td>options in using HyperDoc</td>
<td>...</td>
</tr>
</tbody>
</table>

1 "quit" (41.2 p 638)
45.3 Variables Used

45.4 Functions

Initialize the set variables

The argument settree is initially the $setOption variable. The fourth element is a union-style switch symbol. The fifth element is usually a variable to set. The sixth element is a subtree to recurse for the TREE switch. The seventh element is usually the default value. For more detailed explanations see the list structure section 45.5. [sayMSG p331] [literals p??] [translateYesNo2TrueFalse p658] [tree p??] [initializeSetVariables p653] — defun initializeSetVariables —

(defun |initializeSetVariables| (settree) "Initialize the set variables" (dolist (setdata settree) (case (fourth setdata) (function (if (canFuncall? (fifth setdata)) (funcall (fifth setdata) '|%initialize%|) (|sayMSG| (concatenate 'string " Function not implemented. " (package-name *package*) ":" (string (fifth setdata))))) (integer (set (fifth setdata) (seventh setdata)))) (string (set (fifth setdata) (seventh setdata)))) (literals (set (fifth setdata) (translateYesNo2TrueFalse (seventh setdata)))) (tree (|initializeSetVariables| (sixth setdata))))

Variables with current values of ... have further sub-options. For example, issue )set system to see what the options are for system. For more information, issue )help set.
CHAPTER 45. \textit{SET HELP PAGE COMMAND}

---

Reset the workspace variables

\begin{boxedverbatim}
| defun resetWorkspaceVariables |
| (defun \textit{resetWorkspaceVariables} () |
| "Reset the workspace variables"
| (declare (special /countlist /editfile /sourcefiles /$sourceFiles/ /pretty |
| /spacelist /timerlist /$existingFiles/ /functionTable |$boot |
| /compileMapFlag /echoLineStack /operationNameList /$slamFlag |
| /CommandSynonymAlist /InitialCommandSynonymAlist /UserAbbreviationsAlist |
| /msgAlist /msgDatabase /msgDatabaseName |
| /dependeeClosureAlist /IOindex |
| /coerceIntByMapCounter /e |
| /env |
| /setOptions))
\end{boxedverbatim}
45.4. FUNCTIONS

|setq |/countlist nil|
|setq |/editfile nil|
|setq |/sourcefiles nil|
|setq |$sourceFiles| nil|
|setq |/pretty nil|
|setq |/spacelist nil|
|setq |/timerlist nil|
|setq |$existingFiles| (make-hash-table :test #'equal)|
|setq |$functionTable| nil|
|setq |$boot nil|
|setq |$compileMapFlag| nil|
|setq |$echoLineStack| nil|
|setq |$operationNameList| nil|
|setq |$slamFlag| nil|
|setq |$CommandSynonymAlist| (copy |$InitialCommandSynonymAlist|)|
|setq |$UserAbbreviationsAlist| nil|
|setq |$msgAlist| nil|
|setq |$msgDatabase| nil|
|setq |$msgDatabaseName| nil|
|setq |$dependeeClosureAlist| nil|
|setq |$IOindex| 1|
|setq |$coerceIntByMapCounter| 0|
|setq |$e| (cons (cons nil nil) nil)|
|setq |$env| (cons (cons nil nil) nil)|

|initializeSetVariables| |$setOptions|)

Display the set option information

[displaySetVariableSettings p657]
[centerAndHighlight p??]
[concat p1047]
[object2String p??]
[specialChar p980]
[sayBrightly p??]
[bright p??]
[sayMSG p331]
[boot-equal p??]
[sayMessage p??]
[eval p??]
[literals p??]
[translateTrueFalse2YesNo p659]
[$linelength p774]

— defun displaySetOptionInformation —
(defun |displaySetOptionInformation| (arg setdata)
  "Display the set option information"
  (let (current)
    (declare (special $linelength))
    (cond
      ((eq (fourth setdata) 'tree)
       (|displaySetVariableSettings| (sixth setdata) (first setdata)))
      (t
       (|centerAndHighlight|
        (concat "The " (|object2String| arg) " Option")
        $linelength (|specialChar| '|hbar|))
       (|sayBrightly|
        '([%l] ,@(|bright| "Description:"), (second setdata)))
      (case (fourth setdata)
        (function
         (terpri)
         (if (canFuncall? (fifth setdata))
          (funcall (fifth setdata) '|%describe%|)
          (|sayMSG| " Function not implemented.")
         )
        (integer
         (|sayMessage|
          '('" The" ,@(|bright| arg) "option"
           " may be followed by an integer in the range"
           ,@(|bright| (elt (sixth setdata) 0)) "to"
           ,@(|bright| (elt (sixth setdata) 1)) "inclusive."
           " The current setting is" ,@(|bright| (|eval| (fifth setdata))))
         )
        (string
         (|sayMessage|
          '('" The" ,@(|bright| arg) "option"
           " is followed by a string enclosed in double quote marks."  
           '([%l] " The current setting is"
             ,@(|bright| (list '|"| (|eval| (fifth setdata)) '|"|))
            )
         )
        (literals
         (|sayMessage|
          '('" The" ,@(|bright| arg) "option"
           " may be followed by any one of the following:"))
        (setq current
         (|translateTrueFalse2YesNo| (|eval| (fifth setdata))))
        (dolist (name (sixth setdata))
         (if (boot-equal name current)
          (|sayBrightly| '( " ->" ,@(|bright| (|object2String| name))))
          (|sayBrightly| (list " " (|object2String| name))))
        )
       )))
  )))
Display the set variable settings

```lisp
(defun display-set-variable-settings (settree label)
  "Display the set variable settings"
  (let (setoption opt subtree subname)
    (declare (special $linelength))
    (if (eq label '||)
      (setq label ")set")
      (setq label (concat " " (object2string label) " ")))
    (centerAndHighlight
      (concat "Current Values of" label " Variables") $linelength '| |)
    (terpri)
    (sayBrightly
      (list "Variable " "Description "
        "Current Value " ))
    (say (|fillerSpaces| $linelength (|specialChar| '|hbar|)))
    (setq subtree nil)
    (dolist (setdata settree)
      (when (|satisfiesUserLevel| (third setdata))
        (setq setoption (object2string (first setdata)))
        (setq setoption
          (concat setoption
            (|fillerSpaces| (spaddifference 13 (|#| setoption)) " ")
            (second setdata)))
        (setq setoption
          (concat setoption
            (|fillerSpaces| (spaddifference 55 (|#| setoption)) " ")))
        (case (fourth setdata)
```
(function
(setq opt
(if (canFuncall? (fifth setdata))
(funcall (fifth setdata) '|%display%|
"unimplemented"))
(cond
((consp opt)
(setq opt
(do ((t2 opt (cdr t2)) t1 (o nil))
  ((or (atom t2) (progn (setq o (car t2)) nil)) t1)
  (setq t1 (append t1 (cons o (cons " " nil)))))
(|sayBrightly| (|concat| setoption '|%b| opt '|%d|)))
(string
(setq opt (|object2String| (|eval| (fifth setdata))))
(|sayBrightly| `(,setoption ,@(|bright| opt))))
(integer
(setq opt (|object2String| (|eval| (fifth setdata))))
(|sayBrightly| `(,setoption ,@(|bright| opt))))
(literals
(setq opt (|object2String|
  (|translateTrueFalse2YesNo| (|eval| (fifth setdata))))))
(|sayBrightly| `(,setoption ,@(|bright| opt))))
(TREE
  (|sayBrightly| `(,setoption ,@(|bright| "...")))
  (setq subtree t)
  (setq subname (|object2String| (first setdata))))))
(terpri)
(when subtree
  (|sayBrightly|
    "Variables with current values of" ,@(|bright| "...")
    "have further sub-options. For example,"))
  (|sayBrightly|
    "issue" ,@(|bright| ")set ") ,subname
    " to see what the options are for" ,@(|bright| subname "."
    |%l| "For more information, issue" ,@(|bright| ")help set") ".")))))

Translate options values to t or nil

[member p1048]

— defun translateYesNo2TrueFalse —

(defun |translateYesNo2TrueFalse| (x)
"Translate options values to t or nil"
(cond
  ((|member| x '(|yes| |on|)) t)
(\(\text{member}\ x \ '([\text{no}] \ [\text{off}]])\) nil)
(t x))

---

Translate t or nil to option values

--- defun translateTrueFalse2YesNo ---

(defun translateTrueFalse2YesNo (x)
"Translate t or nil to option values"
(cond
((eq x t) '|on|)
((null x) '|off|)
(t x))

---

45.5 The list structure

The structure of each list item consists of 7 items. Consider this example:

(userlevel
 "operation access level of system user"
 interpreter
 LITERALS
 $UserLevel
 (interpreter compiler development)
 development)

The list contains (the names in bold are accessor names that can be found in property.lisp.pamphlet[1]. Look for "setName").:

1 Name the keyword the user will see. In this example the user would say ")set output userlevel".

2 Label the message the user will see. In this example the user would see "operation access level of system user".

3 Level the level where the command will be accepted. There are three levels: interpreter, compiler, development. These commands are restricted to keep the user from causing damage.
4 Type a symbol, one of FUNCTION, INTEGER, STRING, LITERALS, FILENAME or TREE.

5 Var

FUNCTION is the function to call
INTEGER is the variable holding the current user setting.
STRING is the variable holding the current user setting.
LITERALS variable which holds the current user setting.
FILENAME is the variable that holds the current user setting.
TREE

6 Leaf

FUNCTION is the list of all possible values
INTEGER is the range of possible values
STRING is a list of all possible values
LITERALS is a list of all of the possible values
FILENAME is the function to check the filename
TREE

7 Def is the default value

FUNCTION is the default setting
INTEGER is the default setting
STRING is the default setting
LITERALS is the default setting
FILENAME is the default value
TREE

45.6 breakmode

------------------------ The breakmode Option ------------------------

Description: execute break processing on error

The breakmode option may be followed by any one of the following:

nobreak

-> break
query
resume
fastlinks
quit

The current setting is indicated.

defvar $BreakMode

    — initvars —

    (defvar $BreakMode 'nobreak "execute break processing on error")

    — breakmode —

    (breakmode
     "execute break processing on error"
    )

45.7 debug

Current Values of debug Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambdatype</td>
<td>Show type information for #1 syntax</td>
<td>off</td>
</tr>
<tr>
<td>dalymode</td>
<td>Interpret leading open paren as lisp</td>
<td>off</td>
</tr>
</tbody>
</table>

    — debug —

    (debug
     "debug options"
    )
45.8 debug lambda type

---------------------- The lambdatype Option ----------------------

Description: Show type information for #1 syntax

defvar $lambdatype

--- initvars ---

(defun $lambdatype nil "show type information for #1 syntax")

---

45.9 debug dalymode

The $dalymode variable is used in a case statement in intloopReadConsole. This variable can be set to any non-nil value. When not nil the interpreter will send any line that begins with an "(" to be sent to the underlying lisp. This is useful for debugging Axiom. The normal value of this variable is NIL.

This variable was created as an alternative to prefixing every lisp command with )lisp. When doing a lot of debugging this is tedious and error prone. This variable was created to shortcut
that process. Clearly it breaks some semantics of the language accepted by the interpreter as parens are used for grouping expressions.

---------------------- The dalymode Option ----------------------

Description: Interpret leading open paren as lisp

```
defvar $dalymode

| initvars |
(defvar $dalymode nil "Interpret leading open paren as lisp")
```

```
| debugdalymode |
($dalymode
  "Interpret leading open paren as lisp"
| interpreter|
LITERALS
$dalymode
(|on| |off|)
[off])
```

### 45.10 compile

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>library in which to place compiled code</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>controls libraries from which to load compiled code</td>
<td></td>
</tr>
</tbody>
</table>

```
| compile |
(|compiler|
  "Library compiler options"
)
45.11 compile output

---------------------- The output Option ----------------------

Description: library in which to place compiled code

— compileoutput —

\begin{verbatim}
(defun setOutputLibrary
  (\$outputLibraryName)
  \texttt{null}
)
\end{verbatim}

45.12 Variables Used

45.13 Functions

The set output command handler

\begin{verbatim}
(defvar \$outputLibraryName)
(defun describeOutputLibraryArgs
  (\$outputLibraryName)
)
(defun openOutputLibrary
  (\$outputLibraryName)
)
\end{verbatim}
(defun |setOutputLibrary| (arg)
  "The set output command handler"
  (let (fn)
    (declare (special |$outputLibraryName|))
    (cond
      ((eq arg '|%initialize%|) (setq |$outputLibraryName| nil))
      ((eq arg '|%display%|) (or |$outputLibraryName| "user.lib"))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?) (/= (|#| arg) 1))
        (|describeOutputLibraryArgs|))
      (t
        (when (probe-file (setq fn (princ-to-string (car arg))))
          (setq fn (truename fn)))
        (|openOutputLibrary| (setq |$outputLibraryName| fn))))))

Describe the set output library arguments

[sayBrightly p??]

— defun describeOutputLibraryArgs —

(defun describeOutputLibraryArgs ()
  "Describe the set output library arguments"
  (|sayBrightly| (list
    '|%b| "set compile output library"
    '|%d| "is used to tell the compiler where to place"
    '|%l| "compiled code generated by the library compiler. By default it goes"
    '|%l| "in a file called"
    '|%b| "user.lib"
    '|%d| "in the current directory.")
)

defvar $output-library

— initvars —

(defvar output-library nil)
Open the output library

The input-libraries and output-library are now truename based. [dropInputLibrary p669] [output-library p665] [input-libraries p668]

— defun openOutputLibrary —

(defun openOutputLibrary (lib)
  "Open the output library"
  (declare (special output-library input-libraries))
  (dropInputLibrary lib)
  (setq output-library (truename lib))
  (push output-library input-libraries))

45.14 compile input

---------------------- The input Option ----------------------

Description: controls libraries from which to load compiled code

)set compile input add library is used to tell AXIOM to add library to the front of the path which determines where compiled code is loaded from.
)set compile input drop library is used to tell AXIOM to remove library from this path.

— compileinput —

(defun input
  "controls libraries from which to load compiled code"
  (interpreter)
  FUNCTION
  setInputLibrary
  NIL
  (htSetInputLibrary))
45.15 Variables Used

45.16 Functions

The set input library command handler

The input-libraries is now maintained as a list of truenames. [describeInputLibraryArgs p668]
[qcar p??]
[qcdr p??]
[selectOptionLC p479]
[addInputLibrary p668]
[dropInputLibrary p669]
[setInputLibrary p667]
[input-libraries p668]

---

(defun setInputLibrary | (arg)
"The set input library command handler"
(declare (special input-libraries))
(let (tmp1 filename act)
  (cond
    ((eq arg '|%initialize%|) t)
    ((eq arg '|%display%|) (mapcar #'namestring input-libraries))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (describeInputLibraryArgs))
    ((and (consp arg)
      (progn
        (setq act (qcar arg))
        (setq tmp1 (qcdr arg))
        (and (consp tmp1)
          (eq (qcdr tmp1) nil)
          (progn (setq filename (qcar tmp1)) t)))))
    (cond
      ((eq act '|add|)
       (addInputLibrary! (truename (princ-to-string filename))))
      ((eq act '|drop|)
       (dropInputLibrary! (truename (princ-to-string filename))))
      (t (setInputLibrary nil))))
---
Describe the set input library arguments

[sayBrightly p??]

— defun describeInputLibraryArgs —

(defun |describeInputLibraryArgs| ()
"Describe the set input library arguments"
(sayBrightly (list
'|%b| "set compile input add library"
'|%d| "is used to tell AXIOM to add"
'|%b| "library"
'|%d| "to"
'|%l| "the front of the path used to find compile code."
'|%l|
'|%b| "set compile input drop library"
'|%d| "is used to tell AXIOM to remove"
'|%b| "library"
'|%d|
'|%l| "from this path.")
))

Add the input library to the list

The input-libraries variable is now maintained as a list of truenames. [dropInputLibrary p669]
[input-libraries p686]

— defun addInputLibrary —

(defun |addInputLibrary| (lib)
"Add the input library to the list"
(declare (special input-libraries))
(dropInputLibrary lib)
(push (truename lib) input-libraries)
)

defvar $input-libraries

— initvars —

(defvar input-libraries nil)
45.17. EXPOSE

---

Drop an input library from the list
[input-libraries p668]
— defun dropInputLibrary —

(defun dropInputLibrary (lib)
"Drop an input library from the list"
(declare (special input-libraries))
(setq input-libraries (delete (truename lib) input-libraries :test #'equal)))

---

45.17 expose

-------------------- The expose Option --------------------

Description: control interpreter constructor exposure

The following groups are explicitly exposed in the current frame (called initial):
  basic
categories
  naglink
  anna

The following constructors are explicitly exposed in the current frame:
  there are no explicitly exposed constructors

The following constructors are explicitly hidden in the current frame:
  there are no explicitly hidden constructors

When )set expose is followed by no arguments, the information you now see is displayed. When followed by the initialize argument, the exposure group data in the file interp.exposed is read and is then available. The arguments add and drop are used to add or drop exposure groups or explicit constructors from the local frame exposure data. Issue
  )set expose add or )set expose drop
for more information.
— expose —

(expose
  "control interpreter constructor exposure"
  (INTERPRETER)
  (FUNCTION
    (SET-EXPOSE)
    NIL
    (HT-SET-EXPOSE))

45.18 Variables Used

NOTE: If you add new algebra you must also update this list otherwise the new algebra won’t be loaded by the interpreter when needed.

defvar $globalExposureGroupAlist

— initvars —

(defvar $globalExposureGroupAlist
  '(
    ;; define the groups |basic| |naglink| |anna| |categories| |Hidden| |defaults|
    (|basic|
      (|AffineAlgebraicSetComputeWithGroebnerBasis| . AFALGGRG)
      (|AffineAlgebraicSetComputeWithResultant| . AFALGRES)
      (|AffinePlane| . AFFPL)
      (|AffinePlaneOverPseudoAlgebraicClosureOfFiniteField| . AFFPLPS)
      (|AffineSpace| . AFFSP)
      (|AlgebraicManipulations| . ALGMANIP)
      (|AlgebraicNumber| . AN)
      (|AlgFactor| . ALGFACCT)
      (|AlgebraicMultiFact| . ALGFACT)
      (|AlgebraPackage| . ALGPKG)
      (|AlgebraGivenByStructuralConstants| . ALGSC)
      (|Any| . ANY)
      (|AnyFunctions1| . ANY)
      (|ApplicationProgramInterface| . API)
      (|ArrayStack| . ASTACK)
      (|AssociatedJordanAlgebra| . JORDAN)
      (|AssociatedLieAlgebra| . LIE)
      (|AttachPredicates| . PMPRED)
      (|AxiomServer| . AXSERV)
      (|BalancedBinaryTree| . BBTREE)
45.18. VARIABLES USED

(|BasicStochasticDifferential| . BSD)
(|BasicOperator| . BOP)
(|BasicOperatorFunctions1| . BOP1)
(|Bezier| . BEZIER)
(|BinaryExpansion| . BINARY)
(|BinaryFile| . BINFILE)
(|BinarySearchTree| . BSTREE)
(|BinaryTournament| . BTOURN)
(|BinaryTree| . BTREE)
(|Bits| . BITS)
(|BlasLevelOne| . BLAS1)
(|BlowUpPackage| . BLUPPACK)
(|BlowUpWithHamburgerNoether| . BLHN)
(|BlowUpWithQuadTrans| . BLQT)
(|Boolean| . BOOLEAN)
(|CardinalNumber| . CARD)
(|CartesianTensor| . CARTEN)
(|CartesianTensorFunctions2| . CARTEN2)
(|Character| . CHAR)
(|CharacterClass| . CCLASS)
(|CharacteristicPolynomialPackage| . CHARPOL)
(|CliffordAlgebra| . CLIF)
(|Color| . COLOR)
(|CommonDenominator| . CDEN)
(|Commutator| . COMM)
(|Complex| . COMPLEX)
(|ComplexDoubleFloatMatrix| . CDFMAT)
(|ComplexDoubleFloatVector| . CDFVEC)
(|ComplexFactorization| . COMPFACT)
(|ComplexFunctions2| . COMPLEX2)
(|ComplexRootPackage| . CMPLXRT)
(|ComplexTrigonometricManipulations| . CTRIGMNP)
(|ContinuedFraction| . CONTFRAC)
(|CoordinateSystems| . COORDSYS)
(|CRAPackage| . CRAPACK)
(|CycleIndicators| . CYCLES)
(|Database| . DBASE)
(|DataList| . DLIST)
(|DecimalExpansion| . DECIMAL)
(|DenavitHartenbergMatrix| . DHMATRIX)
(|Dequeue| . DEQUEUE)
(|DesingTree| . DSTREE)
(|DesingTreePackage| . DTP)
(|DiophantineSolutionPackage| . DIOSP)
(|DirichletRing| . DIRRING)
(|DirectProductFunctions2| . DIRPRD2)
(|DisplayPackage| . DISPLAY)
(|DistinctDegreeFactorize| . DDFACT)
(|Divisor| . DIV)
(|DoubleFloat| . DFLOAT)
CHAPTER 45. )SET HELP PAGE COMMAND

(|DoubleFloatMatrix| . DFMAT)
(|DoubleFloatVector| . DFVEC)
(|DoubleFloatSpecialFunctions| . DFSFUN)
(|DrawComplex| . DRAWCX)
(|DrawNumericHack| . DRAWHACK)
(|DrawOption| . DROPT)
(|EigenPackage| . EP)
(|ElementaryFunctionDefiniteIntegration| . DEFINTEF)
(|ElementaryFunctionLODESolver| . LOODEEF)
(|ElementaryFunctionODESolver| . ODEEF)
(|ElementaryFunctionSign| . SIGNEF)
(|ElementaryFunctionStructurePackage| . EFSTRUC)
(|Equation| . EQ)
(|EquationFunctions2| . EQ2)
(|ErrorFunctions| . ERROR)
(|EuclideanGroebnerBasisPackage| . GBEUCLID)
(|Exit| . EXIT)
(|Export3D| . EXP3D)
(|Expression| . EXPR)
(|ExpressionFunctions2| . EXPR2)
(|ExpressionSolve| . EXPRSOL)
(|ExpressionSpaceFunctions2| . ES2)
(|ExpressionSpaceODESolver| . EXPRODE)
(|ExpressionToOpenMath| . OMEXPR)
(|ExpressionToUnivariatePowerSeries| . EXPR2UPS)
(|Factored| . FR)
(|FactoredFunctions2| . FR2)
(|FactorisationOverPseudoAlgebraicClosureOfAlgExtOfRationalNumber| . FACTEXT)
(|FactorisationOverPseudoAlgebraicClosureOfRationalNumber| . FACTRN)
(|File| . FILE)
(|FileName| . FNAME)
(|FiniteAbelianMonoidRingFunctions2| . FAMR2)
(|FiniteDivisorFunctions2| . FDIV2)
(|FiniteField| . FF)
(|FiniteFieldFactorization| . FFFACTOR)
(|FiniteFieldFactorizationWithSizeParseBySideEffect| . FFFACTSE)
(|FiniteFieldCyclicGroup| . FFCG)
(|FiniteFieldPolynomialPackage2| . FFPOLY2)
(|FiniteFieldNormalBasis| . FFBAS)
(|FiniteFieldHommorphisms| . FFHOM)
(|FiniteFieldSquareFreeDecomposition| . FFSCFR)
(|FiniteLinearAggregateFunctions2| . FLAGG2)
(|FiniteLinearAggregateSort| . FLASORT)
(|FiniteSetAggregateFunctions2| . FSAGG2)
(|FlexibleArray| . FARRAY)
(|Float| . FLOAT)
(|FloatingRealPackage| . FLOATPR)
(|FloatingComplexPackage| . FLOATCP)
(|FourierSeries| . FSERIES)
(|Fraction| . FRAC)
45.18. VARIABLES USED

(|FractionalIdealFunctions2| . FRIDEAL2)
(|FractionFreeFastGaussian| . FFFG)
(|FractionFreeFastGaussianFractions| . FFFGF)
(|FractionFunctions2| . FRAC2)
(|FreeNilpotentLie| . FNLA)
(|FullPartialFractionExpansion| . FPARFRAC)
(|FunctionFieldCategoryFunctions2| . FFCAT2)
(|FunctionSpaceAssertions| . PMASSFS)
(|FunctionSpaceAttachPredicates| . PMPREDFS)
(|FunctionSpaceComplexIntegration| . FSCINT)
(|FunctionSpaceFunctions2| . FS2)
(|FunctionSpaceIntegration| . FSINT)
(|FunctionSpacePrimitiveElement| . FSPRMELT)
(|FunctionSpaceSum| . SUMFS)
(|GaussianFactorizationPackage| . GAUSSFAC)
(|GeneralPackageForAlgebraicFunctionField| . GPAFF)
(|GeneralUnivariatePowerSeries| . GSERIES)
(|GenerateUnivariatePowerSeries| . GENUPS)
(|GnuDraw| . GDRAW)
(|GraphicsDefaults| . GRDEF)
(|GroebnerPackage| . GB)
(|GroebnerFactorizationPackage| . GBF)
(|Guess| . GUESS)
(|GuessAlgebraicNumber| . GUESSAN)
(|GuessFinite| . GUESSF)
(|GuessFiniteFunctions| . GUESSF1)
(|GuessInteger| . GUESSINT)
(|GuessOption| . GOPT)
(|GuessPolynomial| . GUESSP)
(|GuessUnivariatePolynomial| . GUESSUP)
(|HallBasis| . HB)
(|Heap| . HEAP)
(|HexadecimalExpansion| . HEXADEC)
(|HTMLFormat| . HTMLFORM)
(|IdealDecompositionPackage| . IDECOMP)
(|IndexCard| . ICARD)
(|InfiniteClosePoint| . IC)
(|InfiniteProductCharacteristicZero| . INFPRODO)
(|InfiniteProductFiniteField| . INFPRODF)
(|InfiniteProductPrimeField| . INFRODPF)
(|InfiniteTuple| . ITUPLE)
(|InfiniteTupleFunctions2| . ITFUN2)
(|InfiniteTupleFunctions3| . ITFUN3)
(|InfinityClosePointOverPseudoAlgebraicClosureOfFiniteField| . INFCLSPS)
(|Infinity| . INFINITY)
(|Integer| . INT)
(|IntegerCombinatoricFunctions| . COMBINAT)
(|IntegerLinearDependence| . ZLINDEP)
(|IntegerNumberTheoryFunctions| . INTHEORY)
45.18. VARIABLES USED

(|MPolyCatFunctions2| . MPC2)
(|MPolyCatRationalFunctionFactorizer| . MPRFF)
(|Multiset| . MSET)
(|MultivariateFactorize| . MULTFACT)
(|MultivariatePolynomial| . MPOLY)
(|MultFiniteFactorize| . MFINFACT)
(|MyUnivariatePolynomial| . MYUP)
(|MyExpression| . MYEXPR)
(|NeitherSparseOrDensePowerSeries| . NSDPS)
(|NewtonPolygon| . NPOLYGON)
(|NoneFunctions1| . NONE1)
(|NonNegativeInteger| . NNI)
(|NottinghamGroup| . NOTTING)
(|NormalizationPackage| . NORMPK)
(|NormInMonogenicAlgebra| . NORMMA)
(|NumberTheoreticPolynomialFunctions| . NTPOLFN)
(|Numeric| . NUMERIC)
(|NumericalOrdinaryDifferentialEquations| . NUMODE)
(|NumericalQuadrature| . NUMQUAD)
(|NumericComplexEigenPackage| . NCEP)
(|NumericRealEigenPackage| . NREP)
(|NumericContinuedFraction| . NCNTFRAC)
(|Octonion| . OCT)
(|OctonionCategoryFunctions2| . OCTCT2)
(|OneDimensionalArray| . ARRAY1)
(|OneDimensionalArrayFunctions2| . ARRAY12)
(|OnePointCompletion| . ONECOMP)
(|OnePointCompletionFunctions2| . ONECOMP2)
(|OpenMathConnection| . OMCONN)
(|OpenMathDevice| . OMDEV)
(|OpenMathEncoding| . OMENC)
(|OpenMathError| . OMERR)
(|OpenMathErrorKind| . OMERRK)
(|OpenMathPackage| . OMPKG)
(|OpenMathServerPackage| . OMSERVER)
(|OperationsQuery| . OPQUERY)
(|OrderedCompletion| . ORDCOMP)
(|OrderedCompletionFunctions2| . ORDCOMP2)
(|OrdinaryDifferentialRing| . ODR)
(|OrdSetInts| . OSI)
(|OrthogonalPolynomialFunctions| . ORTHPOL)
(|OutputPackage| . OUT)
(|PackageForAlgebraicFunctionField| . PAFF)
(|PackageForAlgebraicFunctionFieldOverFiniteField| . PAFFF)
(|PackageForPoly| . PFORP)
(|PadeApproximantPackage| . PADEPAC)
(|Palette| . PALETTE)
(|PartialFraction| . PFR)
(|PatternFunctions2| . PATTERN2)
(|ParametricPlaneCurve| . PARPCURV)
\{(|ParametricSpaceCurve| . PARSCURV)\)
\{(|ParametricSurface| . PARSURF)\)
\{(|ParametricPlaneCurveFunctions2| . PARPC2)\)
\{(|ParametricSpaceCurveFunctions2| . PARSC2)\)
\{(|ParametricSurfaceFunctions2| . PARSU2)\)
\{(|ParametrizationPackage| . PARAMP)\)
\{(|PartitionsAndPermutations| . PARTPERM)\)
\{(|PatternMatch| . PATMATCH)\)
\{(|PatternMatchAssertions| . PMASS)\)
\{(|PatternMatchResultFunctions2| . PATRES2)\)
\{(|PendantTree| . PENDTREE)\)
\{(|Permanent| . PERMAN)\)
\{(|PermutationGroupExamples| . PGE)\)
\{(|PermutationGroup| . PERMGRP)\)
\{(|Permutation| . PERM)\)
\{(|Pi| . HACKPI)\)
\{(|PiCoercions| . PICOERCE)\)
\{(|Places| . PLACES)\)
\{(|PlacesOverPseudoAlgebraicClosureOfFiniteField| . PLACESPS)\)
\{(|Plcs| . PLCS)\)
\{(|PointFunctions2| . PTFUNC2)\)
\{(|PolyGroebner| . PGROEB)\)
\{(|Polynomial| . POLY)\)
\{(|PolynomialAN2Expression| . PAN2EXPR)\)
\{(|PolynomialComposition| . PCOMP)\)
\{(|PolynomialDecomposition| . PDECOMP)\)
\{(|PolynomialFunctions2| . POLY2)\)
\{(|PolynomialIdeals| . IDEAL)\)
\{(|PolynomialPackageForCurve| . PLPKCRV)\)
\{(|PolynomialToUnivariatePolynomial| . POLY2UP)\)
\{(|PositiveInteger| . PI)\)
\{(|PowerSeriesLimitPackage| . LIMITPS)\)
\{(|PrimeField| . PF)\)
\{(|PrimitiveArrayFunctions2| . PRIMARR2)\)
\{(|PrintPackage| . PRINT)\)
\{(|ProjectiveAlgebraicSetPackage| . PRJALGP)\)
\{(|ProjectivePlane| . PROJPL)\)
\{(|ProjectivePlaneOverPseudoAlgebraicClosureOfFiniteField| . PROJPLPS)\)
\{(|ProjectiveSpace| . PROJSP)\)
\{(|PseudoAlgebraicClosureOfAlgExtOfRationalNumber| . PACEXT)\)
\{(|QuadraticForm| . QFORM)\)
\{(|QuasiComponentPackage| . QCMPACK)\)
\{(|Quaternion| . QUAT)\)
\{(|QuaternionCategoryFunctions2| . QUATCT2)\)
\{(|QueryEquation| . QEQUAT)\)
\{(|Queue| . QUEUE)\)
\{(|QuotientFieldCategoryFunctions2| . QFCAT2)\)
\{(|RadicalEigenPackage| . REP)\)
\{(|RadicalSolvePackage| . SOLVERAD)\)
\{(|RadixExpansion| . RADIX)\)
45.18. VARIABLES USED

(RadixUtilities . RADUTIL)
(RandomNumberSource . RANDSRC)
(RationalFunction . RF)
(RationalFunctionDefiniteIntegration . DEFINTRF)
(RationalFunctionFactor . RFFACT)
(RationalFunctionFactorizer . RFFACTOR)
(RationalFunctionIntegration . INTRF)
(RationalFunctionLimitPackage . LIMITRF)
(RationalFunctionSign . SIGNRF)
(RationalFunctionSum . SUMRF)
(RationalRetractions . RATRET)
(RealClosure . RECLOS)
(RealPolynomialUtilitiesPackage . POLUTIL)
(RealZeroPackage . REALO)
(RealZeroPackageQ . REAL0Q)
(RecurrenceOperator . RECOP)
(RectangularMatrixCategoryFunctions2 . RMCAT2)
(RegularSetDecompositionPackage . RSDCMPK)
(RegularTriangularSet . REGSET)
(RegularTriangularSetGcdPackage . RSETGCD)
(RepresentationPackage1 . REP1)
(RepresentationPackage2 . REP2)
(ResolveLatticeCompletion . RESLATC)
(RewriteRule . RULE)
(RightOpenIntervalRootCharacterization . ROIJC)
(RomanNumeral . ROMAN)
(RootsFindingPackage . RFP)
(Ruleset . RULESET)
(ScriptFormulaFormat . FORMULA)
(ScriptFormulaFormat1 . FORMULA1)
(Segment . SEG)
(SegmentBinding . SEGBIND)
(SegmentBindingFunctions2 . SEGBIND2)
(SegmentFunctions2 . SEG2)
(Set . SET)
(SimpleAlgebraicExtensionAlgFactor . SAEFACT)
(SimplifyAlgebraicNumberConvertPackage . SIMPAN)
(SingleInteger . SINT)
(SmithNormalForm . SMITH)
(SparseUnivariatePolynomialExpressions . SUPEXP)
(SparseUnivariatePolynomialFunctions2 . SUP2)
(SpecialOutputPackage . SPECOUT)
(SquareFreeRegularSetDecompositionPackage . SRDCMPK)
(SquareFreeRegularTriangularSet . SREGSET)
(SquareFreeRegularTriangularSetGcdPackage . SFRGCD)
(SquareFreeQuasiComponentPackage . SFQCMPK)
(Stack . STACK)
(Stream . STREAM)
(StreamFunctions1 . STREAM1)
(StreamFunctions2 . STREAM2)
45.18. VARIABLES USED

(\texttt{|ViewDefaultsPackage| . VIEWDEF})
(\texttt{|Void| . VOID})
(\texttt{|WuWenTsunTriangularSet| . WUTSET})
(\texttt{|naglink|}
  (\texttt{|Asp1| . ASP1})
  (\texttt{|Asp4| . ASP4})
  (\texttt{|Asp6| . ASP6})
  (\texttt{|Asp7| . ASP7})
  (\texttt{|Asp8| . ASP8})
  (\texttt{|Asp9| . ASP9})
  (\texttt{|Asp10| . ASP10})
  (\texttt{|Asp12| . ASP12})
  (\texttt{|Asp19| . ASP19})
  (\texttt{|Asp20| . ASP20})
  (\texttt{|Asp24| . ASP24})
  (\texttt{|Asp27| . ASP27})
  (\texttt{|Asp28| . ASP28})
  (\texttt{|Asp29| . ASP29})
  (\texttt{|Asp30| . ASP30})
  (\texttt{|Asp31| . ASP31})
  (\texttt{|Asp33| . ASP33})
  (\texttt{|Asp34| . ASP34})
  (\texttt{|Asp35| . ASP35})
  (\texttt{|Asp41| . ASP41})
  (\texttt{|Asp42| . ASP42})
  (\texttt{|Asp49| . ASP49})
  (\texttt{|Asp50| . ASP50})
  (\texttt{|Asp55| . ASP55})
  (\texttt{|Asp73| . ASP73})
  (\texttt{|Asp74| . ASP74})
  (\texttt{|Asp77| . ASP77})
  (\texttt{|Asp78| . ASP78})
  (\texttt{|Asp80| . ASP80})
(\texttt{|FortranCode| . FC})
(\texttt{|FortranCodePackage| . FCPAK1})
(\texttt{|FortranExpression| . FEXPR})
(\texttt{|FortranMachineTypeCategory| . FMTC})
(\texttt{|FortranMatrixCategory| . FMC})
(\texttt{|FortranMatrixFunctionCategory| . FMFUN})
(\texttt{|FortranOutputStackPackage| . FUP})
(\texttt{|FortranPackage| . FURT})
(\texttt{|FortranProgramCategory| . FORTCAT})
(\texttt{|FortranProgram| . FORTRAN})
(\texttt{|FortranFunctionCategory| . FORTFN})
(\texttt{|FortranScalarType| . FST})
(\texttt{|FortranType| . FT})
(\texttt{|FortranTemplate| . FTEM})
(\texttt{|FortranVectorFunctionCategory| . FVFUN})
(\texttt{|FortranVectorCategory| . FVC})
(\texttt{|MachineComplex| . MCMPLX})
518. VARIABLES USED

(|d01TransformFunctionType| . D01TRNS)
(|d01WeightsPackage| . D01WGTS)
(|d02AgentsPackage| . D02AGNT)
(|d02bbfAnnaType| . D02BBFA)
(|d02bhfAnnaType| . D02BHF)
(|d02cjfAnnaType| . D02CJFA)
(|d02ejfAnnaType| . D02EJFA)
(|d03AgentsPackage| . D03AGNT)
(|d03eefAnnaType| . D03EEFA)
(|d03fafAnnaType| . D03FAFA)
(|e04AgentsPackage| . E04AGNT)
(|e04dgfAnnaType| . E04DGFA)
(|e04fdfAnnaType| . E04FDFA)
(|e04gcfAnnaType| . E04GCF)
(|e04jafAnnaType| . E04JFA)
(|e04mbfAnnaType| . E04MBFA)
(|e04nafAnnaType| . E04NAFA)
(|e04ucfAnnaType| . E04UCFA)
(|ExpertSystemContinuityPackage| . ESCONT)
(|ExpertSystemContinuityPackage1| . ESCONT1)
(|ExpertSystemToolsPackage| . ESTOOLS)
(|ExpertSystemToolsPackage1| . ESTOOLS1)
(|ExpertSystemToolsPackage2| . ESTOOLS2)
(|NumericalIntegrationCategory| . NUMINT)
(|NumericalIntegrationProblem| . NIPROB)
(|NumericalODEProblem| . ODEPROB)
(|NumericalOptimizationCategory| . OPTCAT)
(|NumericalOptimizationProblem| . OPTPROB)
(|NumericalPDEProblem| . PDEPROB)
(|ODEIntensityFunctionsTable| . ODEIFTBL)
(|IntegrationFunctionsTable| . INTFTBL)
(|OrdinaryDifferentialEquationsSolverCategory| . ODECAT)
(|PartialDifferentialEquationsSolverCategory| . PDECAT)
(|RoutinesTable| . ROUTINE)
(categories|
|\{AbelianGroup\| . ABELGRP|
|\{AbelianMonoid\| . ABELMON|
|\{AbelianMonoidRing\| . AMR|
|\{AbelianSemigroup\| . ABELSG|
|\{AdditiveValuationAttribute\| . ATADDVA|
|\{AffineSpaceCategory\| . AFSPCAT|
|\{Aggregate\| . AGG|
|\{Algebra\| . ALGEBRA|
|\{AlgebraicallyClosedField\| . ACF|
|\{AlgebraicallyClosedFunctionSpace\| . ACFS|
|\{ApproximateAttribute\| . ATAPPRO|
|\{ArbitraryExponentAttribute\| . ATARBEX|
|\{ArbitraryPrecisionAttribute\| . ATARBPR|
|\{ArcHyperbolicFunctionCategory\| . AHYP|
|\{ArcTrigonometricFunctionCategory\| . ATRIG|
(\text{AssociationListAggregate} . \text{ALAGG})
(\text{AttributeRegistry} . \text{ATTREG})
(\text{BagAggregate} . \text{BGAGG})
(\text{BasicType} . \text{BASTYPE})
(\text{BiModule} . \text{BMODULE})
(\text{BinaryRecursiveAggregate} . \text{BRAGG})
(\text{BinaryTreeCategory} . \text{BTCAT})
(\text{BitAggregate} . \text{BTAGG})
(\text{BlowUpMethodCategory} . \text{BLMETCT})
(\text{CachableSet} . \text{CACHSET})
(\text{CancellationAbelianMonoid} . \text{CABMON})
(\text{CanonicalAttribute} . \text{ATCANON})
(\text{CanonicalClosedAttribute} . \text{ATCANCL})
(\text{CanonicalUnitNormalAttribute} . \text{ATCUNOR})
(\text{CentralAttribute} . \text{ATCENRL})
(\text{CharacteristicNonZero} . \text{CHARNZ})
(\text{CharacteristicZero} . \text{CHARZ})
(\text{CoercibleTo} . \text{KOERCE})
(\text{Collection} . \text{CLAGG})
(\text{CombinatorialFunctionCategory} . \text{CFCAT})
(\text{Combinatorial1OpsCategory} . \text{COMBOPC})
(\text{CommutativeRing} . \text{COMRING})
(\text{CommutativeStarAttribute} . \text{ATCS})
(\text{Comparable} . \text{COMPAR})
(\text{ComplexCategory} . \text{COMPCAT})
(\text{ConvertibleTo} . \text{KONVERT})
(\text{DequeueAggregate} . \text{DQAGG})
(\text{DesingTreeCategory} . \text{DSTRCAT})
(\text{Dictionary} . \text{DIAGG})
(\text{DictionaryOperations} . \text{DIOPS})
(\text{DifferentialExtension} . \text{DIFEXT})
(\text{DifferentialPolynomialCategory} . \text{DPOLCAT})
(\text{DifferentialRing} . \text{DIFRING})
(\text{DifferentialVariableCategory} . \text{DVARCAT})
(\text{DirectProductCategory} . \text{DIRPCAT})
(\text{DivisionRing} . \text{DIVRING})
(\text{DivisorCategory} . \text{DIVCAT})
(\text{DoublyLinkedAggregate} . \text{DLAGG})
(\text{ElementaryFunctionCategory} . \text{ELEMFUN})
(\text{Eltable} . \text{ELTAB})
(\text{EltableAggregate} . \text{ELTAGG})
(\text{EntireRing} . \text{ENTIRER})
(\text{EuclideanDomain} . \text{EUCCDOM})
(\text{Evalable} . \text{EVALAB})
(\text{ExpressionSpace} . \text{ES})
(\text{ExtensibleLinearAggregate} . \text{ELAGG})
(\text{ExtensionField} . \text{XF})
(\text{Field} . \text{FIELD})
(\text{FieldOfPrimeCharacteristic} . \text{FPC})
(\text{Finite} . \text{FINITE})
45.18. VARIABLES USED

(|PolynomialSetCategory| . PSETCAT)
(|PowerSeriesCategory| . PSCAT)
(|PrimitiveFunctionCategory| . PRIMCAT)
(|PrincipalIdealDomain| . PID)
(|PriorityQueueAggregate| . PRQAGG)
(|ProjectiveSpaceCategory| . PRSPCAT)
(|PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory| . PACEXTC)
(|PseudoAlgebraicClosureOfFiniteField| . PACOFF)
(|PseudoAlgebraicClosureOfFiniteFieldCategory| . PACFFC)
(|PseudoAlgebraicClosureOfPerfectFieldCategory| . PACPERC)
(|PseudoAlgebraicClosureOfRationalNumber| . PACRAT)
(|PseudoAlgebraicClosureOfRationalNumberCategory| . PACRATC)
(|QuaternionCategory| . QUATCAT)
(|QueueAggregate| . QUAGG)
(|QuotientFieldCategory| . QFCAT)
(|RadicalCategory| . RADCAT)
(|RealClosedField| . RCFIELD)
(|RealConstant| . REAL)
(|RealNumberSystem| . RNS)
(|RealRootCharacterizationCategory| . RRCC)
(|RectangularMatrixCategory| . RMATCAT)
(|RecursiveAggregate| . RCAGG)
(|RecursivePolynomialCategory| . RPOLCAT)
(|RegularChain| . RGCHAIN)
(|RegularTriangularSetCategory| . RSETCAT)
(|RetractableTo| . RETRACT)
(|RightModule| . RMODULE)
(|Ring| . RING)
(|Rng| . RNG)
(|SegmentCategory| . SEGCAT)
(|SegmentExpansionCategory| . SEGXCAT)
(|SemiGroup| . SGROUP)
(|SetAggregate| . SETAGG)
(|SetCategory| . SETCAT)
(|SetCategoryWithDegree| . SETCATD)
(|SExpressionCategory| . SEXCAT)
(|ShallowlyMutableAttribute| . ATSHMUT)
(|SpecialFunctionCategory| . SPFCAT)
(|SquareFreeNormalizedTriangularSetCategory| . SNTSCAT)
(|SquareFreeRegularTriangularSetCategory| . SFRTCAT)
(|SquareMatrixCategory| . SMATCAT)
(|StackAggregate| . SKAGG)
(|StepThrough| . STEP)
(|StreamAggregate| . STAGG)
(|StringAggregate| . SRAGG)
(|StringCategory| . STRICAT)
(|StructuralConstantsPackage| . SCPKG)
(|TableAggregate| . TBAGG)
(|ThreeSpaceCategory| . SPACEC)
(|TranscendentalFunctionCategory| . TRANFUN)
45.18. VARIABLES USED

(\textbf{ComplexPatternMatch} . CPMATCH)
(\textbf{ComplexRootFindingPackage} . CRFP)
(\textbf{ConstantLODE} . ODECONST)
(\textbf{CyclicStreamTools} . CSTTOOLS)
(\textbf{CyclotomicPolynomialPackage} . CYCLOM)
(\textbf{DefiniteIntegrationTools} . DFINTTLS)
(\textbf{DegreeReductionPackage} . DEGRED)
(\textbf{DeRhamComplex} . DERM)
(\textbf{DifferentialSparseMultivariatePolynomial} . DSMP)
(\textbf{DirectProduct} . DIRPROD)
(\textbf{DirectProductMatrixModule} . DPMM)
(\textbf{DirectProductModule} . DPMO)
(\textbf{DiscreteLogarithmPackage} . DLP)
(\textbf{DistributedMultivariatePolynomial} . DMP)
(\textbf{DoubleResultantPackage} . DBLRESP)
(\textbf{DrawOptionFunctions0} . DROPT0)
(\textbf{DrawOptionFunctions1} . DROPT1)
(\textbf{ElementaryFunction} . EF)
(\textbf{ElementaryFunctionsUnivariateLaurentSeries} . EFULS)
(\textbf{ElementaryFunctionsUnivariatePuiseuxSeries} . EFUPXS)
(\textbf{ElementaryIntegration} . INTEF)
(\textbf{ElementaryRischDE} . RDEEF)
(\textbf{ElementaryRischDESystem} . RDEEFS)
(\textbf{EllipticFunctionsUnivariateTaylorSeries} . ELFUTS)
(\textbf{EqTable} . EQTBL)
(\textbf{EuclideanModularRing} . EMR)
(\textbf{EvaluateCycleIndicators} . EVALCYC)
(\textbf{ExponentialExpansion} . EXPEXPAN)
(\textbf{ExponentialOfUnivariatePuiseuxSeries} . EXPUPXS)
(\textbf{ExpressionSpaceFunctions1} . ES1)
(\textbf{ExpressionTubePlot} . EXPRTUBE)
(\textbf{ExtAlgBasis} . EAB)
(\textbf{FactoredFunctions} . FACTFUNC)
(\textbf{FactoredFunctionUtilities} . FRUTIL)
(\textbf{FactoringUtilities} . FACUTIL)
(\textbf{FGLMLifCanPackage} . FGLMLC)
(\textbf{FindOrderFinite} . FORDER)
(\textbf{FiniteDivisor} . FDIV)
(\textbf{FiniteFieldCyclicGroupExtension} . FFGTX)
(\textbf{FiniteFieldCyclicGroupExtensionByPolynomial} . FFGPY)
(\textbf{FiniteFieldExtension} . FFX)
(\textbf{FiniteFieldExtensionByPolynomial} . FFP)
(\textbf{FiniteFieldFunctions} . FFF)
(\textbf{FiniteFieldNormalBasisExtension} . FFBNX)
(\textbf{FiniteFieldNormalBasisExtensionByPolynomial} . FFNPY)
(\textbf{FiniteFieldPolynomialPackage} . FFPOLY)
(\textbf{FiniteFieldSolveLinearPolynomialEquation} . FFSLPE)
(\textbf{FormalFraction} . FORMAL)
(\textbf{FourierComponent} . FCOMP)
(\textbf{FractionalIdeal} . FRIDEAL)
CHAPTER 45. \texttt{SET HELP PAGE COMMAND}

\texttt{(|FramedModule| . FRMOD)}
\texttt{(|FreeAbelianGroup| . FAGROUP)}
\texttt{(|FreeAbelianMonoid| . FAMONOID)}
\texttt{(|FreeGroup| . FGROUP)}
\texttt{(|FreeModule| . FM)}
\texttt{(|FreeModule1| . FM1)}
\texttt{(|FreeMonoid| . FMONOID)}
\texttt{(|FunctionalSpecialFunction| . FSPECF)}
\texttt{(|FunctionCalled| . FUNCTION)}
\texttt{(|FunctionFieldIntegralBasis| . FFINTBAS)}
\texttt{(|FunctionSpaceReduce| . FSRED)}
\texttt{(|FunctionSpaceToUnivariatePowerSeries| . FS2UPS)}
\texttt{(|FunctionSpaceToExponentialExpansion| . FS2EXPXP)}
\texttt{(|GaloisGroupFactorizationUtilities| . GALFACTU)}
\texttt{(|GaloisGroupFactorizer| . GALFACT)}
\texttt{(|GaloisGroupPolynomialUtilities| . GALPOLYU)}
\texttt{(|GaloisGroupUtilities| . GALUTIL)}
\texttt{(|GeneralHenselPackage| . GHENSEL)}
\texttt{(|GeneralDistributedMultivariatePolynomial| . GDMP)}
\texttt{(|GeneralPolynomialGcdPackage| . GENPGCD)}
\texttt{(|GeneralSparseTable| . GSTBL)}
\texttt{(|GenericNonAssociativeAlgebra| . GNAALG)}
\texttt{(|GenExEuclid| . GENEZZ)}
\texttt{(|GeneralizedMultivariateFactorize| . GENMFAC)}
\texttt{(|GeneralModulePolynomial| . GMODPOL)}
\texttt{(|GeneralPolynomialSet| . GPOLSET)}
\texttt{(|GeneralTriangularSet| . GTSET)}
\texttt{(|GenUFactorize| . GENUFACT)}
\texttt{(|GenusZeroIntegration| . INTGO)}
\texttt{(|GosperSummationMethod| . GOSPER)}
\texttt{(|GraphImage| . GRIMAGE)}
\texttt{(|GrayCode| . GRAY)}
\texttt{(|GroebnerInternalPackage| . GBIINTERN)}
\texttt{(|GroebnerSolve| . GROEBSQL)}
\texttt{(|GuessOptionFunctions0| . GOPTO)}
\texttt{(|HashTable| . HASHTBL)}
\texttt{(|Heap| . HEAP)}
\texttt{(|HeuGcd| . HEUGCD)}
\texttt{(|HomogeneousDistributedMultivariatePolynomial| . HDMP)}
\texttt{(|HyperellipticFiniteDivisor| . HELLFDIV)}
\texttt{(|IncrementingMaps| . INCMAPS)}
\texttt{(|IndexedBits| . IBITS)}
\texttt{(|IndexedDirectProductAbelianGroup| . IDPAG)}
\texttt{(|IndexedDirectProductAbelianMonoid| . IDPAM)}
\texttt{(|IndexedDirectProductObject| . IDPO)}
\texttt{(|IndexedDirectProductOrderedAbelianMonoid| . IDPOAM)}
\texttt{(|IndexedDirectProductOrderedAbelianMonoidSup| . IDPOAMS)}
\texttt{(|IndexedExponents| . INDE)}
\texttt{(|IndexedFlexibleArray| . IFARRAY)}
45.18. VARIABLES USED

([IndexedList | . ILIST])
([IndexedMatrix | . IMATRIX])
([IndexedOneDimensionalArray | . IARRAY1])
([IndexedString | . ISTRING])
([IndexedTwoDimensionalArray | . IARRAY2])
([IndexedVector | . IVECTOR])
([InnerAlgebraicNumber | . IAN])
([InnerCommonDenominator | . ICDEN])
([InnerFiniteField | . IFF])
([InnerFreeAbelianMonoid | . IFAMON])
([InnerIndexedTwoDimensionalArray | . IIARRAY2])
([InnerMatrixLinearAlgebraFunctions | . IMATLIN])
([InnerMatrixQuotientFieldFunctions | . IMATQF])
([InnerModularGcd | . INMODGCD])
([InnerMultFact | . INNMFACT])
([InnerNormalBasisFieldFunctions | . INBFF])
([InnerNumericEigenPackage | . INEP])
([InnerNumericFloatSolvePackage | . INFSP])
([InnerPAdicInteger | . IPADIC])
([InnerPolySign | . INPSIGN])
([InnerPolySum | . ISUMP])
([InnerPrimeField | . IPF])
([InnerSparseUnivariatePowerSeries | . ISUPS])
([InnerTable | . INTABL])
([InnerTaylorSeries | . ITAYLOR])
([InnerTrigonometricManipulations | . ITRIGMNP])
([InputForm | . INFORM])
([InputFormFunctions1 | . INFORM1])
([IntegerBits | . INTBIT])
([IntegerFactorizationPackage | . INTFACT])
([IntegerMod | . ZMOD])
([IntegerSolveLinearPolynomialEquation | . INTSLPE])
([IntegralBasisPolynomialTools | . IBPTOOLS])
([IntegralBasisTools | . IBATOOL])
([IntegrationResult | . IR])
([IntegrationTools | . INTOOLS])
([InternalPrintPackage | . IPRNTPK])
([InternalRationalUnivariateRepresentationPackage | . IRURPK])
([IrredPolyOverFiniteField | . IRREDFFX])
([Kernel | . KERNEL])
([Kovacic | . KOVACIC])
([LaurentPolynomial | . LAUPOL])
([LeadingCoefficientDetermination | . LEADCDET])
([LexTriangularPackage | . LEXTRIPK])
([LieExponentials | . LEXP])
([LiePolynomial | . LPOLY])
([LinearDependence | . LINDEP])
([LinearOrdinaryDifferentialOperatorFactorizer | . LODOF])
([LinearOrdinaryDifferentialOperator1 | . LODO1])
CHAPTER 45. \textsc{set help page command}

\begin{verbatim}
([LinearOrdinaryDifferentialOperator2] . LODO2)
([LinearOrdinaryDifferentialOperatorsOps] . LODDOPS)
([LinearPolynomialEquationByFractions] . LPEFRAC)
([LinGroebnerPackage] . LGRDBP)
([LiouvillianFunction] . LF)
([ListMonoidOps] . LMOOPS)
([ListMultiDictionary] . LMDICT)
([LocalAlgebra] . LA)
([Localize] . LO)
([LyndonWord] . LWORD)
([Magma] . MAGMA)
([MakeBinaryCompiledFunction] . MKBCFUNC)
([MakeCachableSet] . MKCHSET)
([MakeUnaryCompiledFunction] . MKUCFUNC)
([MappingPackageInternalHacks1] . MAPHACK1)
([MappingPackageInternalHacks2] . MAPHACK2)
([MappingPackageInternalHacks3] . MAPHACK3)
([MeshCreationRoutinesForThreeDimensions] . MESH)
([ModMonic] . MODMON)
([ModularField] . MODFIELD)
([ModularHermitianRowReduction] . MHROWRED)
([ModularRing] . MODRING)
([ModuleMonomial] . MODMONOM)
([MoebiusTransform] . MOEBIUS)
([MonoidRing] . MRING)
([MonomialExtensionTools] . MONOTOOL)
([MPolyCatPolyFactorizer] . MPCPF)
([MPolyCatFunctions3] . MPC3)
([MRationalFactorize] . MRATFAC)
([MultipleMap] . MMAP)
([MultivariateLifting] . MLIFT)
([MultivariateSquareFree] . MULTSQFR)
([HomogeneousDirectProduct] . HDP)
([NewSparseMultivariatePolynomial] . NSMP)
([NewSparseUnivariatePolynomial] . NSUP)
([NewSparseUnivariatePolynomialFunctions2] . NSUP2)
([NonCommutativeOperatorDivision] . NCO DIV)
([NewtonInterpolation] . NEWTON)
([None] . NONE)
([NonLinearFirstOrderODESolver] . NODE1)
([NonLinearSolvePackage] . NLINSOL)
([NormRetractPackage] . NORMRETR)
([NPCoeff] . NPCOEFF)
([NumberFormats] . NUMFMT)
([NumberFieldIntegralBasis] . NFINTBAS)
([NumericTubePlot] . NUMTUBE)
([ODEIntegration] . ODEINT)
([ODETools] . ODETOOLS)
([Operator] . OP)
([OppositeMonogenicLinearOperator] . OMLO)
\end{verbatim}
(|OrderedDirectProduct| . ODP)
(|OrderedFreeMonoid| . OFMONOID)
(|OrderedVariableList| . OVAR)
(|OrderingFunctions| . ORDFUNS)
(|OrdinaryDifferentialPolynomial| . ODPOL)
(|OrdinaryDifferentialVariable| . ODVAR)
(|OrdinaryWeightedPolynomials| . OWP)
(|OutputForm| . OUTFORM)
(|PadeApproximants| . PADE)
(|PadicInteger| . PADIC)
(|PadicRational| . PADICRAT)
(|PadicRationalConstructor| . PADICRC)
(|PadicWildFunctionFieldIntegralBasis| . PWFFINTB)
(|ParadoxicalCombinatorsForStreams| . YSTREAM)
(|ParametricLinearEquations| . PLEQN)
(|PartialFractionPackage| . PFRPAC)
(|Partition| . PRTITION)
(|Pattern| . PATTERN)
(|PatternFunctions| . PATTERN1)
(|PatternMatchFunctionSpace| . PMFS)
(|PatternMatchIntegerNumberSystem| . PMINS)
(|PatternMatchIntegration| . INTPM)
(|PatternMatchKernel| . PMKERNEL)
(|PatternMatchListAggregate| . PMLSAGG)
(|PatternMatchListResult| . PATLRES)
(|PatternMatchPolynomialCategory| . PMPLCAT)
(|PatternMatchPushDown| . PMDOWN)
(|PatternMatchQuotientFieldCategory| . PMQFCAT)
(|PatternMatchResult| . PATRES)
(|PatternMatchSymbol| . PMSYM)
(|PatternMatchTools| . PMTOOLS)
(|PlaneAlgebraicCurvePlot| . ACPLOT)
(|Plot| . PLOT)
(|PlotFunctions| . PLOT1)
(|PlotTools| . PLOTTOOL)
(|Plot3D| . PLOT3D)
(|PoincareBirkhoffWittLyndonBasis| . PBWLB)
(|Point| . POINT)
(|PointsOfFiniteOrder| . PFO)
(|PointsOfFiniteOrderRational| . PFOQ)
(|PointsOfFiniteOrderTools| . PFUTOOLS)
(|PointPackage| . PT PACK)
(|Polynomial| . POLTOPOL)
(|PolynomialCategoryLifting| . POLYLIFT)
(|PolynomialCategoryQuotientFunctions| . POLYCATQ)
(|PolynomialFactorizationByRecursion| . PFBR)
(|PolynomialFactorizationByRecursionUnivariate| . PFBRU)
(|PolynomialGcdPackage| . PGCD)
(|PolynomialInterpolation| . PINTERP)
(|PolynomialInterpolationAlgorithms| . PINTERPA)
45.18. VARIABLES USED

(SparseMultivariateTaylorSeries . SMTS)
(SparseTable . STBL)
(SparseUnivariatePolynomial . SUP)
(SparseUnivariateSkewPolynomial . ORESUP)
(SparseUnivariateLaurentSeries . SULS)
(SparseUnivariatePuiseuxSeries . SUPXS)
(SparseUnivariateTaylorSeries . SUTS)
(SplitHomogeneousDirectProduct . SHDP)
(SplittingNode . SPLNODE)
(SplittingTree . SPLTREE)
(SquareMatrix . SQMATRIX)
(Stack . STACK)
(StorageEfficientMatrixOperations . MATSTOR)
(StreamInfiniteProduct . STINPROD)
(StreamTaylorSeriesOperations . STTAYLOR)
(StreamTranscendentalFunctions . STTF)
(StreamTranscendentalFunctionsNonCommutative . STTFNC)
(StringTable . STRTBL)
(SubResultantPackage . SUBRESP)
(SubSpace . SUBSPACE)
(SubSpaceComponentProperty . COMPPROP)
(SuchThat . SUCH)
(SupFractionFactorizer . SUPFRACF)
(SymmetricFunctions . SYMFUNC)
(SymmetricPolynomial . SYMPOLY)
(SystemODESolver . ODESYS)
(Table . TABLE)
(TableauxBumpers . TABLBUMP)
(TabulatedComputationPackage . TBCMPPK)
(TangentExpansions . TANEXP)
(ToolsForSign . TOOLSIGN)
(TranscendentalHermiteIntegration . INHERTR)
(TranscendentalIntegration . INTR)
(TranscendentalRischDE . RDETR)
(TranscendentalRischDESystem . RDETRS)
(TransSolvePackageService . SOLVESER)
(TriangularMatrixOperations . TRIMAT)
(TubePlot . TUBE)
(TubePlotTools . TUBETOOL)
(Tuple . TUPLE)
(TwoDimensionalArray . ARRAY2)
(TwoDimensionalPlotClipping . CLIP)
(TwoDimensionalViewport . VIEW2D)
(TwoFactorize . TWDFACT)
(UnivariateFactorize . UNIFACT)
(UnivariateLaurentSeries . ULS)
(UnivariateLaurentSeriesConstructor . ULSCONS)
(UnivariatePolynomialDecompositionPackage . UPDECOMP)
(UnivariatePolynomialDivisionPackage . UPDIVP)
(UnivariatePolynomialSquareFree . UPSQFREE)
CHAPTER 45. HELP PAGE COMMAND

(|UnivariatePuiseuxSeries| . UPXS)
(|UnivariatePuiseuxSeriesConstructor| . UPXCONS)
(|UnivariatePuiseuxSeriesWithExponentialSingularity| . UPXSSING)
(|UnivariateSkewPolynomial| . OREUF)
(|UnivariateSkewPolynomialCategoryOps| . OREPCTO)
(|UnivariateTaylorSeries| . UTS)
(|UnivariateTaylorSeriesODESolver| . UTSDDE)
(|UserDefinedPartialOrdering| . UDPO)
(|UTSodetools| . UTSDETL)
(|Variable| . VARIABLE)
(|ViewportPackage| . VIEW)
(|WeierstrassPreparation| . WEIER)
(|WeightedPolynomials| . WP)
(|WildFunctionFieldIntegralBasis| . WFFINTBS)
(|XDistributedPolynomial| . XDPOLY)
(|XExponentialPackage| . XEXPPKG)
(|XPBWPolynomial| . XPBWPOLY)
(|XPolynomial| . XPOLY)
(|XPolynomialRing| . XPR)
(|XRecursivePolynomial| . XRPOLY))

|defaults|
(|AbelianGroup| . ABELGRP-)
(|AbelianMonoid| . ABELMON-)
(|AbelianMonoidRing| . AMR-)
(|AbelianSemiGroup| . ABELESG-)
(|Aggregate| . AGG-)
(|Algebra| . ALGEBRA-)
(|AlgebraicallyClosedField| . ACF-)
(|AlgebraicallyClosedFunctionSpace| . ACFS-)
(|ArcTrigonometricFunctionCategory| . ATRIG-)
(|BagAggregate| . BGAGG-)
(|BasicType| . BASTYPE-)
(|BinaryRecursiveAggregate| . BRAGG-)
(|BinaryTreeCategory| . BTCAT-)
(|BitAggregate| . BITAGG-)
(|Collection| . CLAGG-)
(|ComplexCategory| . COMPCAT-)
(|Dictionary| . DIAGG-)
(|DictionaryOperations| . DIOPS-)
(|DifferentialExtension| . DIFEXT-)
(|DifferentialPolynomialCategory| . DPOLCAT-)
(|DifferentialRing| . DIFRING-)
(|DifferentialVariableCategory| . DVARCAT-)
(|DirectProductCategory| . DIRPCAT-)
(|DivisionRing| . DIVRING-)
(|ElementaryFunctionCategory| . ELEMFUN-)
(|EltableAggregate| . ELTAGG-)
(|EuclideanDomain| . EUCDOM-)
(|Evalable| . EVALAB-)
(|ExpressionSpace| . ES-)

694
45.18. VARIABLES USED

(|ExtensibleLinearAggregate&| . ELAGG-)
(|ExtensionField&| . XF-)
(|Field&| . FIELD-)
(|FieldOfPrimeCharacteristic&| . FPC-)
(|FiniteAbelianMonoidRing&| . FAMR-)
(|FiniteAlgebraicExtensionField&| . FAXF-)
(|FiniteDivisorCategory&| . FDIVCAT-)
(|FiniteFieldCategory&| . FFIELDC-)
(|FiniteLinearAggregate&| . FLAGG-)
(|FiniteSetAggregate&| . FSAGG-)
(|FiniteRankAggregate&| . FINRALG-)
(|FiniteRankNonAssociativeAlgebra&| . FINAALG-)
(|FloatingPointSystem&| . FPS-)
(|FramedAlgebra&| . FRAMALG-)
(|FramedNonAssociativeAlgebra&| . FRNAALG-)
(|FullyEvalableOver&| . FEVALAB-)
(|FullyLinearlyExplicitRingOver&| . FLINEXP-)
(|FullyRetractableTo&| . FRETRCT-)
(|FunctionFieldCategory&| . FFCAT-)
(|FunctionSpace&| . FS-)
(|GcdDomain&| . GCDDOM-)
(|GradedAlgebra&| . GRLG-)
(|GradedModule&| . GRMOD-)
(|Group&| . GROUP-)
(|HomogeneousAggregate&| . HOAGG-)
(|HyperbolicFunctionCategory&| . HYPCAT-)
(|IndexedAggregate&| . IXAGG-)
(|InnerEvalable&| . IEVALAB-)
(|IntegerNumberSystem&| . INS-)
(|IntegralDomain&| . INTDOM-)
(|KeyedDictionary&| . KDAGG-)
(|LazyStreamAggregate&| . LZSTAGG-)
(|LeftAlgebra&| . LALG-)
(|LieAlgebra&| . LIECAT-)
(|LinearAggregate&| . LINAGG-)
(|ListAggregate&| . LSAGG-)
(|Logic&| . LOGIC-)
(|LinearOrdinaryDifferentialOperatorCategory&| . LOOCAT-)
(|MatrixCategory&| . MATCAT-)
(|Module&| . MODULE-)
(|Monad&| . MONAD-)
(|MonadWithUnit&| . MONADWU-)
(|Monoid&| . MONOID-)
(|MonogenicAlgebra&| . MONOGEN-)
(|NonAssociativeAlgebra&| . NAALG-)
(|NonAssociativeRing&| . NASRING-)
(|NonAssociativeRng&| . NARNG-)
(|OctonionCategory&| . OC-)
(|OneDimensionalArrayAggregate&| . A1AGG-)
(|OrderedRing&| . ORDRING-)
(\texttt{|OrderedSet|} . \texttt{ORDSET}-)
(\texttt{|PartialDifferentialRing|} . \texttt{PDRING}-)
(\texttt{|PolynomialCategory|} . \texttt{POLYCAT}-)
(\texttt{|PolynomialFactorizationExplicit|} . \texttt{PFECAT}-)
(\texttt{|PolynomialSetCategory|} . \texttt{PSETCAT}-)
(\texttt{|PowerSeriesCategory|} . \texttt{PSCAT}-)
(\texttt{|QuaternionCategory|} . \texttt{QUATCAT}-)
(\texttt{|QuotientFieldCategory|} . \texttt{QFCAT}-)
(\texttt{|RadicalCategory|} . \texttt{RADCAT}-)
(\texttt{|RealClosedField|} . \texttt{RCFIELD}-)
(\texttt{|RealNumberSystem|} . \texttt{RNS}-)
(\texttt{|RealRootCharacterizationCategory|} . \texttt{RRCC}-)
(\texttt{|RectangularMatrixCategory|} . \texttt{RMATCAT}-)
(\texttt{|RecursiveAggregate|} . \texttt{RCAGG}-)
(\texttt{|RecursivePolynomialCategory|} . \texttt{RPOLCAT}-)
(\texttt{|RegularTriangularSetCategory|} . \texttt{RSETCAT}-)
(\texttt{|RetractableTo|} . \texttt{RETRACT}-)
(\texttt{|Ring|} . \texttt{RING}-)
(\texttt{|SemiGroup|} . \texttt{SGROUP}-)
(\texttt{|SetAggregate|} . \texttt{SETAGG}-)
(\texttt{|SetCategory|} . \texttt{SETCAT}-)
(\texttt{|SquareMatrixCategory|} . \texttt{SMATCAT}-)
(\texttt{|StreamAggregate|} . \texttt{STAGG}-)
(\texttt{|StringAggregate|} . \texttt{SRAGG}-)
(\texttt{|TableAggregate|} . \texttt{TBAGG}-)
(\texttt{|TranscendentalFunctionCategory|} . \texttt{TRANFUN}-)
(\texttt{|TriangularSetCategory|} . \texttt{TSETCAT}-)
(\texttt{|TrigonometricFunctionCategory|} . \texttt{TRIGCAT}-)
(\texttt{|TwoDimensionalArrayCategory|} . \texttt{ARR2CAT}-)
(\texttt{|UnaryRecursiveAggregate|} . \texttt{URAGG}-)
(\texttt{|UniqueFactorizationDomain|} . \texttt{UFD}-)
(\texttt{|UnivariateLaurentSeriesConstructorCategory|} . \texttt{ULSCCAT}-)
(\texttt{|UnivariatePolynomialCategory|} . \texttt{UPOLYCAT}-)
(\texttt{|UnivariatePowerSeriesCategory|} . \texttt{UPSCAT}-)
(\texttt{|UnivariatePuiseuxSeriesConstructorCategory|} . \texttt{UPXSCCAT}-)
(\texttt{|UnivariateTaylorSeriesCategory|} . \texttt{UTSCAT}-)
(\texttt{|VectorCategory|} . \texttt{VECTCAT}-)
(\texttt{|VectorSpace|} . \texttt{VSPACE}-))}

defvar $\texttt{localExposureDataDefault}$

— initvars —

(defvar $\texttt{localExposureDataDefault}$)
45.19. FUNCTIONS

(VECTOR
 ;;These groups will be exposed
 (LIST '|BASIC| 'CATEGORIES| 'NAGLINK| 'ANNA|)
 ;;These constructors will be explicitly exposed
 (LIST )
 ;;These constructors will be explicitly hidden
 (LIST )))

---

defvar $localExposureData

--- initvars ---

(defvar $localExposureData (COPY-SEQ $localExposureDataDefault))

---

45.19 Functions

The top level set expose command handler

[displayExposedGroups p705]
[sayMSG p331]
[displayExposedConstructors p705]
[displayHiddenConstructors p705]
[sayKeyedMsg p329]
[namestring p1040]
[pathname p1042]
[qcar p??]
[qcdr p??]
[selectOptionLC p479]
[setExposeAdd p698]
[setExposeDrop p701]
[setExpose p697]

--- defun setExpose ---

(defun |setExpose| |arg|
 "The top level set expose command handler"
 (let (fnargs fn)
   (cond
((eq arg '|%initialize%|))
((eq arg '|%display%|) "..."
(or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
(displayExposedGroups))
(sayMSG "")
(displayExposedConstructors)
(sayMSG "")
(displayHiddenConstructors)
(sayMSG "")
(and (consp arg)
(progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
(setq fn (selectOptionLC fn '|add| |drop|) nil)))
(defun setExposeAdd (arg)
"The top level set expose add command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The add Option| $linelength (specialChar '|hbar|))
(defun setExposeAddGroup (arg)
"The top level set expose add group command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The add group| $linelength (specialChar '|hbar|)))
(defun setExposeAddConstr (arg)
"The top level set expose add constructor command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The add constructor| $linelength (specialChar '|hbar|))
(defun setExposeDrop (arg)
"The top level set expose drop command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The drop| $linelength (specialChar '|lbar|)))
(defun setExposeDropGroup (arg)
"The top level set expose drop group command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The drop group| $linelength (specialChar '|hbar|))
(defun setExposeDropConstr (arg)
"The top level set expose drop constructor command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The drop constructor| $linelength (specialChar '|hbar|))
(defun setExpose (arg)
"The top level set expose command handler"
(let (fnargs fn)
(cond
(null arg)
(centerAndHighlight '|The expose| $linelength (specialChar '|hbar|))))
45.19. FUNCTIONS

|displayExposedGroups|
|sayMSG| ""
|displayExposedConstructors|
|sayMSG| ""
|sayKeyedMsg| 's2iz0049e nil))
(and (consp arg)
  (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
  (setq fn (|selectOptionLC| fn '(|group| |constructor|) nil)))
(cond
  ((eq fn '|group|) (|setExposeAddGroup| fnargs))
  ((eq fn '|constructor|) (|setExposeAddConstr| fnargs))
  (t nil)))
(t (|setExposeAdd| nil))))

---

Exposure a group

Note that $localExposureData$ is a vector of lists. It consists of [exposed groups, exposed constructors, hidden constructors] [object2String p??]

[|qcar p??|]
[|setq elt p??|]
[displayExposedGroups p705]
[sayMSG p331]
[displayExposedConstructors p705]
[displayHiddenConstructors p705]
[clearClams p??]
[getalist p??]
[sayKeyedMsg p329]
[member p1048]
[msort p??]
[centerAndHighlight p??]
[specialChar p980]
[namestring p1040]
[pathname p1042]
[sayAsManyPerLineAsPossible p??]
[$globalExposureGroupAlist p670]
[$localExposureData p697]
[$interpreterFrameName p??]
[$linelength p774]

—— defun setExposeAddGroup ——

(defun |setExposeAddGroup| (arg)
  "Exposure a group"
  (declare (special $globalExposureGroupAlist| $localExposureData|)}
The top level set expose add constructor handler

| unabbrev p?? |
| qcar p?? |
| getdatabase p1010 |
| sayKeyedMsg p329 |
| member p1048 |
| setelt p?? |
| delete p?? |
| msort p?? |
| clearClams p?? |
FUNCTIONS

45.19. FUNCTIONS

... [centerAndHighlight p??] [specialChar p980] [displayExposedConstructors p705] [$linelength p774] [$localExposureData p697] [$interpreterFrameName p??]

--- defun setExposeAddConstr ---

(defun |setExposeAddConstr| (arg)
"The top level set expose add constructor handler"
(declare (special $linelength $localExposureData $interpreterFrameName))
(if (null arg)
  (progn
    (|centerAndHighlight| 'The constructor Option| $linelength (|specialChar| '|hbar|))
    (displayExposedConstructors))
  (dolist (x arg)
    (setq x (|unabbrev| x))
    (when (consp x) (setq x (qcar x)))
    (cond
      ((null (getdatabase x 'constructorkind))
       (|sayKeyedMsg| 's2iz0049j (list x)))
      ((|member| x (elt $localExposureData 1))
       (|sayKeyedMsg| 's2iz0049k (list x $interpreterFrameName)))
      (t
       (when (|member| x (elt $localExposureData 2))
        (setelt $localExposureData 2
          (|delete| x (elt $localExposureData 2)))))
    (setelt $localExposureData 1
      (msort (cons x (elt $localExposureData 1))))
    (|clearClams|)
    (|sayKeyedMsg| 's2iz0049p (list x $interpreterFrameName)))))

---

The top level set expose drop handler

... [centerAndHighlight p??] [specialChar p980] [displayHiddenConstructors p705] [sayMSG p331] [sayKeyedMsg p329] [qcar p??] [qcdr p??] [selectOptionLC p479]
The top level set expose drop handler

(defun setExposeDrop (arg)
 "The top level set expose drop handler"
 (declare (special $linelength))
 (let ((fnargs fn)

 (cond
   (null arg)
   ((null arg)
     (centerAndHighlight
      'The drop Option! $linelength (specialChar 'hbar)))
     (displayHiddenConstructors)
     (sayMSG "")
     (sayKeyedMsg 's2iz0049f nil))
 (and (consp arg)
   (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
   (setq fn (selectOptionLC fn '(group constructor) nil)))
   (cond
    ((eq fn 'group) (setExposeDropGroup fnargs))
    ((eq fn 'constructor) (setExposeDropConstr fnargs))
    (t nil)))
   (t (setExposeDrop nil))))

---

The top level set expose drop group handler
— defun setExposeDropGroup —

(defun setExposeDropGroup (arg)
"The top level set expose drop group handler"
(defvar $linelength $localExposureData $interpreterFrameName
$globalExposureGroupAlist)

(if (null arg)
  (progn
    (centerAndHighlight
      'The group Option $linelength (specialChar 'hbar))
    (sayKeyedMsg 'siz0049l nil)
    (sayMSG "")
    (displayExposedGroups))
  (dolist (x arg)
    (when (consp x) (setq x (qcar x)))
    (cond
      ((eq x 'all)
       (setelt $localExposureData 0 nil)
       (setelt $localExposureData 1 nil)
       (setelt $localExposureData 2 nil)
       (displayExposedGroups)
       (sayMSG "")
       (displayExposedConstructors)
       (sayMSG "")
       (displayHiddenConstructors)
       (clearClams))
      ((member x (elt $localExposureData 0))
       (setelt $localExposureData 0
         (delete x (elt $localExposureData 0)))
       (clearClams))
      ((getalist $globalExposureGroupAlist x)
       (sayKeyedMsg 'siz0049i (list x $interpreterFrameName)))
      (t (sayKeyedMsg 'siz0049h (list x)))
    )))

The top level set expose drop constructor handler

[unabbrev p??]
[qcar p??]
[getdatabase p1010]
[sayKeyedMsg p329]
[member p1048]
[setelt p?]
[delete p?]
[msort p?]
[clearClams p?]
[centerAndHighlight p?]
[specialChar p980]
[sayMSG p331]
[displayExposedConstructors p705]
[displayHiddenConstructors p705]
[$linelength p774]
[$localExposureData p697]
[$interpreterFrameName p?]

— defun setExposeDropConstr —

(defun |setExposeDropConstr| (arg)
"The top level set expose drop constructor handler"
(declare (special $linelength |$localExposureData| |$interpreterFrameName|))
(if (null arg)
(progn
 (|centerAndHighlight|)
 'The constructor Option| $linelength (|specialChar| '|hbar|))
 (|sayKeyedMsg| 's2iz0049n nil)
 (|sayMSG| "")
 (|displayExposedConstructors|)
 (|sayMSG| "")
 (|displayHiddenConstructors|))
(dolist (x arg)
 (setq x (|unabbrev| x))
 (when (consp x) (setq x (qcar x)))
 (cond
 (null (getdatabase x 'constructorkind))
 (|sayKeyedMsg| 's2iz0049j (list x))
 (|member| x (elt |$localExposureData| 2))
 (|sayKeyedMsg| 's2iz0049o (list x |$interpreterFrameName|))
 (t
 (when (|member| x (elt |$localExposureData| 1))
 (setelt |$localExposureData| 1
 (|delete| x (elt |$localExposureData| 1))))
 (setelt |$localExposureData| 2
 (msort (cons x (elt |$localExposureData| 2)))))
 (|clearClams|)
 (|sayKeyedMsg| 's2iz0049q (list x |$interpreterFrameName|))))))))

—
Display exposed groups

(defun |displayExposedGroups| ()
  "Display exposed groups"
  (declare (special |$interpreterFrameName| |$localExposureData|))
  (|sayKeyedMsg| 's2iz0049a (list |$interpreterFrameName|))
  (if (null (elt |$localExposureData| 0))
      (|centerAndHighlight| "there are no exposed groups")
      (dolist (c (elt |$localExposureData| 0))
        (|centerAndHighlight| c))))

———

Display exposed constructors

(defun |displayExposedConstructors| ()
  "Display exposed constructors"
  (declare (special |$localExposureData|))
  (|sayKeyedMsg| 's2iz0049b nil)
  (if (null (elt |$localExposureData| 1))
      (|centerAndHighlight| "there are no explicitly exposed constructors")
      (dolist (c (elt |$localExposureData| 1))
        (|centerAndHighlight| c))))

———

Display hidden constructors

(defun |displayHiddenConstructors| ()
  "Display hidden constructors"
  (declare (special |$localExposureData|))
  (|sayKeyedMsg| 's2iz0049c nil)
  (if (null (elt |$localExposureData| 2))
      (|centerAndHighlight| "there are no references")
      (dolist (c (elt |$localExposureData| 2))
        (|centerAndHighlight| c))))

———

Display aliasing constructors

(defun |displayAliasingConstructors| ()
  "Display aliasing constructors"
  (declare (special |$localExposureData|))
  (|sayKeyedMsg| 's2iz0049d nil)
  (if (null (elt |$localExposureData| 3))
      (|centerAndHighlight| "there are no aliases")
      (dolist (c (elt |$localExposureData| 3))
        (|centerAndHighlight| c))))

———
CHAPTER 45. )SET HELP PAGE COMMAND

--- defun displayHiddenConstructors ---

(defun displayHiddenConstructors ()
"Display hidden constructors"
(declare (special $localExposureData))
|sayKeyedMsg| 's2iz0049c nil)
(if (null (elt $localExposureData 2))
  (|centerAndHighlight| "there are no explicitly hidden constructors")
  (dolist (c (elt $localExposureData 2))
    (|centerAndHighlight| c))))

45.20 functions

Current Values of functions Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>number of function results to cache</td>
<td>0</td>
</tr>
<tr>
<td>compile</td>
<td>compile, don’t just define function bodies off</td>
<td></td>
</tr>
<tr>
<td>recurrence</td>
<td>specially compile recurrence relations</td>
<td>on</td>
</tr>
</tbody>
</table>

--- functions ---

(|functions|
  "some interpreter function options"
  |interpreter|
  TREE
  |novar|
  {
\getchunk{functions.cache}
\getchunk{functions.compile}
\getchunk{functions.recurrence}
  })

45.21 functions cache

---------------------- The cache Option ----------------------
45.22. VARIABLES USED

Description: number of function results to cache

)set functions cache is used to tell AXIOM how many values
computed by interpreter functions should be saved. This can
save quite a bit of time in recursive functions, though one
must consider that the cached values will take up (perhaps
valuable) room in the workspace.

The value given after cache must either be the word all or
a positive integer. This may be followed by any number of
function names whose cache sizes you wish to so set. If no
functions are given, the default cache size is set.

Examples: )set fun cache all
          )set fun cache 10 f g Legendre

In general, functions will cache no returned values.

— functionscache —

(|cache|
 "number of function results to cache"
|interpreter|
FUNCTION
|setFunctionsCache|
NIL
|htSetCache|)

———

45.22 Variables Used

defvar $cacheAlist

— initvars —

(defvar |$cacheAlist| nil)

———
45.23 Functions

The top level set functions cache handler

\begin{chunk}{defun setFunctionsCache}
\begin{verbatim}
(defun |setFunctionsCache| (arg)
  "The top level set functions cache handler"
  (let (|$options| n)
    (declare (special |$options| |$cacheCount| |$cacheAlist|))
    (cond
      ((eq arg '|%initialize%|)
       (setq |$cacheCount| 0)
       (setq |$cacheAlist| nil))
      ((eq arg '|%display%|)
       (if (null |$cacheAlist|)
           (|object2String| |$cacheCount|)
           "..."))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
       (|describeSetFunctionsCache|)
       (terpri)
       (|sayAllCacheCounts|))
      (t
       (setq n (car arg))
       (cond
        ((and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
         (|sayMessage|
          
          "Your value of" ,@(|bright| n) "is invalid because ...")
         (|describeSetFunctionsCache|)
         (|terminateSystemCommand|))
        (t
         (when (cdr arg) (list (cons '|vars| (cdr arg))))
         (|countCache| n)))))
    \end{verbatim}
\end{chunk}

\defunsec{countCache}{Display a particular cache count}
\begin{verbatim}
\defunsec{countCache}{Display a particular cache count}
\end{verbatim}
\begin{chunk}{defun countCache}
(defun countCache (n)
"Display a particular cache count"
(let (tmp1 l cachecountname)
  (declare (special $options | $cacheAlist | $cacheCount))
  (cond
   (|$options|
    (cond
     ((and (consp |$options|)
       (eq (qcdr |$options|) nil)
      (progn
       (setq tmp1 (qcar |$options|))
       (and (consp tmp1)
        (eq (qcar tmp1) '|vars|)
        (progn (setq l (qcdr tmp1)) t)))
     (dolist (x l)
      (if (null (identp x))
       (|sayKeyedMsg| 's2if0007 (list x))
      (progn
       (setq |$cacheAlist| (|insertAlist| x n |$cacheAlist|))
       (setq cachecountname (internl x ';COUNT'))
       (set cachecountname n)
       (|sayCacheCount| x n))))
    (t (|optionError| (caar |$options|) nil))))
   (t
    (|sayCacheCount| nil (setq |$cacheCount| n))))
\end{chunk}

\begin{chunk}{defun insertAlist}
(defun insertAlist (a b z)
(labels (fn (a b z)
  (cond
   ((null (cdr z)) (rplac (cdr z) (list (cons a b))))
   ((equal a (elt (elt z 1) 0)) (rplac (cdr (elt z 1)) b))
   ((?order (elt (elt z 1) 0) a) (rplac (cdr z) (cons (cons a b) (cdr z)))
    (t (fn a b (cdr z))))))))
  (cond
(null z) (list (cons a b)))
((equal a (elt (elt z 0) 0)) (rplac (cdar z) b) z)
((?order (elt (elt z 0) 0) a) (cons (cons a b) z))
(t (fn a b z))))

\end{chunk}

\defunsec{describeSetFunctionsCache}{Describe the set functions cache}
\calls{describeSetFunctionsCache}{sayBrightly}
\begin{chunk}{defun describeSetFunctionsCache}
\begin{verbatim}
(defun |describeSetFunctionsCache| ()
"Describe the set functions cache"
(|sayBrightly| (list
'|%b| "set functions cache"
'|%d| "is used to tell AXIOM how many"
'|%l| "values computed by interpreter functions should be saved. This"
'|%l| "can save quite a bit of time in recursive functions, though one"
'|%l| "must consider that the cached values will take up (perhaps"
'|%l| "valuable) room in the workspace."
'|%l|
'|%l| "The value given after"
'|%b| "cache"
'|%d| "must either be the word"
'|%b| "all"
'|%d| "or a positive integer."
'|%l| "This may be followed by any number of function names whose cache"
'|%l| "sizes you wish to so set. If no functions are given, the default"
'|%l| "cache size is set."
'|l|
'|l| "Examples:"
'|l| " )set fun cache all )set fun cache 10 f g Legendre"
\end{verbatim}
\end{chunk}

\defunsec{sayAllCacheCounts}{Display all cache counts}
\calls{sayAllCacheCounts}{sayCacheCount}
\usesdollar{sayAllCacheCounts}{cacheCount}
\usesdollar{sayAllCacheCounts}{cacheAlist}
\begin{chunk}{defun sayAllCacheCounts}
\begin{verbatim}
(defun |sayAllCacheCounts| ()
"Display all cache counts"
(let (x n)
(declare (special |$cacheCount| |$cacheAlist|))
(|sayCacheCount| nil |$cacheCount|))
(when |$cacheAlist|
(do ((t0 |$cacheAlist| (cdr t0)) (t1 nil))
(or (atom t0)
 (progn (setq t1 (car t0)) nil)
 (progn
 (progn (setq x (car t1)) (setq n (cdr t1)) t1)
\begin{chunk}
\defunsec{sayCacheCount}{Describe the cache counts}
\calls{sayCacheCount}{bright}
\calls{sayCacheCount}{linearFormatName}
\calls{sayCacheCount}{sayBrightly}
\begin{chunk}{defun sayCacheCount}
(defun |sayCacheCount| (fn n)
 "Describe the cache counts"
 (let (prefix phrase)
   (setq prefix
     (cond
       (fn (cons '|function| (|bright| (|linearFormatName| fn))))
       ((eql n 0) (list '|interpreter functions |))
       (t (list '|In general, interpreter functions |)))))
   (cond
     ((eql n 0)
      (cond
       (fn
        (|sayBrightly|
         "Caching for " ,prefix "is turned off")))
       (t
        (|sayBrightly| "In general, functions will cache no returned values.")))
     (t
      (setq phrase
        (cond
          ((eq n '|all|) `(,@(|bright| '|all|) |values.|))
          ((eql n 1) (list '|only the last value.|))
          (t `(|the last| ,@(|bright| n) |values.|)))))
       (|sayBrightly|
        " " ,@prefix "will cache" ,@phrase))))
   ))
\end{chunk}

\section{functions compile}
\begin{verbatim}
--------------------- The compile Option ----------------------
Description: compile, don't just define function bodies

The compile option may be followed by any one of the following:

- \texttt{on}
- \texttt{off}
\end{verbatim}
defvar $compileDontDefineFunctions

— initvars —

(defun $compileDontDefineFunctions t
  "Compile, don’t just define function bodies")

45.24 functions recurrence

------------------ The recurrence Option ------------------

Description: specially compile recurrence relations

The recurrence option may be followed by any one of the following:

-> on
  off

The current setting is indicated.

defvar $compileRecurrence

— initvars —
(defvar $compileRecurrence t "specially compile recurrence relations")

———

— functionsrecurrence —

(|recurrence|
"specially compile recurrence relations"
|interpreter|
LITERALS
|$compileRecurrence|
(|on| |off|)
|on|)

———

45.25 fortran

Current Values of fortran Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ints2floats</td>
<td>where sensible, coerce integers to reals</td>
<td>on</td>
</tr>
<tr>
<td>fortindent</td>
<td>the number of characters indented</td>
<td>6</td>
</tr>
<tr>
<td>fortlength</td>
<td>the number of characters on a line</td>
<td>72</td>
</tr>
<tr>
<td>typedecs</td>
<td>print type and dimension lines</td>
<td>on</td>
</tr>
<tr>
<td>defaulttype</td>
<td>default generic type for FORTRAN object</td>
<td>REAL</td>
</tr>
<tr>
<td>precision</td>
<td>precision of generated FORTRAN objects</td>
<td>double</td>
</tr>
<tr>
<td>intrinsic</td>
<td>whether to use INTRINSIC FORTRAN functions</td>
<td>off</td>
</tr>
<tr>
<td>explength</td>
<td>character limit for FORTRAN expressions</td>
<td>1320</td>
</tr>
<tr>
<td>segment</td>
<td>split long FORTRAN expressions</td>
<td>on</td>
</tr>
<tr>
<td>optlevel</td>
<td>FORTRAN optimisation level</td>
<td>0</td>
</tr>
<tr>
<td>startindex</td>
<td>starting index for FORTRAN arrays</td>
<td>1</td>
</tr>
<tr>
<td>calling</td>
<td>options for external FORTRAN calls</td>
<td>...</td>
</tr>
</tbody>
</table>

Variables with current values of ... have further sub-options.
For example, issue )set calling to see what the options are for calling.
For more information, issue )help set .

—— fortran ——

(|fortran|
"view and set options for FORTRAN output"
|interpreter|
**ints2floats**

----------------- The ints2floats Option -----------------

Description: where sensible, coerce integers to reals

The ints2floats option may be followed by any one of the following:

- `on`
- `off`

The current setting is indicated.

**defvar $fortInts2Floats**

--- initvars ---

(defvar $fortInts2Floats t "where sensible, coerce integers to reals")
(|ints2floats|
"where sensible, coerce integers to reals"
|interpreter|
LITERALS
|$fortInts2Floats|
(|on| |off|)
|on|)

________

fortindent

------------------------ The fortindent Option ------------------------

Description: the number of characters indented

The fortindent option may be followed by an integer in the range
0 to inclusive. The current setting is 6

defvar $fortIndent

| initvars |

(defvar|$fortIndent| 6 "the number of characters indented")

________

| fortranfortindent |

(|fortindent|
"the number of characters indented"
|interpreter|
INTEGER
|$fortIndent|
(0 NIL)
6)

________

fortlength

------------------------- The fortlength Option -------------------------
CHAPTER 45. \texttt{)SET HELP PAGE COMMAND}

Description: the number of characters on a line

The \texttt{fortlength} option may be followed by an integer in the range 1 to inclusive. The current setting is 72

\begin{verbatim}
defvar $fortLength

   -- initvars --

(\texttt{defvar \$fortLength\ 72 \"the number of characters on a line\")

---

--- fortranfortlength ---

(\texttt{$fortlength\}
   \"the number of characters on a line\" \texttt{|interpreter|}
   \texttt{INTEGER \$fortLength}
   \texttt{(1 NIL)}
   72)

---

\texttt{typedecs}

--------------------- The typedecs Option ---------------------

Description: print type and dimension lines

The typedecs option may be followed by any one of the following:

- \texttt{on}
- \texttt{off}

The current setting is indicated.

\begin{verbatim}
defvar $printFortranDecs

   -- initvars --

(\texttt{defvar \$printFortranDecs\ 72 \"the number of characters on a line\")

---
(defvar $printFortranDecs |t| "print type and dimension lines")

——
— fortrantypedefec —

(|t|ypedecs|)
"print type and dimension lines"
|interpreter|
|LITERALS
|($printFortranDecs|)
(|[on] |off]|)
|on|

——
defaulttype

------------------- The defaulttype Option -------------------

Description: default generic type for FORTRAN object

The defaulttype option may be followed by any one of the following:

-> REAL
|INTEGER
|COMPLEX
|LOGICAL
|CHARACTER

The current setting is indicated.

defvar $defaultFortranType

—— initvars ——

(defvar |$defaultFortranType| 'real "default generic type for FORTRAN object")

——
— fortrandefaulttype ——
CHAPTER 45. )SET HELP PAGE COMMAND

((|defaulttype|
 "default generic type for FORTRAN object"
|interpreter|
LITERALS
|$defaultFortranType|
(REAL INTEGER COMPLEX LOGICAL CHARACTER)
REAL)


precision

------------------ The precision Option ------------------

Description: precision of generated FORTRAN objects

The precision option may be followed by any one of the
following:

   single
-> double

The current setting is indicated.

defvar $fortranPrecision

  — initvars —

(defvar |$fortranPrecision| '|double| "precision of generated FORTRAN objects")


  — fortranprecision —

((|precision|
 "precision of generated FORTRAN objects"
|interpreter|
LITERALS
|$fortranPrecision|
(|single| |double|)
|double|))


intrinsic

-------------- The intrinsic Option --------------

Description: whether to use INTRINSIC FORTRAN functions

The intrinsic option may be followed by any one of the following:

on
-> off

The current setting is indicated.

defvar $useIntrinsicFunctions

| initvars |
(defvar |$useIntrinsicFunctions| nil
"whether to use INTRINSIC FORTRAN functions")

--------

| fortranintrinsic |
{|intrinsic|
"whether to use INTRINSIC FORTRAN functions"
{|interpreter|
LITERALS
|$useIntrinsicFunctions|
(|on| |off|)
|off|)

--------

explength

-------------- The explength Option --------------

Description: character limit for FORTRAN expressions

The explength option may be followed by an integer in the range 0 to inclusive. The current setting is 1320
defvar $maximumFortranExpressionLength

— initvars —

(defun $maximumFortranExpressionLength 1320
  "character limit for FORTRAN expressions")

---------

— fortranexplenlength —

(defun $fortranexplenlength
  (NIL
   "character limit for FORTRAN expressions"
   (integer
    (0 NIL)
    1320))

---------

segment

--------------------- The segment Option ----------------------

Description: split long FORTRAN expressions

The segment option may be followed by any one of the following:

-> on
  off

The current setting is indicated.

defvar $fortranSegment

— initvars —

(defun $fortranSegment t "split long FORTRAN expressions")

----------
45.25. FORTRAN

--- fortransegment ---

"split long FORTRAN expressions"
LITERALS
($fortranSegment
(|on| |off|)
|on|)

-----

optlevel

------------------- The optlevel Option -------------------

Description: FORTRAN optimisation level

The optlevel option may be followed by an integer in the range 0 to 2 inclusive. The current setting is 0

defvar $fortranOptimizationLevel

--- initvars ---

(defvar |$fortranOptimizationLevel| 0 "FORTRAN optimisation level")

-----

--- fortranoptlevel ---

|optlevel|
"FORTRAN optimisation level"
|interpreter|
INTEGER
|$fortranOptimizationLevel|
(0 2)
0)

-----
The startindex Option

Description: starting index for FORTRAN arrays

The startindex option may be followed by an integer in the range 0 to 1 inclusive. The current setting is 1.

```
defvar $fortranArrayStartingIndex
    "starting index for FORTRAN arrays"
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tempfile</td>
<td>set location of temporary data files</td>
<td>/tmp/</td>
</tr>
<tr>
<td>directory</td>
<td>set location of generated FORTRAN files</td>
<td>./</td>
</tr>
<tr>
<td>linker</td>
<td>linker arguments (e.g. libraries to search)</td>
<td>-lxlf</td>
</tr>
</tbody>
</table>
The tempfile Option

Description: set location of temporary data files

)set fortran calling tempfile is used to tell AXIOM where to place intermediate FORTRAN data files. This must be the name of a valid existing directory to which you have permission to write (including the final slash).

Syntax:
 )set fortran calling tempfile DIRECTORYNAME

The current setting is /tmp/

defvar $fortranTmpDir

— initvars —

(defvar $fortranTmpDir "/tmp/" "set location of temporary data files")
CHAPTER 45. \textsc{)set Help Page Command}

\begin{verbatim}
(defun setFortTmpDir (arg)
  "The top level set fortran calling tempfile handler"
  (let (mode)
    (declare (special $fortranTmpDir))
    (cond
      ((eq arg '|%initialize%|) (setq $fortranTmpDir "$/tmp/"))
      ((eq arg '|%display%|) (if (stringp $fortranTmpDir)
                                   $fortranTmpDir
                                   (pname $fortranTmpDir)))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
       (|describeSetFortTmpDir|))
      ((null (setq mode (|validateOutputDirectory| arg)))
       (|sayBrightly| `(" Sorry, but your argument(s)" ,@(|bright| arg)
                      "is(are) not valid." |%l|))
       (|describeSetFortTmpDir|))
      (t (setq $fortranTmpDir mode))))

(defun validateOutputDirectory (arg)
  "Validate the output directory"
  (declare (special $fortranTmpDir))
  (if (stringp $fortranTmpDir)
      (error `(" Error! Directory" ,@(|bright| arg)
                "is not a directory." |%l|))
      (|validateOutputDirectory| arg))))
\end{verbatim}

The top level set fortran calling tempfile handler

Validate the output directory
(defun |validateOutputDirectory| (x)
  "Validate the output directory"
  (let ((dirname (car x)))
    (when (and (pathname-directory dirname) (null (probe-file dirname)))
      dirname)))

Describe the set fortran calling tempfile

[sayBrightly p??]

[$fortranTmpDir p723]

— defun describeSetFortTmpDir —

(defun |describeSetFortTmpDir| ()
  "Describe the set fortran calling tempfile"
  (declare (special $fortranTmpDir))
  (sayBrightly (list
    '|%b| "set fortran calling tempfile"
    '|%d| " is used to tell AXIOM where"
    '|%l| " to place intermediate FORTRAN data files. This must be the "
    '|%l| " name of a valid existing directory to which you have permission "
    '|%l| " to write (including the final slash)."
    '|%l|
    '|%l| " Syntax:"
    '|%l| " )set fortran calling tempfile DIRECTORYNAME"
    '|%l|
    '|%l| " The current setting is"
    '|%b| |$fortranTmpDir|
    '|%d|)))

directory

------------------ The directory Option ------------------

Description: set location of generated FORTRAN files

)set fortran calling directory is used to tell AXIOM where to place generated FORTRAN files. This must be the name of a valid existing directory to which you have permission to write (including the final slash).

Syntax:
\texttt{(defvar \$fortranDirectory \"")/" "set location of generated FORTRAN files")}

\texttt{(defun setFortDir \(arg\)) (declare (special \$fortranDirectory)))
(let (mode)
  (COND
    ((eq arg \'\%initialize\%) (setq \$fortranDirectory \"./")
    )
((eq arg '(%display%))
 (if (stringp $fortranDirectory$)
    $fortranDirectory$
    (pname $fortranDirectory$))
((or (null arg) (eq arg '%describe%|) (eq (car arg) '?))
 (|describeSetFortDir|))
((null (setq mode (|validateOutputDirectory| arg)))
 |sayBrightly| '(" Sorry, but your argument(s)" ,@(|bright| arg)
 "is(are) not valid." |%l|))
 (|describeSetFortDir|)
 (t (setq $fortranDirectory$ mode))))

defun describeSetFortDir
[sayBrightly p??]
[$fortranDirectory p726]

— defun describeSetFortDir —

(defun |describeSetFortDir| ()
 (declare (special $fortranDirectory$))
 (|sayBrightly| (list
   '|%b| ")set fortran calling directory"'
   '|%d| " is used to tell AXIOM where"
   '|%l| " to place generated FORTRAN files. This must be the name "
   '|%l| " of a valid existing directory to which you have permission "
   '|%l| " to write (including the final slash)."
   '|%l| (|%i| " Syntax:"'
   (|%l| " )set fortran calling directory DIRECTORYNAME"'
   (|%l| " The current setting is"
   (|%b| $fortranDirectory$
   (|%d|))

linker

---------------------- The linker Option ----------------------

Description: linker arguments (e.g. libraries to search)
)set fortran calling linkerargs is used to pass arguments to
the linker when using mkFort to create functions which call
Fortran code. For example, it might give a list of libraries
to be searched, and their locations.
The string is passed verbatim, so must be the correct syntax for
the particular linker being used.

Example: )set fortran calling linker "-lxlf"

The current setting is -lxlf

defvar $fortranLibraries

— initvars —

(defvar $fortranLibraries| "-lxlf"
"linker arguments (e.g. libraries to search)"
)

— callinglinker —

(linker
"linker arguments (e.g. libraries to search)"
(interpreter)
FUNCTION
(setLinkerArgs)
(("enter linker arguments 
 STRING
|$fortranLibraries|
|chkDirectory| 
"-lxlf")
NIL
)

defun setLinkerArgs

[object2String p??]
[describeSetLinkerArgs p729]
[$fortranLibraries p728]

— defun setLinkerArgs —
(defun |setLinkerArgs| (arg)
  (declare (special |$fortranLibraries|))
  (cond
   ((eq arg '|%initialize%|) (setq |$fortranLibraries| "-lxlf"))
   ((eq arg '|%display%|) (|object2String| |$fortranLibraries|))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeSetLinkerArgs|))
   ((and (listp arg) (stringp (car arg)))
    (setq |$fortranLibraries| (car arg))
   (t (|describeSetLinkerArgs|))))

defun describeSetLinkerArgs

[sayBrightly p??]
[$fortranLibraries p728]

— defun describeSetLinkerArgs —

(defun |describeSetLinkerArgs| ()
  (declare (special |$fortranLibraries|))
  (|sayBrightly| (list
    '|%b| "set fortran calling linkerargs"
    '|%d| " is used to pass arguments to the linker"
    '|%l| " when using "
    '|%b| "mkFort"
    '|%d| " to create functions which call Fortran code."
    '|%l| " For example, it might give a list of libraries to be searched,"
    '|%l| " and their locations."
    '|%l| " The string is passed verbatim, so must be the correct syntax for"
    '|%l| " the particular linker being used."
    '|%l| " Example: )set fortran calling linker \"-lxlf\\"
    '|%l| " The current setting is"
    '|%b| |$fortranLibraries|
    '|%d|)))

45.26  hyperdoc

Current Values of  hyperdoc Variables
CHAPTER 45. )SET HELP PAGE COMMAND

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullscreen</td>
<td>use full screen for this facility</td>
<td>off</td>
</tr>
<tr>
<td>mathwidth</td>
<td>screen width for history output</td>
<td>120</td>
</tr>
</tbody>
</table>

--- hyperdoc ---

```lisp
(hyperdoc
 "options in using HyperDoc"
 (interpreter
 TREE
 (novar
 (getchunk hyperdocfullscreen)
 (getchunk hyperdocmathwidth)
 ))

```

---

fullscreen

------------- The fullscreen Option ---------------

Description: use full screen for this facility

The fullscreen option may be followed by any one of the following:

- on
  -> off

The current setting is indicated.

defvar $fullScreenSysVars

--- initvars ---

(defvar $fullScreenSysVars nil "use full screen for this facility")

---

--- hyperdocfullscreen ---
mathwidth

--------------------- The mathwidth Option ---------------------

Description: screen width for history output

The mathwidth option may be followed by an integer in the range
0 to inclusive. The current setting is 120

defvar $historyDisplayWidth

— initvars —

(defvar $historyDisplayWidth 120 "screen width for history output")

— hyperdocmathwidth —

{|mathwidth|
 "screen width for history output"
 |interpreter|
 INTEGER
 |$historyDisplayWidth|
 (0 NIL)
 120)


---

**fullscreen**

--- The fullscreen Option ---

Description: use fullscreen facility, if possible

The fullscreen option may be followed by any one of the following:

- `on`
- `off`

The current setting is indicated.

**defvar $useFullScreenHelp**

--- initvars ---

```lisp
(defvar $useFullScreenHelp t "use fullscreen facility, if possible")
```

---

--- help ---

```
(defun helpfullscreen
  "view and set some help options"
  (interpreter
   TREE
   novar
   (getchunk helpfullscreen))
```

---

---

--- helpfullscreen ---
45.28 history

--------------------- The history Option ----------------------

Description: save workspace values in a history file

The history option may be followed by any one of the following:

-> on
   off

The current setting is indicated.

defvar $HiFiAccess

     — initvars —

( defvar $HiFiAccess t "save workspace values in a history file"
 )

     — history —

( history
   "save workspace values in a history file"
   $HiFiAccess
   ( on off )
   on )

      —
## 45.29 messages

Current Values of messages Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoload</td>
<td>print file auto-load messages</td>
<td>off</td>
</tr>
<tr>
<td>bottomup</td>
<td>display bottom up modemap selection</td>
<td>off</td>
</tr>
<tr>
<td>coercion</td>
<td>display datatype coercion messages</td>
<td>off</td>
</tr>
<tr>
<td>dropmap</td>
<td>display old map defn when replaced</td>
<td>off</td>
</tr>
<tr>
<td>expose</td>
<td>warning for unexposed functions</td>
<td>off</td>
</tr>
<tr>
<td>file</td>
<td>print msgs also to SPADMSG LISTING</td>
<td>off</td>
</tr>
<tr>
<td>frame</td>
<td>display messages about frames</td>
<td>off</td>
</tr>
<tr>
<td>highlighting</td>
<td>use highlighting in system messages</td>
<td>off</td>
</tr>
<tr>
<td>instant</td>
<td>present instantiation summary</td>
<td>off</td>
</tr>
<tr>
<td>insteach</td>
<td>present instantiation info</td>
<td>off</td>
</tr>
<tr>
<td>interponly</td>
<td>say when function code is interpreted</td>
<td>on</td>
</tr>
<tr>
<td>number</td>
<td>display message number with message</td>
<td>off</td>
</tr>
<tr>
<td>prompt</td>
<td>set type of input prompt to display</td>
<td>step</td>
</tr>
<tr>
<td>selection</td>
<td>display function selection msgs</td>
<td>off</td>
</tr>
<tr>
<td>set</td>
<td>show )set setting after assignment</td>
<td>off</td>
</tr>
<tr>
<td>startup</td>
<td>display messages on start-up</td>
<td>off</td>
</tr>
<tr>
<td>summary</td>
<td>print statistics after computation</td>
<td>off</td>
</tr>
<tr>
<td>testing</td>
<td>print system testing header</td>
<td>off</td>
</tr>
<tr>
<td>time</td>
<td>print timings after computation</td>
<td>off</td>
</tr>
<tr>
<td>type</td>
<td>print type after computation</td>
<td>on</td>
</tr>
<tr>
<td>void</td>
<td>print Void value when it occurs</td>
<td>off</td>
</tr>
<tr>
<td>any</td>
<td>print the internal type of objects of domain Any</td>
<td>on</td>
</tr>
<tr>
<td>naglink</td>
<td>show NAGLink messages</td>
<td>on</td>
</tr>
</tbody>
</table>

---

(messages)

"show messages for various system features"

(messagesany)

(messagesautoload)

(messagesbottomup)

(messagescoercion)

(messagesdropmap)

(messagesexpose)

(messagesfile)

(messagesframe)

(messageshighlighting)
any

----------------------- The any Option -----------------------

Description: print the internal type of objects of domain Any

The any option may be followed by any one of the following:

-> on
   off

The current setting is indicated.

defvar $printAnyIfTrue

— initvars —

(defvar |$printAnyIfTrue| t
   "print the internal type of objects of domain Any")

— messagesany —

(|any|
"print the internal type of objects of domain Any"
|interpreter|
LITERALS
|$\text{printAnyIfTrue}$|
(|on| |off|)
|on|)

autoload

--------------------- The autoload Option ---------------------

Description: print file auto-load messages

defvar $\text{printLoadMsgs}$

— initvars —

(defvar $\text{printLoadMsgs}$ nil "print file auto-load messages")

--------------------- The bottomup Option ---------------------

Description: display bottom up modemap selection

The bottomup option may be followed by any one of the
following:

on
-> off

The current setting is indicated.

defvar $reportBottomUpFlag

--- initvars ---

(defvar |$reportBottomUpFlag| nil "display bottom up modemap selection")

---

--- messagesbottomup ---

(|bottomup|
 "display bottom up modemap selection"
 |development|
 LITERALS
 |$reportBottomUpFlag|
 (|on| |off|)
 |off|)

---

coercion

------------------- The coercion Option -------------------

Description: display datatype coercion messages

The coercion option may be followed by any one of the following:

on
-> off

The current setting is indicated.
defvar $reportCoerceIfTrue

— initvars —

(defun $reportCoerceIfTrue nil "display datatype coercion messages")

——

— messagescoercion —

(\| coercion\| "display datatype coercion messages" \|development\| LITERALS \|$reportCoerceIfTrue\| (\|on\| \|off\|) \|off\|)

——

dropmap

--------------------- The dropmap Option ---------------------

Description: display old map defn when replaced

The dropmap option may be followed by any one of the following:

  on
  \rightarrow off

The current setting is indicated.

defvar $displayDroppedMap

— initvars —

(defun $displayDroppedMap nil "display old map defn when replaced")

——
— messagesdropmap —

(|dropmap|
 "display old map defn when replaced"
|interpreter|
LITERALS
|$displayDroppedMap|
(|on| |off|)
|off|)

———

expose

---------------------- The expose Option ----------------------

Description: warning for unexposed functions

The expose option may be followed by any one of the
following:

   on
   -> off

The current setting is indicated.

defvar $giveExposureWarning

— initvars —

(defvar |$giveExposureWarning| nil "warning for unexposed functions")

———

— messagesexpose —

(|expose|
 "warning for unexposed functions"
|interpreter|
LITERALS
|$giveExposureWarning|
(|on| |off|)

CHAPTER 45. )SET HELP PAGE COMMAND

|off|

----------------------- The file Option -----------------------

Description: print msgs also to SPADMSG LISTING

The file option may be followed by any one of the following:

- on
  -> off

The current setting is indicated.

defvar $printMsgsToFile

— initvars —

(defvar |$printMsgsToFile| nil "print msgs also to SPADMSG LISTING")

---------------------- The frame Option -----------------------

|file|

"print msgs also to SPADMSG LISTING"

LITERALS

|$printMsgsToFile|

(|on| |off|)

|off|
Description: display messages about frames

The frame option may be followed by any one of the following:

```lisp
  on
  -> off
```

The current setting is indicated.

```lisp
(defvar $frameMessages
  nil "display messages about frames")
```

highlighting

------------------- The highlighting Option -------------------

Description: use highlighting in system messages

The highlighting option may be followed by any one of the following:

```lisp
  on
  -> off
```

The current setting is indicated.
defvar $highlightAllowed

— initvars —

(defvar $highlightAllowed nil "use highlighting in system messages")

— messageshighlighting —

(|highlighting|
 "use highlighting in system messages"
 |interpreter|
 LITERALS
 |$highlightAllowed|
 (|on| |off|)
 |off|)

instant

--------------------- The instant Option ---------------------

Description: present instantiation summary

The instant option may be followed by any one of the following:

    on
    -> off

The current setting is indicated.

defvar $reportInstantiations

— initvars —

(defvar $reportInstantiations nil "present instantiation summary")

———
--- messagesinstant ---

|instant|
"present instantiation summary"
|development|
LITERALS
|$reportInstantiations|
(|on| |off|)
|off|)

-------

insteach

--------------------- The insteach Option ---------------------

Description: present instantiation info

The insteach option may be followed by any one of the following:

on

-> off

The current setting is indicated.

defvar $reportEachInstantiation—

--- initvars ---

(defvar |$reportEachInstantiation| nil "present instantiation info")

-------

--- messagesinsteach ---

|insteach|
"present instantiation info"
|development|
LITERALS
|$reportEachInstantiation|
(|on| |off|)
interponly

----------------------- The interponly Option -----------------------

Description: say when function code is interpreted

The interponly option may be followed by any one of the following:

- on
- off

The current setting is indicated.

defvar $reportInterpOnly

--- initvars ---

(defun $reportInterpOnly (t "say when function code is interpreted")

---

--- messagesinterponly ---

(interponly
 "say when function code is interpreted"
 $reportInterpOnly
 (on |off|)
 |on|)

---

naglink

----------------------- The naglink Option -----------------------
Description: show NAGLink messages

The naglink option may be followed by any one of the following:

- on
- off

The current setting is indicated.

defvar $nagMessages

— initvars —

(defvar |$nagMessages| t "show NAGLink messages")

— messagesnaglink —

{|naglink|
"show NAGLink messages"
|interpreter|
LITERALS
|$nagMessages|
(|on| |off|)
|on|)

number

---------------------- The number Option ----------------------

Description: display message number with message

The number option may be followed by any one of the following:

- on
- off
The current setting is indicated.

\textbf{defvar $displayMsgNumber}

\textbf{— initvars —}

(defvar \$displayMsgNumber nil "display message number with message")

\textbf{— messagesnumber —}

(defvar \$displayMsgNumber nil "display message number with message"
|interpreter|
LITERALS
\$displayMsgNumber
(on| off)
|off|)

\textbf{prompt}

\textbf{------------------- The prompt Option -------------------}

\textbf{Description: set type of input prompt to display}

The prompt option may be followed by any one of the following:

\begin{itemize}
  \item none
  \item frame
  \item plain
  \item step
  \item verbose
\end{itemize}

The current setting is indicated.

\textbf{defvar $inputPromptType}

\textbf{— initvars —}
(defvar $inputPromptType 'step "set type of input prompt to display"

---

--- messagesprompt ---

(prompt "set type of input prompt to display"
(interpreter LITERALS
($inputPromptType
 (\none\ |frame\ |plain\ |step\ |verbose\)
|step|)

---

selection

------------------------ The selection Option ------------------------

Description: display function selection msgs

The selection option may be followed by any one of the following:

on

→ off

The current setting is indicated.

TPDHERE: This is a duplicate of )set mes bot on because both use the $reportBottomUpFlag flag

--- messagesselection ---

(selection
 "display function selection msgs"
(interpreter LITERALS
($reportBottomUpFlag
 (\on\ |off\)
|off|)

---
set

----------------------- The set Option ------------------------

Description: show )set setting after assignment

The set option may be followed by any one of the following:

on

\rightarrow off

The current setting is indicated.

defvar $displaySetValue

— initvars —

(defvar $displaySetValue nil "show )set setting after assignment")

———

— messagesset —

(let|
"show )set setting after assignment"
|interpreter|
LITERALS
|$displaySetValue|
(|on| |off|)
|off|)

———

startup

--------------------- The startup Option ---------------------

Description: display messages on start-up

The startup option may be followed by any one of the following:

on
defvar $displayStartMsgs

— initvars —
(defvar "display messages on start-up")

summary

--------------------- The summary Option ----------------------

Description: print statistics after computation

The summary option may be followed by any one of the following:

on
-> off

The current setting is indicated.

defvar $printStatisticsSummaryIfTrue

— initvars —
(defvar $printStatisticsSummaryIfTrue| nil
  "print statistics after computation")

)——

— messageSummary —

(|summary|
 "print statistics after computation"
|interpreter|
LITERALS
|$printStatisticsSummaryIfTrue|
(|on| |off|)
|off|)

)——

testing

--------------------- The testing Option ---------------------

Description: print system testing header

The testing option may be followed by any one of the following:

    on
-> off

The current setting is indicated.

defvar $testingSystem

)—— initvars ——

(defvar $testingSystem| nil "print system testing header")

)——

— messageTesting —
45.29. MESSAGES

(time

----------------------- The time Option -----------------------

Description: print timings after computation

The time option may be followed by any one of the following:

  on
  -> off
  long

The current setting is indicated.

defvar $printTimeIfTrue

— initvars —

(defvar |$printTimeIfTrue| nil "print timings after computation")

— messagestime —

(time
  "print timings after computation"
  (interpreter)
  LITERALS
  |$printTimeIfTrue|
  ((|on| |off| |long|)
   |off|)

— — —


type

----------------------- The type Option -----------------------

Description: print type after computation

The type option may be followed by any one of the following:

- on
- off

The current setting is indicated.

```
defvar $printTypeIfTrue

| initvars |
(defvar |$printTypeIfTrue| t "print type after computation")

messagestype

(type |messagestype| "print type after computation"
(interpreter |LITERALS| |$printTypeIfTrue|
(on|off|)
(on|)

| void |

----------------------- The void Option -----------------------

Description: print Void value when it occurs

The void option may be followed by any one of the following:

- on
- off
The current setting is indicated.

\[
\text{defvar $printVoidIfTrue}
\]

---

(\text{defvar $printVoidIfTrue nil "print Void value when it occurs"})

---

\[
\text{messagesvoid}
\]

---

\[
\text{45.30 naglink}
\]

Current Values of naglink Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>internet address of host for NAGLink</td>
<td>localhost</td>
</tr>
<tr>
<td>persistence</td>
<td>number of (fortran) functions to remember</td>
<td>1</td>
</tr>
<tr>
<td>messages</td>
<td>show NAGLink messages</td>
<td>on</td>
</tr>
<tr>
<td>double</td>
<td>enforce DOUBLE PRECISION ASPs</td>
<td>on</td>
</tr>
</tbody>
</table>

---

(\text{naglink})

"options for NAGLink"

<table>
<thead>
<tr>
<th>interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE</td>
</tr>
</tbody>
</table>
host

--------------- The host Option ---------------

Description: internet address of host for NAGLink

)set naglink host is used to tell AXIOM which host to contact for a NAGLink request. An Internet address should be supplied. The host specified must be running the NAGLink daemon.

The current setting is localhost

defvar $nagHost

— initvars —

(defvar |$nagHost| "localhost" "internet address of host for NAGLink")

———

— naglinkhost —

(|host|
"internet address of host for NAGLink"
|interpreter|
FUNCTION
|setNagHost|
|("enter host name"
DIRECTORY
|$nagHost|
|chkDirectory|
"localhost")
NIL)
defun setNagHost

[object2String p??]
[describeSetNagHost p755]
[$nagHost p754]

— defun setNagHost —

(defun setNagHost (arg)
  (declare (special $nagHost))
  (cond
    ((eq arg '|%initialize%|) (setq $nagHost "localhost"))
    ((eq arg '|%display%|) (object2String $nagHost))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (describeSetNagHost))
    (t (setq $nagHost (object2String arg)))))

— defun describeSetNagHost —

(defun describeSetNagHost ()
  (declare (special $nagHost))
  (sayBrightly (list
    '|%b| "set naglink host"
    '|%d| "is used to tell AXIOM which host to contact for"
    '|%l| "a NAGLink request. An Internet address should be supplied. The host"
    '|%l| "specified must be running the NAGLink daemon."
    '|%l|
    '|%l| "The current setting is"
    '|%b| $nagHost
    '|%d|)))

persistence

------------------- The persistence Option -------------------
Description: number of (fortran) functions to remember

)set naglink persistence is used to tell the nagd daemon how many ASP source and object files to keep around in case you reuse them. This helps to avoid needless recompilations. The number specified should be a non-negative integer.

The current setting is 1

defvar $fortPersistence

— initvars —

(defvar |$fortPersistence| 1 "number of (fortran) functions to remember")

——

— naglinkpersistence —

(|persistence| "number of (fortran) functions to remember"
|interpreter| FUNCTION
|setFortPers| ("Requested remote storage (for asps):"
INTEGER
|$fortPersistence|
(0 NIL)
10))
NIL)

——

defun setFortPers

[describeFortPersistence p757]
[sayMessage p??]
[bright p??]
[terminateSystemCommand p452]
|$fortPersistence p756]

— defun setFortPers —
(defun |setFortPers| (arg)
(let (n)
 (declare (special |$fortPersistence|))
 (cond
 ((eq arg '|'initialize%|) (setq |$fortPersistence| 1))
 ((eq arg '|'display%|) |$fortPersistence|)
 ((or (null arg) (eq arg '|'describe%|) (eq (car arg) '?))
  (|describeFortPersistence|))
 (t
  (setq n (car arg))
  (cond
   ((or (null (integerp n)) (minusp n))
    (|sayMessage|
     '("Your value of" ,(bright n) "is invalid because ..."))
    (|describeFortPersistence|)
    (|terminateSystemCommand|))
   (t (setq |$fortPersistence| (car arg))))))))

——

defun describeFortPersistence

[sayBrightly p??]
[|$fortPersistence| p756]

— defun describeFortPersistence —

(defun |describeFortPersistence| ()
 (declare (special |$fortPersistence|))
 (|sayBrightly| (list
 '('|b| "set naglink persistence"
 ''|b| "is used to tell the"
 ''|b| "nagd"
 ''|d| " daemon how many ASP"
 ''|l|
 " source and object files to keep around in case you reuse them. This helps"
 ''|l| " to avoid needless recompilations. The number specified should be a "
 ''|l| " non-negative integer."
 ''|l|
 ''|l| " The current setting is"
 ''|b| |$fortPersistence|
 ''|d|)))

——
messages

---------------------- The messages Option ----------------------

Description: show NAGLink messages

The messages option may be followed by any one of the following:

-> on
   off

The current setting is indicated.

TPDHERE: this is the same as )set nag mes on

— naglinkmessages —

(defvar $nagMessages |
   (on | off)
   $nagMessages |
   "show NAGLink messages" |
   "Interpreter"
)

----------

double

---------------------- The double Option ----------------------

Description: enforce DOUBLE PRECISION ASPs

The double option may be followed by any one of the following:

-> on
   off

The current setting is indicated.

(defvar $nagEnforceDouble |
   — initvars —
)
The result of the )set output command is:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbreviate</td>
<td>abbreviate type names</td>
<td>off</td>
</tr>
<tr>
<td>algebra</td>
<td>display output in algebraic form</td>
<td>On:CONSOLE</td>
</tr>
<tr>
<td>characters</td>
<td>choose special output character set</td>
<td>plain</td>
</tr>
<tr>
<td>fortran</td>
<td>create output in FORTRAN format</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>fraction</td>
<td>how fractions are formatted</td>
<td>vertical</td>
</tr>
<tr>
<td>html</td>
<td>create output in HTML style</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>length</td>
<td>line length of output displays</td>
<td>77</td>
</tr>
<tr>
<td>mathml</td>
<td>create output in MathML style</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>openmath</td>
<td>create output in OpenMath style</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>script</td>
<td>display output in SCRIPT formula format</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>scripts</td>
<td>show subscripts,... linearly</td>
<td>off</td>
</tr>
<tr>
<td>showeditor</td>
<td>view output of )show in editor</td>
<td>off</td>
</tr>
<tr>
<td>tex</td>
<td>create output in TeX style</td>
<td>Off:CONSOLE</td>
</tr>
</tbody>
</table>

Since the output option has a bunch of sub-options each suboption is defined within the output structure.

--- output ---

(output)
"view and set some output options"
(interpreter)
TREE
(novar)
|abbreviate|-------------------- The abbreviate Option --------------------|

Description: abbreviate type names

The abbreviate option may be followed by any one of the following:

on

-> off

The current setting is indicated.

defvar $abbreviateTypes

— initvars —

(defvar $abbreviateTypes nil "abbreviate type names")

———

— outputabbreviate —

(|abbreviate|
  "abbreviate type names")
algebra

------------------------ The algebra Option ------------------------

Description: display output in algebraic form

)set output algebra is used to tell AXIOM to turn algebra-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax: )set output algebra <arg>

where arg can be one of

  on        turn algebra printing on (default state)
  off       turn algebra printing off
  console   send algebra output to screen (default state)
  fp<.fe>   send algebra output to file with file prefix fp
            and file extension .fe. If not given, .fe defaults to .spout.

If you wish to send the output to a file, you may need to issue this command twice: once with on and once with the file name.
For example, to send algebra output to the file polymer.spout, issue the two commands

  )set output algebra on
  )set output algebra polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: On:CONSOLE

defvar $algebraFormat

    — initvars —

(defvar $algebraFormat t "display output in algebraic form")
defvar $algebraOutputFile

---

initvars

(defvar |$algebraOutputFile| "CONSOLE"
"where algebra printing goes (enter \em console or a pathname)?)")

---

outputalgebra

(algebra
"display output in algebraic form"
interpreter
FUNCTION
(setOutputAlgebra
("display output in algebraic form"
LITERALS
|$algebraFormat|
(off |on|)
on)
break $algebraFormat
("where algebra printing goes (enter \em console or a pathname)"
FILENAME
|$algebraOutputFile|
chkOutputFileName
"console")
NIL)

---

defvar $algebraOutputStream

---

initvars

(defvar |$algebraOutputStream| *standard-output*)

---
defun setOutputAlgebra

(let (label tmp1 tmp2 ptype fn ft fm filename teststream)
  (cond
    ((eq arg '|%initialize%|)
      (setq |$algebraOutputStream|
        (defiostream '((mode . output) (device . console)) 255 0))
      (setq |$algebraOutputFile| "CONSOLE")
      (setq |$algebraFormat| t))
    ((eq arg '|%display%|)
      (if |$algebraFormat|
        (setq label "On:")
        (setq label "Off:"))
      (concat label |$algebraOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (describeSetOutputAlgebra))
    (t
      (cond
        ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (member fn '(y n ye yes no o on of off console
              |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|))))
'\(\text{ok}\))
(t (setq arg (list fn '\(\text{spout}\))))
(cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    (((\|member\| (upcase fn) '\(y n ye o of\))
      (\|sayKeyedMsg\| 's2iv0002 '(\|algebra\| \|algebra\|)))
     (((\|member\| (upcase fn) '\(no off\)) (setq $\|algebraFormat\| nil))
      (((\|member\| (upcase fn) '\(yes on\)) (setq $\|algebraFormat\| t))
       (eq (upcase fn) '\(\text{console}\))
       (shut $\|algebraOutputStream\))
       (setq $\|algebraOutputStream\|
            (defiostream '((mode . output) (device . console)) 255 0))
       (setq $\|algebraOutputFile\| "\text{CONSOLE}")))))
    (t
     (progn
      (setq fn (qcar arg))
      (setq tmp1 (qcdr arg))
      (and (consp tmp1)
       (eq (qcdr tmp1) nil)
       (progn (setq ft (qcar tmp1)) t))))
    (t
     (when (setq ptype (\|pathnameType\| fn))
       (setq fn (concat (\|pathnameDirectory\| fn) (\|pathnameName\| fn)))
       (setq ft ptype))
     (unless fm (setq fm 'a))
     (setq filename ($filep fn ft fm))
     (cond
      ((null filename)
       (\|sayKeyedMsg\| 's2iv0003 (list fn ft fm)))
      (setq teststream (make-outstream filename 255 0))
      (shut $\|algebraOutputStream\))
      (setq $\|algebraOutputStream\| teststream)
      (setq $\|algebraOutputFile\| (\|object2String\| filename))
      (\|sayKeyedMsg\| 's2iv0004 (list "\text{Algebra}" $\|algebraOutputFile\|))
      (t (\|sayKeyedMsg\| 's2iv0003 (list fn ft fm))))
    (t
defun describeSetOutputAlgebra

[sayBrightly p??]

(setOutputAlgebra p763)

— defun describeSetOutputAlgebra —

(defun |describeSetOutputAlgebra| ()
  (|sayBrightly| (list
    '|%b| "set output algebra"
    '|%d| "is used to tell AXIOM to turn algebra-style output"
    '|%l| "printing on and off, and where to place the output. By default, the"
    '|%l| "destination for the output is the screen but printing is turned off."
    '|%l| "Syntax: )set output algebra <arg>"
    '|%l| "where arg can be one of"
    '|%l| "on turn algebra printing on (default state)"
    '|%l| "off turn algebra printing off"
    '|%l| "console send algebra output to screen (default state)"
    '|%l| "fp<.fe> send algebra output to file with file prefix fp"
    '|%l|
    "and file extension .fe. If not given, .fe defaults to .spout."
    '|%l| "If you wish to send the output to a file, you may need to issue this command"
    '|%l| "twice: once with"
    '|%b| "on"
    '|%d| "and once with the file name. For example, to send"
    '|%l| "algebra output to the file"
    '|%b| "polymer.spout."
    '|%d| "issue the two commands"
    '|%l| " )set output algebra on"
    '|%l| " )set output algebra polymer"
    '|%l| "The output is placed in the directory from which you invoked AXIOM or"
    '|%l| "the one you set with the )cd system command."
    '|%l| "The current setting is:"
    '|%b| (setqOutputAlgebra| '|%display%|)
    '|%d|)))

— defun describeSetOutputAlgebra —
characters

-------------------- The characters Option --------------------

Description: choose special output character set

The characters option may be followed by any one of the following:

   default
-> plain

The current setting is indicated. This option determines the special characters used for algebraic output. This is what the current choice of special characters looks like:

   ulc is shown as +    urc is shown as +
   llc is shown as +    lrc is shown as +
   vbar is shown as |   hbar is shown as -
   quad is shown as ?   lbrk is shown as [   rbrk is shown as ]
   rbrk is shown as ]   lbrc is shown as {   rbrc is shown as }
   btee is shown as +   rtee is shown as +
   ltee is shown as +   ctee is shown as +
   bslash is shown as \\"
— defun setOutputCharacters —

(defun setOutputCharacters (arg)
  (let (current char s l fn)
    (declare (special "$specialCharacters" "$plainRTspecialCharacters" "$RTspecialCharacters" "$specialCharacterAlist"))
    (if (eq arg '|%initialize%|)
      (setq "$specialCharacters" "$plainRTspecialCharacters")
    (progn
      (setq current
        (cond
          ((eq "$specialCharacters" "$RTspecialCharacters") "default")
          ((eq "$specialCharacters" "$plainRTspecialCharacters") "plain")
          (t "unknown")))
        (setq current
          (cond
            ((or (null arg) (eq arg '|%display%|) (eq (car arg) '?))
              (sayMessage
                "The characters option may be followed by any one of the following:")
              (dolist (name '("default" "plain")
                (if (string= (string current) name)
                  (sayBrightly
                    (list " ->" name)))))
            (terpri)
            (sayBrightly
              "The current setting is indicated within the list. This option determines")
            (sayBrightly
              "the special characters used for algebraic output. This is what the")
            (sayBrightly
              "current choice of special characters looks like:")
            (do ((t1 $specialCharacterAlist| (CDR t1)) (t2 nil))
              ((or (atom t1)
                (progn (setq t2 (car t1)) nil)
                (progn (progn (setq char (car t2)) t2) nil)) nil)
              (setq s
                (concat " " (pname char) " is shown as "
                (pname (|specialChar| char))))))
  )
)
fortran

--------------------- The fortran Option ---------------------

Description: create output in FORTRAN format

)set output fortran is used to tell AXIOM to turn FORTRAN-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Also See: )set fortran

Syntax: )set output fortran <arg>

where arg can be one of

on turn FORTRAN printing on
off turn FORTRAN printing off (default state)
console send FORTRAN output to screen (default state)
fp<.fe> send FORTRAN output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .sfort.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send FORTRAN output to the file polymer.sf, issue the two commands

)set output fortran on
)set output fortran polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: Off:CONSOLE

defvar $fortranFormat

    — initvars —

(defvar $fortranFormat nil "create output in FORTRAN format")


defvar $fortranOutputFile

    — initvars —

(defvar $fortranOutputFile "CONSOLE"
"where FORTRAN output goes (enter \em console) or a a pathname")


defoutputfortran

(|fortran|
"create output in FORTRAN format"
|interpreter|
FUNCTION
|setOutputFortran|
("create output in FORTRAN format"
LITERALS
|$fortranFormat|
(|off| |on|)
|off|)
(|break| |$fortranFormat|)
("where FORTRAN output goes (enter \em console) or a a pathname")
FILENAME
|$fortranOutputFile|
|chkOutputFileName|"console")
NIL)
defun setOutputFortran

(defiofstream p982)
(concat p1047)
(describeSetOutputFortran p772)
(upcase p22)
(qcdr p22)
(qcar p22)
(member p1048)
(sayKeyedMsg p329)
(shut p982)
(pathnameType p1040)
(pathnameDirectory p1041)
(pathnameName p1040)
($filep p22)
(makeStream p983)
(object2String p22)
($fortranOutputStream p22)
($fortranOutputFile p769)
($filep p22)
($fortranFormat p769)

--- defun setOutputFortran ---

(defun setOutputFortran (arg)
(let (label APPEND quiet tmp1 tmp2 ptype fn ft fm filename teststream)
(declare (special $fortranOutputStream $fortranOutputFile $filep $fortranFormat))
(cond
  ((eq arg '|%initialize%|)
   (setq $fortranOutputStream
     (defiostream '((mode . output) (device . console)) 255 0))
   (setq $fortranOutputFile "CONSOLE")
   (setq $fortranFormat nil))
  ((eq arg '|%display%|)
   (if $fortranFormat
     (setq label "On:")
     (setq label "Off:"))
   (concat label $fortranOutputFile))
  ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
   (describeSetOutputFortran)))
(t
do ()
  ((null (and (listp arg)
              (member (upcase (car arg)) '(append quiet))))
    nil)
  (cond
    ((eq (upcase (car arg)) 'append) (setq append t))
...
((eq (upcase (car arg)) 'quiet) (setq quiet t))
  (t nil))
(setq arg (cdr arg)))
(cond
  ((and (consp arg)
    (eq (qcdr arg) nil)
    (progn (setq fn (qcar arg)) t)
    (member fn '(Y N YE YES NO O ON OF OFF CONSOLE
                  |yl| |ln| |yl| |yes| |no| |o| |on| |of| |off| |console|)))
    'ok))
  (t (setq arg (list fn '|sfort|)))))
(cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
    (cond
      ((member (upcase fn) 'y n ye o of))
        (sayKeypadMsg 's2iv0002 'fortran |fortran|))
      ((member (upcase fn) 'no off) (setq $fortranFormat nil))
      ((member (upcase fn) 'yes on) (setq $fortranFormat t))
      ((eq (upcase fn) 'console)
        (shut $fortranOutputStream)
        (setq $fortranOutputStream
          (defiostream '((mode . output) (device . console)) 255 0))
        (setq $fortranOutputFile "CONSOLE")))))
  (or
    (and (consp arg)
      (progn
        (setq fn (qcar arg))
        (setq tmp1 (qcdr arg))
        (and (consp tmp1)
          (eq (qcdr tmp1) nil)
          (progn (setq ft (qcar tmp1)) t))))
    (and (consp arg)
      (progn
        (setq fn (qcar arg))
        (setq tmp1 (qcdr arg))
        (and (consp tmp1)
          (progn
            (setq ft (qcar tmp1))
            (setq tmp2 (qcdr tmp1))
            (and (consp tmp2)
              (eq (qcdr tmp2) nil)
              (progn (setq fm (qcar tmp2)) (setq ft ptype))))))
    (when (setq ptype (pathnameType fn))
      (setq fn (concat (pathnameDirectory fn) (pathnameName fn)))
      (setq ft ptype))
    (unless fm (setq fm 'a))
    (setq filename ($filep fn ft fm))
    (cond
      (null filename)
        (sayKeypadMsg 'S2IV0003 (list fn ft fm)))
defun describeSetOutputFortran

[sayBrightly p??]
[setOutputFortran p770]

— defun describeSetOutputFortran —

(defun |describeSetOutputFortran| ()
  (|sayBrightly| (list
    '|%b| "\)set output fortran"
    '|%d| "is used to tell AXIOM to turn FORTRAN-style output"
    '|%l| "printing on and off, and where to place the output. By default, the"
    '|%l| "destination for the output is the screen but printing is turned off."
    '|%l|
    '|%l| "Also See: )set fortran"
    '|%l|
    '|%l| "Syntax: )set output fortran <arg>"
    '|%l| " where arg can be one of"
    '|%l| " on turn FORTRAN printing on"
    '|%l| " off turn FORTRAN printing off (default state)"
    '|%l| " console send FORTRAN output to screen (default state)"
    '|%l|
    " fp<.fe> send FORTRAN output to file with file prefix fp and file"
    '|%l| " extension .fe. If not given, .fe defaults to .sfort."
    '|%l|
    '|%l| "If you wish to send the output to a file, you must issue this command"
    '|%l| " twice: once with"
    '|%b| "on"
    '|%d| "and once with the file name. For example, to send"
    '|%l| "FORTRAN output to the file"
    '|%b| "polymer.sfort,"
    '|%d| "issue the two commands"
fraction

---------------- The fraction Option ----------------

Description: how fractions are formatted

The fraction option may be followed by any one of the following:

-> vertical
   horizontal

The current setting is indicated.

defvar $fractionDisplayType

    — initvars —

(defvar !$fractionDisplayType! '`vertical! "how fractions are formatted")

    — outputfraction —

(!fraction!
"how fractions are formatted"
interpreter!
LITERALS
!$fractionDisplayType!
(!vertical! !horizontal!)
!vertical!)
length

------------------------------------- The length Option -------------------------------------

Description: line length of output displays

The length option may be followed by an integer in the range 10 to 245 inclusive. The current setting is 77

defvar $margin

--- initvars ---

(defvar $margin 3)

---

defvar $linelength

--- initvars ---

(defvar $linelength 77 "line length of output displays")

---

--- outputlength ---

([length]
   "line length of output displays"
   [interpreter]
   INTEGER
   $LINELENGTH
   (10 245)
   77)
The mathml Option

Description: create output in MathML style

)set output mathml is used to tell AXIOM to turn MathML-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax:  )set output mathml <arg>

where arg can be one of
on        turn MathML printing on
off       turn MathML printing off (default state)
console   send MathML output to screen (default state)
fp<.fe>   send MathML output to file with file prefix fp
          and file extension .fe. If not given, .fe defaults to .smml.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send MathML output to the file polymer.smml, issue the two commands

)set output mathml on
)set output mathml polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: Off:CONSOLE

defvar $mathmlFormat

    — initvars —

(defvar $mathmlFormat nil "create output in MathML format")

defvar $mathmlOutputFile

    — initvars —

(defvar $mathmlOutputFile "CONSOLE")
"where MathML output goes (enter \{em console\} or a pathname)"

-----

— outputmathml —

(|outputmathml|
  "create output in MathML style"
|interpreter|
FUNCTION
|setOutputMathml|
(('create output in MathML format"
|literals|
|mathmlFormat|
(on |off)
|off|
('break |$mathmlFormat|)
('where MathML output goes (enter \{em console\} or a pathname)"
|filename|
|mathmlOutputFile|
chkOutputFileName|"console")
NIL)

-----

defun setOutputMathml

[defiostream p982]
[concat p1047]
[describeSetOutputMathml p778]
[qcdr p778]
[qcar p778]
|member p1048|
|upcase p778|
sayKeyedMsg p329]
|shut p982|
|pathnameType p1040|
|pathnameDirectory p1041|
|pathnameName p1040|
|$filep p778|
|make-outstream p981|
|object2String p778|
|$mathmlOutputStream p778|
|$mathmlOutputFile p778|
\[
\text{defun setOutputMathml} \ (\text{arg})
\]
\[
\text{let} \ (\text{label} \ tmp1 \ tmp2 \ ptype \ fn \ ft \ fm \ filename \ teststream)
\]
\[
\text{(declare} \ (\text{special} \ \text{$\text{mathmlOutputStream}$} \ \text{$\text{mathmlOutputFile}$} \ \text{$\text{mathmlFormat}$} \ \\
\text{$\text{filep}$})))
\]
\[
\text{(cond}
\]
\[
((\text{eq} \ \text{arg} \ '|%initialize%'|)
\]
\[
(\text{setq} \ \text{$\text{mathmlOutputStream}$}
\]
\[
(\text{defiostream} \ ((\text{mode} \ . \ \text{output}) \ (\text{device} \ . \ \text{console})) \ 255 \ 0))
\]
\[
(\text{setq} \ \text{$\text{mathmlOutputFile}$} \ "CONSOLE")
\]
\[
(\text{setq} \ \text{$\text{mathmlFormat}$} \ \text{nil})
\]
\[
))
\]
\[
((\text{eq} \ \text{arg} \ '|%display%'|)
\]
\[
(\text{if} \ \text{$\text{mathmlFormat}$}
\]
\[
(\text{setq} \ \text{label} \ "On:"))
\]
\[
(\text{setq} \ \text{label} \ "Off:"))
\]
\[
(\text{concat} \ \text{label} \ \text{$\text{mathmlOutputFile}$}))
\]
\[
((\text{or} \ \text{null} \ \text{arg}) \ (\text{eq} \ \text{arg} \ '|%describe%'|) \ (\text{eq} \ (\text{car} \ \text{arg}) \ '?'))
\]
\[
(\text{(|describeSetOutputMathml|}))
\]
\[
(t
\]
\[
\text{(cond}
\]
\[
((\text{and} \ (\text{consp} \ \text{arg})
\]
\[
(\text{eq} \ (\text{qcdr} \ \text{arg}) \ \text{nil})
\]
\[
(\text{progn} \ (\text{setq} \ \text{fn} \ (\text{qcar} \ \text{arg}) \ \text{t})
\]
\[
(\text{|member|} \ \text{fn} \ '(y \ n \ ye \ yes \ no \ on \ of \ off \ console)
\]
\[
|y| \ |n| \ |ye| \ |yes| \ |no| \ |on| \ |of| \ |off| \ |console|)))
\]
\[
'|ok|)
\]
\[
(t \ (\text{setq} \ \text{arg} \ (\text{list} \ \text{fn} \ '|\text{smml}|))))
\]
\[
\text{(cond}
\]
\[
((\text{and} \ (\text{consp} \ \text{fn})
\]
\[
(\text{eq} \ (\text{qcdr} \ \text{fn}) \ \text{nil})
\]
\[
(\text{progn} \ (\text{setq} \ \text{fn} \ (\text{qcar} \ \text{fn}) \ \text{t}))
\]
\[
(\text{cond}
\]
\[
((\text{|member|} \ (\text{upcase} \ \text{fn}) \ '(y \ n \ ye \ of))
\]
\[
(\text{|sayKeyedMsg|} \ 's2iv0002 \ '((\text{MathML} \ \text{mathml})))
\]
\[
((\text{|member|} \ (\text{upcase} \ \text{fn}) \ 'no \ off)) \ (\text{setq} \ \text{$\text{mathmlFormat}$} \ \text{nil})
\]
\[
((\text{|member|} \ (\text{upcase} \ \text{fn}) \ 'yes \ on)) \ (\text{setq} \ \text{$\text{mathmlFormat}$} \ \text{t})
\]
\[
((\text{eq} \ (\text{upcase} \ \text{fn}) \ 'console)
\]
\[
(\text{shut} \ \text{$\text{mathmlOutputStream}$})
\]
\[
(\text{setq} \ \text{$\text{mathmlOutputStream}$}
\]
\[
(\text{defiostream} \ ((\text{mode} \ . \ \text{output}) \ (\text{device} \ . \ \text{console})) \ 255 \ 0))
\]
\[
(\text{setq} \ \text{$\text{mathmlOutputFile}$} \ "CONSOLE")))
\]
\[
(\text{|or}
\]
\[
(\text{and} \ (\text{consp} \ \text{arg})
\]
\[
(\text{progn}
\]
\[
(\text{setq} \ \text{fn} \ (\text{qcar} \ \text{arg}))
\]
\[
(\text{setq} \ \text{tmp1} \ (\text{qcdr} \ \text{arg}))
\]
\[
)
(and (consp tmp1)
   (eq (qcdr tmp1) nil)
   (progn (setq ft (qcar tmp1)) t)))

(and (consp arg)
   (progn (setq fn (qcar arg))
         (setq tmp1 (qcdr arg))
         (and (consp tmp1)
               (progn
                 (setq ft (qcar tmp1))
                 (setq tmp2 (qcdr tmp1))
                 (and (consp tmp2)
                      (eq (qcdr tmp2) nil)
                      (progn
                        (setq fm (qcar tmp2))
                        t))))))

(when (setq ptype (|pathnameType| fn))
   (setq fn
         (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
   (setq ft ptype))

(unless fm (setq fm 'a))

(setq filename ($filep fn ft fm))

(cond
   ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
   ((setq teststream (make-outstream filename 255 0))
    (shut |$mathmlOutputStream|)
    (setq |$mathmlOutputStream| teststream)
    (setq |$mathmlOutputFile| (|object2String| filename))
    (|sayKeyedMsg| 's2iv0004 (list "MathML" |$mathmlOutputFile|)))
   (t (|sayKeyedMsg| 's2iv0003 (list fn ft fm))))

(t
   (|sayKeyedMsg| 's2iv0005 nil)
   (|(describeSetOutputMathml|))))

---

defun describeSetOutputMathml

[sayBrightly p??]
[setOutputMathml p776]

— defun describeSetOutputMathml —

(defun |describeSetOutputMathml| ()
  (|sayBrightly| (LIST
    '|%b| "set output mathml"
    '|%d| "is used to tell AXIOM to turn MathML-style output"
    '|%l| "printing on and off, and where to place the output. By default, the"
'\%l| "destination for the output is the screen but printing is turned off."
'\%l|
'\%l| "Syntax: )set output mathml <arg>"
'\%l| " where arg can be one of"
'\%l| " on turn MathML printing on"
'\%l| " off turn MathML printing off (default state)"
'\%l| " console send MathML output to screen (default state)"
'\%l| " fp<.fe> send MathML output to file with file prefix fp and file"
'\%l| " extension .fe. If not given, .fe defaults to .stex."
'\%l|
'\%l| "If you wish to send the output to a file, you must issue this command"
'\%l| " twice: once with"
'\%l| "on"
'\%l| "and once with the file name. For example, to send"
'\%l| "MathML output to the file"
'\%l| " polymer.smm1,"
'\%l| "issue the two commands"
'\%l|
'\%l| " )set output mathml on"
'\%l| " )set output mathml polymer"
'\%l|
'\%l| " The output is placed in the directory from which you invoked AXIOM or"
'\%l| " the one you set with the )cd system command."
'\%l| " The current setting is: "
'\%l| (\%display\%)
'\%l|

**html**

----------------------- The html Option ------------------------

**Description:** create output in html style

)set output html is used to tell AXIOM to turn html-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

**Syntax:**  )set output html <arg>

where arg can be one of

- **on** turn html printing on
- **off** turn html printing off (default state)
- **console** send html output to screen (default state)
- **fp<.fe>** send html output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .html.
If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send MathML output to the file polymer.html, issue the two commands

\( \)set output html on
\( \)set output html polymer

The output is placed in the directory from which you invoked Axiom or the one you set with the \( \)cd system command.

The current setting is: Off:CONSOLE

defvar $htmlFormat

— initvars —

(defvar $htmlFormat nil "create output in HTML format")

-------

defvar $htmlOutputFile

— initvars —

(defvar $htmlOutputFile "CONSOLE"
"where HTML output goes (enter \{em console\} or a pathname")

-------

— outputhtml —

(|html|
"create output in HTML style"
|interpreter|
FUNCTION
|setOutputHtml|
("create output in HTML format"
LITERALS
|$htmlFormat|
(|off| |on|)
|off|)
(|break||$htmlFormat|))
defun setOutputHtml

(defun |setOutputHtml| (arg)  
(let (label tmp1 tmp2 ptype fn ft fm filename teststream)  
  (declare (special |$htmlOutputStream| |$htmlOutputFile| |$htmlFormat| |$filep|))  
  (cond  
    ((eq arg '|%initialize%|)  
     (setq |$htmlOutputStream|  
           (defiostream '((mode . output) (device . console)) 255 0))  
     (setq |$htmlOutputFile| "CONSOLE")  
     (setq |$htmlFormat| nil)  
     ((eq arg '|%display%|)  
      (if |$htmlFormat|  
        (setq label "On:"))  
      (setq label "On:")))

("where HTML output goes (enter \em console) or a pathname")
FILENAME
|$htmlOutputFile|
|chkOutputFileName|
"console")
NIL)
(setq label "Off: \n")
(concat label |$htmlOutputFile|)
(or (null arg) (eq arg '|%describe%|) (eq (car arg) ')?)
(describeSetOutputHtml)
(t
(cond
((and (consp arg)
    (eq (qcdr arg) nil)
    (progn (setq fn (qcar arg)) t)
    (member fn '(y n ye yes no o on of off console
                 |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
 '|ok|)
(t (setq arg (list fn '|smml|)))))
(cond
((and (consp arg)
    (eq (qcdr arg) nil)
    (progn (setq fn (qcar arg)) t))
(cond
((member (upcase fn) '(y n ye o of))
 (sayKeyedMsg 's2iv0002 '(|HTML| |html|)))
((member (upcase fn) '(no off)) (setq |$htmlFormat| nil))
((member (upcase fn) '(yes on)) (setq |$htmlFormat| t))
(eq (upcase fn) 'console)
(shut |$htmlOutputStream|)
(setq |$htmlOutputStream|
  (defiostream '((mode . output) (device . console)) 255 0))
(setq |$htmlOutputFile| "CONSOLE")))))
(or
(and (consp arg)
  (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (eq (qcdr tmp1) nil)
      (progn (setq fn (qcar tmp1)) t))))
(and (consp arg)
  (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (progn
       (setq ft (qcar tmp1))
       (setq tmp2 (qcdr tmp1))
       (and (consp tmp2)
         (eq (qcdr tmp2) nil)
         (progn
          (setq fn (qcar tmp2))
          t)))))))
(when (setq ptype (pathnameType fn))
  (setq fn
    (concat ([pathnameDirectory] fn) ([pathnameName] fn)))

(setq ft ptype)
(unless fm (setq fm 'a))
(setq filename ($filep fn ft fm))
(cond
  ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
  ((setq teststream (make-outstream filename 255 0))
   (shut |$htmlOutputStream|)
   (setq |$htmlOutputStream| teststream)
   (setq |$htmlOutputFile| (|object2String| filename))
   (|sayKeyedMsg| 's2iv0004 (list "HTML" |$htmlOutputFile|)))
  (t (|sayKeyedMsg| 's2iv0003 (list fn ft fm))))
(t
 (|sayKeyedMsg| 's2iv0005 nil)
 (|describeSetOutputHtml|))))

---

defun describeSetOutputHtml

[sayBrightly p??]
[setOutputHtml p781]

defun describeSetOutputHtml ()
  (|sayBrightly| (LIST
    '|%b| "set output html"
    '|%d| "is used to tell AXIOM to turn HTML-style output"
    '|%l| "printing on and off, and where to place the output. By default, the"
    '|%l| "destination for the output is the screen but printing is turned off."
    '|%l| "Syntax: )set output html <arg>"
    '|%l| "where arg can be one of"
    '|%l| " on turn HTML printing on"
    '|%l| " off turn HTML printing off (default state)"
    '|%l| " console send HTML output to screen (default state)"
    '|%l| " fp<.fe> send HTML output to file with file prefix fp and file"
    '|%l| " extension .fe. If not given, .fe defaults to .stex."
    '|%l| "If you wish to send the output to a file, you must issue this command"
    '|%l| " twice: once with"
    '|%b| "on"
    '|%d| "and once with the file name. For example, to send"
    '|%l| "HTML output to the file"
    '|%b| "polymer.smml,"
    '|%d| "issue the two commands"
    '|%l|)
openmath

----------------------- The openmath Option -----------------------

Description: create output in OpenMath style

)set output tex is used to tell AXIOM to turn OpenMath output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax: )set output tex <arg>
where arg can be one of
on turn OpenMath printing on
off turn OpenMath printing off (default state)
console send OpenMath output to screen (default state)
fp<.fe> send OpenMath output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .sopen.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send OpenMath output to the file polymer.sopen, issue the two commands

)set output openmath on
)set output openmath polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: Off:CONSOLE

defvar $openMathFormat

— initvars —
(defvar $openMathFormat nil "create output in OpenMath format")

defvar $openMathOutputFile

— initvars —

(defvar $openMathOutputFile "CONSOLE"
  "where TeX output goes (enter \em console) or a pathname")

— outputopenmath —

(openmath
  "create output in OpenMath style"
  (interpreter)
  (setOutputOpenMath)
  ("create output in OpenMath format"
   (LITERALS
    $openMathFormat
    (off | on)
    off
    (break $openMathFormat))
  ("where TeX output goes (enter \em console) or a pathname")
  (FILENAME
   $openMathOutputFile
   (chkOutputFileName)
   "console")
  NIL)

—

defun setOutputOpenMath

[defostream p982]
[concat p1047]
[describeSetOutputOpenMath p788]
[qcdr p??]
[qcar p??]
CHAPTER 45. )SET HELP PAGE COMMAND

(defun |setOutputOpenMath| (arg)
  (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special |$openMathOutputStream| |$openMathFormat| |filep| |openMathOutputFile|))
    (cond
      ((eq arg '|%initialize%|)
       (setq |$openMathOutputStream| (defiostream '((mode . output) (device . console)) 255 0))
       (setq |$openMathOutputFile| "CONSOLE")
       (setq |$openMathFormat| NIL))
      ((eq arg '|%display%|)
       (if |$openMathFormat|
        (setq label "On:"))
        (setq label "Off:"))
      (cond
        ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
         (|describeSetOutputOpenMath|))
        (t
         (cond
          ((and (consp arg)
                (eq (qcdr arg) nil)
                (progn (setq fn (qcar arg)) t)
                (member fn '(y n ye yes no o on of off console |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
           '|ok|)
          (t (setq arg (list fn '|som|))))))
    (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t))
        (cond
         ...)
(((|member| (upcase fn) '(y n ye o of))
  (|sayKeyedMsg| 's2iv0002 '(|OpenMath| |openmath|)))
(((|member| (upcase fn) '(no off)) (setq |$openMathFormat| nil))
(((|member| (upcase fn) '(yes on)) (setq |$openMathFormat| t))
((eq (upcase fn) 'console)
  (shut |$openMathOutputStream|)
  (setq |$openMathOutputStream|
    (defiostream '((mode . output) (device . console)) 255 0))
  (setq |$openMathOutputFile| "CONSOLE")))))
((or
  (and (consp arg)
    (progn (setq fn (qcar arg))
      (setq tmp1 (qcdr arg))
      (and (consp tmp1)
        (eq (qcdr tmp1) nil)
        (progn (setq ft (qcar tmp1)) t))))
  (and (consp arg)
    (progn
      (setq fn (qcar arg))
      (setq tmp1 (qcdr arg))
      (and (consp tmp1)
        (progn (setq ft (qcar tmp1))
          (setq tmp2 (qcdr tmp1))
          (and (consp tmp2)
            (eq (qcdr tmp2) nil)
            (progn (setq fm (qcar tmp2)) t))))))
  (when (setq ptype (|pathnameType| fn))
    (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
    (setq ft ptype))
  (unless fm (setq fm 'a))
  (setq filename ($filep fn ft fm))
  (cond
    (null filename)
    (|sayKeyedMsg| 's2iv0003 (list fn ft fm))
    (setq teststream (make-outstream filename 255 0))
    (shut |$openMathOutputStream|)
    (setq |$openMathOutputStream| teststream)
    (setq |$openMathOutputFile| (|object2String| filename))
    (|sayKeyedMsg| 's2iv0004 (list "OpenMath" |$openMathOutputFile|))
    (t
     (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
    (t
     (|sayKeyedMsg| 's2iv0005 nil)
     (|describeSetOutputOpenMath|)))))))
defun describeSetOutputOpenMath

[sayBrightly p??]
[setOutputOpenMath p785]

— defun describeSetOutputOpenMath —

(defun |describeSetOutputOpenMath| ()
  (sayBrightly! (list
    '|%b| ")set output openmath"
    '|%d| "is used to tell AXIOM to turn OpenMath output"
    '|%l| "printing on and off, and where to place the output. By default, the"
    '|%l| "destination for the output is the screen but printing is turned off."
    '|%l| "Syntax: )set output openmath <arg>"
    '|%l| "  where arg can be one of"
    '|%l| "  on turn OpenMath printing on"
    '|%l| "  off turn OpenMath printing off (default state)"
    '|%l| "  console send OpenMath output to screen (default state)"
    '|%l| "  fp<.fe> send OpenMath output to file with file prefix fp and file"
    '|%l| "    extension .fe. If not given, .fe defaults to .som."
    '|%l| "If you wish to send the output to a file, you must issue this command"
    '|%b| " twice: once with"
    '|%b| "on"
    '|%d| "and once with the file name. For example, to send"
    '|%l| "OpenMath output to the file"
    '|%b| "polymer.som."
    '|%d| "issue the two commands"
    '|%l| "  )set output openmath on"
    '|%l| "  )set output openmath polymer"
    '|%l| "The output is placed in the directory from which you invoked AXIOM or"
    '|%l| "the one you set with the )cd system command."
    '|%l| "The current setting is: "
    '|%b| ((setOutputOpenMath! '')|%display%))
    '|%d)))

script

---------------------- The script Option ----------------------

Description: display output in SCRIPT formula format
set output script is used to tell AXIOM to turn IBM Script formula-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax:  )set output script <arg>

where arg can be one of

- on   turn IBM Script formula printing on
- off  turn IBM Script formula printing off
        (default state)
- console send IBM Script formula output to screen
        (default state)
- fp<.fe> send IBM Script formula output to file with file
          prefix fp and file extension .fe. If not given,
          .fe defaults to .sform.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send IBM Script formula output to the file polymer.sform, issue the two commands

)set output script on
)set output script polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command. The current setting is: Off:CONSOLE

defvar $formulaFormat

— initvars —

(defvar |$formulaFormat| nil "display output in SCRIPT format")

——

defvar $formulaOutputFile

— initvars —

(defvar |$formulaOutputFile| "CONSOLE"
     "where script output goes (enter \em console) or a a pathname")
---

outputscript

\(\text{(|script|)}\)
"display output in SCRIPT formula format"
\(\text{|interpreter|}\)
FUNCTION
\(\text{|setOutputFormula|}\)
\(\text{(|"display output in SCRIPT format"|)}\)
LITERALS
\(\text{|$formulaFormat|}\)
\(\text{|(off| on)|}\)
\(\text{|off|}\)
\(\text{|(break| \$formulaFormat|)}\)
\(\text{("where script output goes (enter \{em console\} or a a pathname")|)}\)
FILENAME
\(\text{|$formulaOutputFile|}\)
\(\text{|chkOutputFileName|}\)
"console")\)
NIL

---

defun setOutputFormula

defostream p982
concat p1047
describeSetOutputFormula p792
qcdr p??
qcar p??
member p1048
upcase p??
sayKeyedMsg p329
shut p982
pathnameType p1040
pathnameDirectory p1041
pathnameName p1040
$file p??
make-outstream p981
object2String p??
$formulaOutputStream p??
$formulaOutputStream p??
$formulaOutputFile p789
$filep p??
$formulaFormat p789
(defun setOutputFormula |arg|
  (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special |$formulaOutputStream| |$formulaOutputFile| $filep
                  |$file| |$formulaFormat|))
    (cond
      ((eq arg |%initialize%|)
       (setq |$formulaOutputStream|
             (defiostream '((mode . output) (device . console)) 255 0))
       (setq |$formulaOutputFile| "CONSOLE")
       (setq |$formulaFormat| nil))
      ((eq arg |%display%|)
       (if |$formulaFormat|
           (setq label "On:"))
       (setq label "Off:"))
      (cond
        ((or (null arg) (eq arg |%describe%|) (eq (car arg) '?))
         (|describeSetOutputFormula|))
        (t
         (cond
          ((and (consp arg)
                 (eq (qcdr arg) nil))
           (progn (setq fn (qcar arg)) t)
           (|member| fn '(y n ye yes no o on of off console)
                     |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|))
           'ok)
         (t (setq arg (list fn |$sform|))))))
    (cond
      ((and (consp arg)
             (eq (qcdr arg) nil))
       (progn (setq fn (qcar arg)) t))
      (cond
        ((|member| (upcase fn)'(y n ye o of))
         (|sayKeyedMsg| 's2iv0002 '(|script| |script|)))
        ((|member| (upcase fn) '(no off)) (setq |$formulaFormat| nil))
        ((|member| (upcase fn) '(yes on)) (setq |$formulaFormat| t))
        (eq (upcase fn) 'console)
        (SHUT |$formulaOutputStream|)
        (setq |$formulaOutputStream|
              (defiostream '((mode . output) (device . console)) 255 0))
        (setq |$formulaFormat| "CONSOLE"))
      (or
       (and (consp arg)
            (progn (setq fn (qcar arg)))
            (setq tmp1 (qcdr arg))
            (and (consp tmp1)
                 (eq (qcdr tmp1) nil)
                 (progn (setq ft (qcar tmp1)) t))))
(and (consp arg)
 (progn (setq fn (qcar arg))
 (setq tmp1 (qcdr arg))
 (and (consp tmp1)
   (progn (setq ft (qcar tmp1))
     (setq tmp2 (qcdr tmp1))
     (and (consp tmp2)
       (eq (qcdr tmp2) nil)
       (progn
         (setq fm (qcar tmp2)) t)))))))
(if (setq ptype (|pathnameType| fn))
 (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
 (setq ft ptype))
(unless fm (setq fm 'a))
(setq filename ($filep fn ft fm))
(cond
 ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
 ((setq teststream (make-outstream filename 255 0))
   (shut |$formulaOutputStream|)
   (setq |$formulaOutputStream| teststream)
   (setq |$formulaOutputFile| (|object2String| filename))
   (|sayKeyedMsg| 's2iv0004
    (list "IBM Script formula" |$formulaOutputFile| )))
 (t
   (|sayKeyedMsg| 's2iv0003 (list fn ft fm))))
(t
 (|sayKeyedMsg| 's2iv0003 (list fn ft fm))))

---

defun describeSetOutputFormula

[sayBrightly p??]
[setOutputFormula p790]

— defun describeSetOutputFormula —

(defun describeSetOutputFormula ()
 (|sayBrightly| (list
 '('|%b| ")set output script"
 '('|%d| "is used to tell Axiom to turn IBM Script formula-style"
 '('|%l| "output printing on and off, and where to place the output. By default, the"
 '('|%l| "destination for the output is the screen but printing is turned off."
 '('|%l| "Syntax:  )set output script <arg>"
The scripts Option

Description: show subscripts,... linearly

The scripts option may be followed by any one of the following:

yes
no

The current setting is indicated.

defvar $linearFormatScripts

— initvars —
(defvar $linearFormatScripts nil "show subscripts,... linearly")

———

— outputscripts —

(|scripts|
 "show subscripts,... linearly"
 |interpreter|
 LITERALS
 $linearFormatScripts|
 (|on| |off|)
 |off|)

———

showeditor

---------------------- The showeditor Option ----------------------

Description: view output of )show in editor

The showeditor option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $useEditorForShowOutput

——— initvars ——

(defvar $useEditorForShowOutput nil "view output of )show in editor")

———

— outputshoweditor —

(|showeditor|
"view output of )show in editor"

LITERALS
|$useEditorForShowOutput|
(|on| |off|)
|off|)

-------------
tex

----------------------- The tex Option ------------------------

Description: create output in TeX style

)set output tex is used to tell AXIOM to turn TeX-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax:  )set output tex <arg>
where arg can be one of
 on     turn TeX printing on
 off    turn TeX printing off (default state)
 console send TeX output to screen (default state)
 fp<.fe> send TeX output to file with file prefix fp
 and file extension .fe. If not given, .fe defaults to .stex.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send TeX output to the file polymer.stex, issue the two commands

)set output tex on
)set output tex polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: Off:CONSOLE

defvar $texFormat

— initvars —
(defvar |$texFormat| nil "create output in TeX format")
defvar $texOutputFile

— initvars —

(defvar $texOutputFile "CONSOLE"
 "where TeX output goes (enter \em console} or a pathname")

— outputtex —

(|tex|
 "create output in TeX style"
 |interpreter|
 FUNCTION
 |setOutputTex|
 ("create output in TeX format"
 LITERALS
 |$texFormat|
 (|off| |on|)
 |off|)
 (|break| |$texFormat|)
 ("where TeX output goes (enter \em console} or a pathname"
 FILENAME
 |$texOutputFile|
 |chkOutputFileName|
 "console")
 NIL)

defun setOutputTex

[defiostream p982]
[concat p1047]
[describeSetOutputTex p798]
[qcdr p??]
[qcar p??]
[member p1048]
(defun |setOutputTex| (arg)
  (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special |$texOutputStream| |$texOutputFile| |$texFormat| $filep))
    (cond
      ((eq arg '|%initialize%|)
       (setq |$texOutputStream|
        (defiostream '((mode . output) (device . console)) 255 0))
       (setq |$texOutputFile| "CONSOLE")
       (setq |$texFormat| nil))
      ((eq arg '|%display%|) (if |$texFormat|
                     (setq label "On:")
                     (setq label "Off:"))
       (concat label |$texOutputFile|)
       (or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
       (|describeSetOutputTex|))
      (t
       (cond
         ((and (consp arg)
              (eq (qcdr arg) nil)
              (progn (setq fn (qcar arg)) t)
              (|member| fn '(y n ye yes no o on of off console
                    |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
          '|ok|)
       (t (setq arg (list fn '|stex| nil))))))))
    (cond
      ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
        (cond
          ((|member| (upcase fn) '(y n ye o of))
           (|sayKeyedMsg| 's2iv00002 '(|TeX| |tex|)))))
      ((|member| (upcase fn) '(no off)) (setq |$texFormat| nil))
      (t nil)))
(((|member| (upcase fn) '(yes on)) (setq |$texFormat| t))
((eq (upcase fn) 'console)
 (shut |$texOutputStream|)
 (setq |$texOutputStream|
 (defiostream '((mode . output) (device . console)) 255 0))
 (setq |$texOutputFile| "CONSOLE")))
((or
 (and (consp arg)
 (progn (setq fn (qcar arg))
 (setq tmp1 (qcdr arg))
 (and (consp tmp1)
 (eq (qcdr tmp1) nil)
 (progn (setq ft (qcar tmp1)) t))))
 (and (consp arg)
 (progn (setq fn (qcar arg))
 (setq tmp1 (qcdr arg))
 (and (consp tmp1)
 (progn (setq ft (qcar tmp1))
 (setq tmp2 (qcdr tmp1))
 (and (consp tmp2)
 (eq (qcdr tmp2) nil)
 (progn (setq fm (qcar tmp2)) t))))))
 (when (setq ptype (|pathnameType| fn))
 (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn))
 (setq ft ptype))
 (unless fm (setq fm 'A))
 (setq filename ($filep fn ft fm))
 (cond
 ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
 ((setq teststream (make-outstream filename 255 0))
 (shut |$texOutputStream|)
 (setq |$texOutputStream| teststream)
 (setq |$texOutputFile| (|object2String| filename))
 (|sayKeyedMsg| 's2iv0004 (list "TeX" |$texOutputFile|))
 (t (|sayKeyedMsg| 'S2IV0003 (list fn ft fm)))))

——

defun describeSetOutputTex

[sayBrightly p??]
[setOutputTex p796]

— defun describeSetOutputTex —
45.32 QUIT

----------------------- The quit Option -----------------------

Description: protected or unprotected quit

The quit option may be followed by any one of the following:

    protected
    -> unprotected

The current setting is indicated.
defvar $quitCommandType

— initvars —

(defvar $quitCommandType 'protected "protected or unprotected quit")


— quit —

{|quit|
 "protected or unprotected quit"
|interpreter|
LITERALS
|$quitCommandType|
(|protected| |unprotected|)
|protected|)


45.33 streams

Current Values of streams Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate</td>
<td>specify number of elements to calculate</td>
<td>10</td>
</tr>
<tr>
<td>showall</td>
<td>display all stream elements computed</td>
<td>off</td>
</tr>
</tbody>
</table>

— streams —

{|streams|
 "set some options for working with streams"
|interpreter|
TREE
|novar|
{
 \getchunk{streamscalculate}
 \getchunk{streamsshowall}
 }


calculate

------------------------ The calculate Option ------------------------

Description: specify number of elements to calculate

)set streams calculate is used to tell AXIOM how many elements of a stream to calculate when a computation uses the stream. The value given after calculate must either be the word all or a positive integer.

The current setting is 10.

defvar $streamCount

— initvars —

(defun |$streamCount| 10
  "number of initial stream elements you want calculated")

— streamscalculate —

(|calculate|
  "specify number of elements to calculate"
|interpreter|
FUNCTION
|setStreamsCalculate|
("number of initial stream elements you want calculated"
INTEGER
|$streamCount|
(0 NIL)
10)
NIL)

—

defun setStreamsCalculate

[object2String p?]
[describeSetStreamsCalculate p802]
[sayMessage p?]
[bright p?]
-- defun setStreamsCalculate --

(defun setStreamsCalculate (arg)
  (let (n)
    (declare (special |$streamCount|))
    (cond
      ((eq arg '|%initialize%|) (setq |$streamCount| 10))
      ((eq arg '|%display%|) (object2String |$streamCount|))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
        (describeSetStreamsCalculate))
      (t
        (setq n (car arg))
        (cond
          (and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
            (sayMessage)
            ('("Your value of" ,@(|bright| n) "is invalid because ..."))
            (describeSetStreamsCalculate)
            (terminateSystemCommand))
      (t (setq |$streamCount| n))))))

defun describeSetStreamsCalculate

[sayKeyedMsg p329]
[$streamCount p801]

-- defun describeSetStreamsCalculate --

(defun describeSetStreamsCalculate ()
  (declare (special |$streamCount|))
  (sayKeyedMsg 's2iv0001 (list |$streamCount|)))

showall

--------------------- The showall Option ---------------------

Description: display all stream elements computed

The showall option may be followed by any one of the following:
The current setting is indicated.

defvar $streamsShowAll

--- initvars ---

(defvar |$streamsShowAll| nil "display all stream elements computed")

---

--- streamsshowall ---

(|showall|
    "display all stream elements computed"
    |interpreter|
    LITERALS
    |$streamsShowAll|
    (|on| |off|)
    |off|)

---

45.34 system

Current Values of system Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>functioncode</td>
<td>show gen. LISP for functions when compiled off</td>
<td>off</td>
</tr>
<tr>
<td>optimization</td>
<td>show optimized LISP code</td>
<td>off</td>
</tr>
<tr>
<td>prettyprint</td>
<td>prettyprint BOOT func's as they compile</td>
<td>off</td>
</tr>
</tbody>
</table>

--- system ---

(|system|
    "set some system development variables"
functioncode

------------------- The functioncode Option -------------------

Description: show gen. LISP for functions when compiled

The functioncode option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $reportCompilation

  -- initvars --

(defvar $reportCompilation nil "show gen. LISP for functions when compiled")

  -- systemfunctioncode --

  (|functioncode|
   "show gen. LISP for functions when compiled"|development|
   LITERALS
   |$reportCompilation|
   (|on| |off|)
   |off|)
optimization

------------- The optimization Option -------------

Description: show optimized LISP code

The optimization option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

(defvar $reportOptimization
  ;; initvars
  (defvar $reportOptimization nil "show optimized LISP code")

-----

(defvar $reportOptimization
  ;; system optimization
  (optimization
   "show optimized LISP code"
   (development)
   (LITERALS
    ($reportOptimization
     (|on| |off|)
     |off|)

-----

prettyprint

------------- The prettyprint Option -------------

Description: prettyprint BOOT func’s as they compile
The prettyprint option may be followed by any one of the following:

   on
   -> off

The current setting is indicated.

`defvar $prettyprint`  

   — initvars —  

(defvar $prettyprint t "prettyprint BOOT func's as they compile")

   — systemprettyprint —  

(|prettyprint|
 "prettyprint BOOT func's as they compile"
 |development|
 LITERALS
 $prettyprint
 (|on| |off|)
 |on|)

45.35 userlevel  

-------------------------- The userlevel Option --------------------------

Description: operation access level of system user

The userlevel option may be followed by any one of the following:

   interpreter
   compiler
   -> development

The current setting is indicated.
defvar $UserLevel

— initvars —

def $UserLevel
interpreter "operation access level of system user"

———

— userlevel —

userlevel
interpreter "operation access level of system user"
interpreter |compiler| |development|

———

— initvars —

$setOptions

(interpreter |breakmode| |compile| |debug| |expose| |functions| |fortran| |kernel| |hyperdoc| |help| |history| |messages| |naglink| |output| |quit| |streams| |system| |userlevel|

)}
defvar $setOptionNames

— initvars —
(defvar $setOptionNames (mapcar #'car $setOptions))

——

— postvars —
(eval-when (eval load)
  (initializeSetVariables $setOptions))

——

45.36 Set code

defun set
[set1 p808]
[$setOptions p??]

— defun set —
(defun |set| (l)
  (declare (special |$setOptions|))
  (|set1| l |$setOptions|))

——

defun set1

This function will be called with the top level arguments to )set. For instance, given the command

)set break break

this function gets

(set1 (|break| |break|) ....)
and given the command

```lisp
)set mes auto off
```

this function gets

```lisp
(set1 (|mes| |auto| |off|) ....)
```

which, because “message” is a TREE, generates the recursive call:

```lisp
(set1 (|auto| |off|) <the message subtree>)
```

The “autoload” subtree contains a FUNCTION called printLoadMessages, which gets called with

```lisp
%describe% [displaySetVariableSettings p657]
```

```lisp
[seq p??] [exit p??] [selectOption p479] [ downcase p??] [ lassoc p??] [ satisfiesUserLevel p451] [sayKeyedMsg p329] [poundsign p??] [displaySetOptionInformation p655] [kdr p??] [sayMSG p331] [sayMessage p??] [bright p??] [object2String p??] [translateYesNo2TrueFalse p658] [use-fast-links p??] [literals p??] [tree p??] [set1 p808] [$setOptionNames p808] [$UserLevel p807] [$displaySetValue p748]
```

---

```lisp
(defun set1 (1 settree)
  (let ((|$setOptionNames| arg setData st setfunarg num upperlimit arg2)
        (declare (special |$setOptionNames| |$UserLevel| |$displaySetValue|))
        (cond ((null 1) (|displaySetVariableSettings| settree '||))
              (t (setq |$setOptionNames|
                       (do ((t1 settree (cdr t1)) t0 (x nil)))
```
((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
(seq
  (exit
    (setq t0 (cons (elt x 0) t0))))))
(setq arg
  (|selectOption| (downcase (car l)) |$setOptionNames| |optionError|)
(setq setdata (cons arg (lassoc arg settree)))
(cond
  ((null (|satisfiesUserLevel| (third setdata)))
    (|sayKeyedMsg| 's2iz0007 (list |$UserLevel| "set option" nil)))
  ((eql 1 (|#| l)) (|displaySetOptionInformation| arg setdata))
  (t
    (setq st (fourth setdata))
    (case (fourth setdata)
      (function
        (setq setfunarg
          (if (eq (elt l 1) 'default)
            '\|initialize|
            (kdr l)))
          (if (canFuncall? (fifth setdata))
            (funcall (fifth setdata) setfunarg)
            (|sayMSG| (concatenate 'string " Function not implemented. 
                                   " (string (fifth setdata)))))
        (when |$displaySetValue|
          (|displaySetOptionInformation| arg setdata))
        NIL)
      (string
        (setq arg2 (elt l 1))
        (cond
          ((eq arg2 'default) (set (fifth setdata) (seventh setdata)))
          (arg2 (set (fifth setdata) arg2))
          (t nil))
        (when (or |$displaySetValue| (null arg2))
          (|displaySetOptionInformation| arg setdata))
        NIL)
      (integer
        (setq arg2
          (progn
            (setq num (elt l 1))
            (cond
              ((and (integerp num)
                  (>= num (elt (sixth setdata) 0))
               (or (null (setq upperlimit (elt (sixth setdata) 1)))
                (<= num upperlimit)))
              (num)
              (t
                (|selectOption|
                 (elt l 1)
                 (cons '\|default| (sixth setdata)) nil)))))
            (cond

((eq arg2 'default) (set (fifth setdata) (seventh setdata)))
(arg2 (set (fifth setdata) arg2))
(t nil))
(cond
(or $displaySetValue| (null arg2))
($displaySetOptionInformation| arg setdata))
(cond
(null arg2)
(sayMessage| '(" Your value" ,0([bright| ([object2String| (elt l 1)]]
  "is not among the valid choices.\")))
(t nil)))
(literals
(cond
(setq arg2
(selectOption| (elt l 1))
(cons '|default| (sixth setdata) nil))
(cond
((eq arg2 'default)
(set (fifth setdata)
  (translateYesNo2TrueFalse| (seventh setdata))))
(t
  (cond ((eq arg2 '|nobreak|)
    #*:GCL (si::use-fast-links t)))
  (cond
    ((eq arg2 '|fastlinks|)
      #*:GCL (si::use-fast-links nil)
      (setq arg2 '|break|)))
  (set (fifth setdata) (translateYesNo2TrueFalse| arg2))))))
(when (or $displaySetValue| (null arg2))
  ($displaySetOptionInformation| arg setdata))
(cond
(null arg2)
(sayMessage| '(" Your value"
  (append ([bright| ([object2String| (elt l 1)]]
    (cons "is not among the valid choices." nil))))
  (t nil)))
(tree ([set| (kdr l) (sixth setdata)] nil)
(t
  (sayMessage|
    ('("Cannot handle set tree node type" ,0([bright| st] |yet|) nil)))))))
CHAPTER 45. \textit{\textsc{})set help page command}
Chapter 46

)show help page Command

46.1  show help page man page

— show.help —

====================================================================
A.22.  )show
====================================================================

User Level Required:  interpreter

Command Syntax:

- )show nameOrAbbrev
- )show nameOrAbbrev )operations
- )show nameOrAbbrev )attributes

Command Description:
This command displays information about AXIOM domain, package and category
constructors. If no options are given, the )operations option is assumed. For
example,

)show POLY
)show POLY )operations
)show Polynomial
)show Polynomial )operations

each display basic information about the Polynomial domain constructor and
then provide a listing of operations. Since Polynomial requires a Ring (for
example, Integer) as argument, the above commands all refer to a unspecified
ring R. In the list of operations, $ means Polynomial(R).
The basic information displayed includes the signature of the constructor (the name and arguments), the constructor abbreviation, the exposure status of the constructor, and the name of the library source file for the constructor.

If operation information about a specific domain is wanted, the full or abbreviated domain name may be used. For example,

)show POLY INT
)show POLY INT )operations
)show Polynomial Integer
)show Polynomial Integer )operations

are among the combinations that will display the operations exported by the domain Polynomial(Integer) (as opposed to the general domain constructor Polynomial). Attributes may be listed by using the )attributes option.

Also See:
  o )display
  o )set
  o )what

---

1

defun The )show command

[showSpad2Cmd p814]

— defun show —

(defun |show| (arg) (|showSpad2Cmd| arg))

---

defun The internal )show command

[member p1048]
[helpSpad2Cmd p572]
[sayKeyedMsg p329]
[qcar p??]
[reportOperations p815]
[$showOptions p??]

1 “display” (29.2 p 535) “set” (45.36 p 808) “what” (54.1 p 939)
### 46.1. SHOW HELP PAGE MAN PAGE

[|defun showSpad2Cmd| |
(defun |showSpad2Cmd| (arg)
  (let ((|$showOptions| |$e| |$env| constr)
    (declare (special |$showOptions| |$e| |$env| |$InteractiveFrame| |$options|))
    (if (equal arg (list nil))
      (|helpSpad2Cmd| '(|show|))
    (progn
      (setq |$showOptions| '(|attributes| |operations|))
      (unless |$options| (setq |$options| '((|operations|))))
      (setq |$e| |$InteractiveFrame|)
      (setq |$env| |$InteractiveFrame|)
      (cond
        ((and (consp arg) (eq (qcdr arg) nil) (progn (setq constr (qcar arg)) t))
         (cond
          ((|member| constr '(|Union| |Record| |Mapping|))
            (cond
              ((eq constr '|Record|)
               (|sayKeyedMsg| 'S2IZ0044R
                (list constr ")show Record(a: Integer, b: String)") )))
              ((eq constr '|Mapping|) (|sayKeyedMsg| 'S2IZ0044M nil))
              (t
               (|sayKeyedMsg| 'S2IZ0045T
                (list constr "show Union(a: Integer, b: String)") ))))
              ((and (consp constr) (eq (qcar constr) '|Mapping|))
               (|sayKeyedMsg| 'S2IZ0045U
                (list constr "show Union(Integer, String)") ))))
          ((and (consp constr) (eq (qcar constr) '|Mapping|))
            (|sayKeyedMsg| 'S2IZ0044M nil))
          (t (|reportOperations| constr constr)))
        (t (|reportOperations| arg arg)))))
    )))

---

defun reportOperations

[sayBrightly p??]
[bright p??]
[sayKeyedMsg p329]
[qcar p??]
isNameOfType p??]
isDomainValuedVariable p959]
[reportOpsFromUnitDirectly0 p821]
[opOf p??]
[unabbrev p??]
[reportOpsFromLisplib0 p817]
[evaluateType p916]
[mkAtree p??]
[removeZeroOneDestructively p??]
[isType p??]
[$env p??]
[$eval p??]
[$genValue p51]
[$quadSymbol p??]
[$doNotAddEmptyModeIfTrue p??]

--- defun reportOperations ---

(defvar reportOperations)
(declare (special $env $eval $genValue $quadSymbol $doNotAddEmptyModeIfTrue))
(setq $env (list (list nil)))
(setq $eval t)
(setq $genValue t)
(when u
  (setq $doNotAddEmptyModeIfTrue t)
  (cond
    (equal u $quadSymbol)
      (sayBrightly
        (cons " mode denotes" (append (|bright| "any") (list '|type|))))
    (eq u ")")
      (sayKeyedMsg 'S2IZ0063 nil)
      (sayKeyedMsg 'S2IZ0064 nil))
    ((and (null (and (consp u) (eq (qcar u) '?)))))
      (null (and (consp u) (eq (qcar u) '?)))
      (null (and (consp u)
        (eq (qcar u) '|typeOf|)
        (progn
          (setq tmp1 (qcdr u))
          (and (consp tmp1) (eq (qcdr tmp1) nil)))))
    (when (atom oldArg) (setq oldArg (list oldArg)))
      (sayKeyedMsg 'S2IZ0063 nil)
      (dolist (op oldArg)
        (sayKeyedMsg 'S2IZ0062 (list (|opOf| nil)))
        ((setq v (isDomainValuedVariable u)) (|reportOpsFromUnitDirectly0| v))
        (t
          (if (atom u)
            (setq unitForm (|opOf| (|unabbrev| u)))
            (setq unitForm (|unabbrev| u))))
          (println (|unabbrev| u))))
(if (atom unitForm)
     (|reportOpsFromLisplib0| unitForm u)
   (progn
     (setq unitFormp (|evaluateType| unitForm))
     (setq tree (|mkAtree| (|removeZeroOneDestructively| unitForm)))
     (if (setq unitFormp (|isType| tree))
         (|reportOpsFromUnitDirectly0| unitFormp)
         (|sayKeyedMsg| 'S2IZ0041 (list unitForm)))))
)

(defun reportOpsFromLisplib0
        [reportOpsFromLisplib1 p817]
        [reportOpsFromLisplib p818]
        [useEditorForShowOutput p794]

        — defun reportOpsFromLisplib0 —

    (defun |reportOpsFromLisplib0| (unitForm u)
      (declare (special |$useEditorForShowOutput|))
      (if |$useEditorForShowOutput|
          (|reportOpsFromLisplib1| unitForm u)
          (|reportOpsFromLisplib| unitForm u)))
)

(defun reportOpsFromLisplib1
        [pathname p1042]
        [erase p??]
        [defiostream p982]
        [sayShowWarning p826]
        [reportOpsFromLisplib p818]
        [shut p982]
        [editFile p545]
        [$sayBrightlyStream p??]
        [$erase p??]

        — defun reportOpsFromLisplib1 —

    (defun |reportOpsFromLisplib1| (unitForm u)
      (let (|$sayBrightlyStream| showFile)
        (declare (special |$sayBrightlyStream| $erase))

        (defun |reportOpsFromLisplib1| (unitForm u)
          (declare (special |$useEditorForShowOutput|))
          (if |$useEditorForShowOutput|
              (|reportOpsFromLisplib1| unitForm u)
              (|reportOpsFromLisplib| unitForm u)))
)

——
CHAPTER 46. SHOW HELP PAGE COMMAND

(defun reportOpsFromLisplib
    [constructor? p??]
    [sayKeyedMsg p329]
    [getConstructorSignature p??]
    [kdr p??]
    [getdatabase p1010]
    [eqsubstlist p??]
    [nreverse0 p??]
    [sayBrightly p??]
    [concat p1047]
    [bright p??]
    [form2StringWithWhere p??]
    [isExposedConstructor p820]
    [strconc p??]
    [namestring p1040]
    [selectOptionLC p479]
    [dc1 p??]
    [centerAndHighlight p??]
    [specialChar p980]
    [remdup p??]
    [nsort p??]
    [form2String p??]
    [say2PerLine p??]
    [formatAttribute p??]
    [displayOperationsFromLisplib p820]
    [$linelength p774]
    [$showOptions p??]
    [$options p??]
    [$FormalMapVariableList p??]

    — defun reportOpsFromLisplib —

(defun |reportOpsFromLisplib| (op u)
(let (fn s typ nArgs argList functorForm argml tmp1 functorFormWithDecl
   verb sourceFile opt attList)
  (declare (special $lineLength |showOptions| $options|
          |FormalMapVariableList|))
  (if (null (setq fn (|constructor?| op)))
      (|sayKeyedMsg| 'S2IZ0054 (list u))
      (progn
        (setq argml (when (setq s (|getConstructorSignature| op)) (kdr s)))
        (setq typ (getdatabase op 'constructorKind))
        (setq nArgs (! argml))
        (setq argList (kdr (getdatabase op 'constructorForm)))
        (setq functorForm (cons op argList))
        (setq argml (eqsubstlist argList |FormalMapVariableList| argml))
        (mapcar #'(lambda (a m) (push (list '|:| a m) tmp1)) argList argml)
        (setq functorFormWithDecl (cons op (nreverse0 tmp1)))
        (|sayBrightly|)
        (concat|bright| (|form2StringWithWhere| functorFormWithDecl))
        "is a" (|bright| typ) "constructor")
      (|sayBrightly|
        (cons " Abbreviation for" (append (|bright| op) (cons "is" (|bright| fn))))))
  (if (|isExposedConstructor| op)
      (setq verb "is")
      (setq verb "is not"))
  (|sayBrightly|
    (cons " This constructor"
      (append (|bright| verb) (list "exposed in this frame."))))
  (setq sourceFile (getdatabase op 'sourceFile))
  (|sayBrightly|
    (cons " Issue"
      (append (|bright| (strconc ")edit " (|nameString| sourceFile)))
      (cons "to see algebra source code for"
        (append (|bright| fn) (list '|%l|))))))
  (dolist (item |options|)
    (setq opt (|selectOptionLC| (car item) |showOptions| '|optionError|))
    (cond
      ((eq opt 'layout) (|dc1| fn))
      ((eq opt 'views)
        (|sayBrightly|
          (cons "To get" (append (|bright| "views")
            (list "you must give parameters of constructor"))))))
    (eq opt '|attributes|)
    (|centerAndHighlight| "Attributes" $lineLength (|specialChar| '|hbar|))
    (|sayBrightly| "=")
    (setq attList
      (remdup
        (msort
          (mapcar #'(lambda (x) (caar x))
            (reverse (getdatabase op 'attributes)))))
      (if (null attList)

defun isExposedConstructor

(defun isExposedConstructor (name)
  (let (x found)
    (declare (special $globalExposureGroupAlist $localExposureData))
    (cond
      ((member name '([Union |Record |Mapping]|) t)
       t)
      ((member name (elt $localExposureData 2)) nil)
      ((member name (elt $localExposureData 1)) t)
      (t
       (loop for g in (elt $localExposureData 0)
             when (not found)
             do
          (setq x (getalist $globalExposureGroupAlist g))
          (when (and x (getalist x name)) (setq found t))))
    found)))

defun displayOperationsFromLisplib

(defun displayOperationsFromLisplib
  (defun displayOperationsFromLisplib (functorForm)
    (|sayBrightly|
     (|concat| '|%b| (|form2String| functorForm)
       '|%d| '|has no attributes.| '|%l|)))
    (|say2PerLine| (mapcar #'|formatAttribute| attList)))))))
    ((eq opt '|operations|)
     (|displayOperationsFromLisplib| functorForm)))))))

|defun isExposedConstructor |

— defun isExposedConstructor —

(defun |isExposedConstructor| (name)
  (let (x found)
    (declare (special |$globalExposureGroupAlist| |$localExposureData|))
    (cond
      ((member name '([Union |Record |Mapping]|) t)
       t)
      ((member name (elt |$localExposureData| 2)) nil)
      ((member name (elt |$localExposureData| 1)) t)
      (t
       (loop for g in (elt |$localExposureData| 0)
             when (not found)
             do
          (setq x (getalist |$globalExposureGroupAlist| g))
          (when (and x (getalist x name)) (setq found t))))
    found)))

— defun displayOperationsFromLisplib —

(defun |displayOperationsFromLisplib| (functorForm)
  (|sayBrightly|
   (|concat| '|%b| (|form2String| functorForm)
     '|%d| '|has no attributes.| '|%l|)))
   (|say2PerLine| (mapcar #'|formatAttribute| attList)))))))
   ((eq opt '|operations|)
    (|displayOperationsFromLisplib| functorForm)))))))

— defun displayOperationsFromLisplib —
(defun displayOperationsFromLisplib (form)
  (let ((name (car form))
        (argl (cdr form))
        (kind (getdatabase name 'constructorkind))
        (opList (getdatabase name 'operationalist))
        (opList (remdup (msort (eqsubstlist argl name opList))))
        (ops nil))
    (dolist (x opList)
      (setq ops (append ops (formatOperationAlistEntry x))))
    (say2PerLine ops))

(defun reportOpsFromUnitDirectly0 (D)
  (declare (special $useEditorForShowOutput))
  (if $useEditorForShowOutput
      (reportOpsFromUnitDirectly1 D)
      (reportOpsFromUnitDirectly D)))

defun reportOpsFromUnitDirectly

(defun reportOpsFromUnitDirectly1 (D)
(defun reportOpsFromUnitDirectly (unitForm)
(let ((|$commentedOps| isRecordOrUnion unit top kind abb sourceFile verb opt attList constructorFunction tmp1 funlist a sigList tmp2)
(declare (special $commentedOps| $CategoryFrame| $linelength $options| $showOptions|)))
(setq isRecordOrUnion
(and (consp unitForm)
(progn (setq a (qcar unitForm)) t)
|member| a '(|Record| |Union|))))
(setq unit (|evalDomain| unitForm))
(setq top (car unitForm))
(setq kind (getdatabase top 'constructorkind))
|sayBrightly|
|concat| '|%b| (|formatOpType| unitForm) '|%d|
"is a" '|%b| kind '|%d| "constructor.")
(unless isRecordOrUnion
(setq abb (getdatabase top 'abbreviation))
(setq sourceFile (getdatabase top 'sourcefile))
(|sayBrightly|
 (cons " Abbreviation for"
  (append (|bright| top) (cons "is" (|bright| abb))))))
(if (|isExposedConstructor| top)
 (setq verb "is")
 (setq verb "is not"))
(|sayBrightly|
 (cons " This constructor"
  (append (|bright| verb) (list "exposed in this frame." )))))
(|sayBrightly|
 (cons " Issue"
  (append (|bright| (strconc ")edit " (|namestring| sourceFile)))
   (cons "to see algebra source code for"
    (append (|bright| abb) (list ’|%l|)))))))
(dolist (item |$options|)
 (setq opt (|selectOptionLC| (car item) |$showOptions| ’|optionError|))
 (cond
  ((eq opt ’|attributes|)
   (|centerAndHighlight| "Attributes" $linelength (|specialChar| ’|hbar|))
    (if isRecordOrUnion
     (|sayBrightly| " Records and Unions have no attributes.")
     (progn
      (|sayBrightly| "")
      (setq attList
       (remdup
        (msort
         (mapcar #'(lambda (unit2) (car unit2)) (reverse (elt unit 2)))))
      (|say2PerLine|
       (mapcar #'|formatAttribute| attList)
       nil)))))
  ((eq opt ’|operations|)
   (setq |$commentedOps| 0)
   ;; --new form is (<op> <signature> <slotNumber> <condition> <kind>)
   (|centerAndHighlight| "Operations" $linelength (|specialChar| ’|hbar|))
   (|sayBrightly| "")
   (cond
    (isRecordOrUnion
     (setq constructorFunction (getl top ’|makeFunctionList|))
     (unless constructorFunction
      (|systemErrorHere| "reportOpsFromUnitDirectly"))
     (setq tmp1
      (funcall constructorFunction ’$ unitForm |$CategoryFrame|))
     (setq funlist (car tmp1))
     (setq sigList
      (remdup
       (msort
        (dolist (fun funlist (nreverse0 tmp2))
         (push ‘((,(caar fun),,(cadar fun)) t ,,(caddar fun) 0 1)))))
        (|
defun getOplistForConstructorForm

The new form is an op-Alist which has entries

(<op> . signature-Alist)

where signature-Alist has entries

(<signature> . item)

where item has form (slotNumber; condition; kind).

(<slotNumber> <condition> <kind>)

where kind = ELT | CONST | Subsumed | (XLAM..)

<kind> = ELT | CONST | Subsumed | (XLAM..)

--- defun getOplistForConstructorForm ---
(loop for item in opAlist do
  (setq op (car item))
  (setq signatureAlist (cdr item))
  (setq result
    (append result
      (|getOplistWithUniqueSignatures| op pairlis signatureAlist))))
result))

---

defun getOplistWithUniqueSignatures

  — defun getOplistWithUniqueSignatures —

(defun |getOplistWithUniqueSignatures| (op pairlis signatureAlist)
  (let (sig slotNumber pred kind alist)
    (loop for item in signatureAlist
      when (not (eq (fourth item) '|Subsumed|))
        do
          (setq sig (first item))
          (setq slotNumber (second item))
          (setq pred (third item))
          (setq kind (fourth item))
          (setq alist
            (|insertAlist|
          (sublis pairlis (list op sig))
          (sublis pairlis (list pred (list kind nil slotNumber)))
          alist)))
    alist))

---

defun reportOpsFromUnitDirectly1

[pathname p1042]
[erase p??]
[defiostream p982]
[sayShowWarning p826]
[reportOpsFromUnitDirectly p821]
[shut p982]
[editFile p545]
[$sayBrightlyStream p??]
[$erase p??]
--- defun reportOpsFromUnitDirectly1 ---

(defun reportOpsFromUnitDirectly1 (D)
  (let ((|sayBrightlyStream| showFile)
        (declare (special |sayBrightlyStream| $erase))
        (setq showFile (pathname (list 'show 'listing 'a)))
        ($erase showFile)
        (setq |sayBrightlyStream|
              (defiostream `((file ,showFile) (mode . output)) 255 0))
        (sayShowWarning)
        (reportOpsFromUnitDirectly D)
        (shut |sayBrightlyStream|)
        (editFile showFile)))

-----

defun sayShowWarning

[sayBrightly p??]

--- defun sayShowWarning ---

(defun sayShowWarning ()
  (sayBrightly
   "Warning: this is a temporary file and will be deleted the next")
  (sayBrightly
   " time you use )show. Rename it and FILE if you wish to"
   (save the contents.")
  (sayBrightly ""))

-----
Chapter 47

)spool help page Command

47.1 spool help page man page

— spool.help —

====================================================================
A.23. )spool
====================================================================

User Level Required: interpreter

Command Syntax:

- )spool [fileName]
- )spool

Command Description:

This command is used to save (spool) all AXIOM input and output into a file, called a spool file. You can only have one spool file active at a time. To start spool, issue this command with a filename. For example,

)spool integrate.out

To stop spooling, issue )spool with no filename.

If the filename is qualified with a directory, then the output will be placed in that directory. If no directory information is given, the spool file will be placed in the current directory. The current directory is the directory from which you started AXIOM or is the directory you specified using the )cd command.
Also See:
  o )cd
Chapter 48

)summary help page Command

48.1  summary help page man page

— summary.help —

)credits   : list the people who have contributed to Axiom

)help <command> gives more information
)quit      : exit AXIOM

)abbreviation : query, set and remove abbreviations for constructors
)cd         : set working directory
)clear      : remove declarations, definitions or values
)close      : throw away an interpreter client and workspace
)compile    : invoke constructor compiler
)copyright  : show copyright and trademark information
)describe   : show database information for a category, domain, or package
)display    : display Library operations and objects in your workspace
)edit       : edit a file
)frame      : manage interpreter workspaces
)history    : manage aspects of interactive session
)library    : introduce new constructors
)lisp       : evaluate a LISP expression
)read       : execute AXIOM commands from a file
)savesystem : save LISP image to a file
)set        : view and set system variables
)show       : show constructor information
)spool      : log input and output to a file
)synonym    : define an abbreviation for system commands
)system     : issue shell commands
)trace      : trace execution of functions
)undo       : restore workspace to earlier state
defun summary
[obey p??]
[concat p1047]
[getenv p29]

--- defun summary ---

(defun summary (l)
  (declare (ignore l))
  (obey (concat "cat " (getenv "AXIOM") "/doc/spadhelp/summary.help")))
Chapter 49

)synonym help page Command

49.1 synonym help page man page

— synonym.help —

====================================================================
A.24.  )synonym
====================================================================

User Level Required: interpreter

Command Syntax:

- )synonym
- )synonym synonym fullCommand
- )what synonyms

Command Description:

This command is used to create short synonyms for system command expressions. For example, the following synonyms might simplify commands you often use.

)synonym save history )save
)synonym restore history )restore
)synonym mail system mail
)synonym ls system ls
)synonym fortran set output fortran

Once defined, synonyms can be used in place of the longer command expressions. Thus

)fortran on
is the same as the longer
)set fortran output on
To list all defined synonyms, issue either of
)synonyms
)what synonyms
To list, say, all synonyms that contain the substring ‘“ap”’, issue
)what synonyms ap
Also See:
  o )set
  o )what

\footnote{1 “set” (45.36 p 808) “what” (54.1 p 939)

\textbf{defun The }\texttt{)synonym} command

\begin{verbatim}
(defun |synonym| (&rest ignore)
  (declare (ignore ignore))
  (|synonymSpad2Cmd|))
\end{verbatim}

\textbf{defun The }\texttt{)synonym} command implementation

\begin{verbatim}
(defun |synonymSpad2Cmd|)
\end{verbatim}
(defun |synonymSpad2Cmd| ()
  (let (line pair)
    (declare (special |$CommandSynonymAlist|))
    (setq line (|getSystemCommandLine|))
    (if (string= line "")
      (|printSynonyms| nil)
      (progn
        (setq pair (|processSynonymLine| line))
        (if |$CommandSynonymAlist|
          (putalist |$CommandSynonymAlist| (car pair) (cdr pair))
          (setq |$CommandSynonymAlist| (cons pair nil))))
      (|terminateSystemCommand|)))

---

defun Return a sublist of applicable synonyms

The argument is a list of synonyms, and this returns a sublist of applicable synonyms at the
current user level. [string2id-n p79]
[selectOptionLC p479]
[commandsForUserLevel p448]
[$systemCommands p443]
[$UserLevel p807]

---

defun synonymsForUserLevel ---

(defun |synonymsForUserLevel| (arg)
  (let (cmd nl)
    (declare (special |$systemCommands| |$UserLevel|))
    (if (eq |$UserLevel| '|development|)
      arg
      (dolist (syn (reverse arg))
        (setq cmd (string2id-n (cdr syn) 1))
        (when (|selectOptionLC| cmd (|commandsForUserLevel| |$systemCommands|) nil)
          (push syn nl))))
    nl))

---

defun Get the system command from the input line

[strpos p1045]
[substring p72]
[$currentLine p72]
--- defun getSystemCommandLine ---

(defun getSystemCommandLine ()
  (let (p line)
    (declare (special $currentLine$))
    (setq p (strpos "$currentLine" 0 nil))
    (if p
      (setq line (substring "$currentLine" p nil))
      (setq line "$currentLine"))
    (string-left-trim '(\space) line)))

---

defun Remove system keyword

[dropLeadingBlanks p??]
[maxindex p??]

--- defun processSynonymLine,removeKeyFromLine ---

(defun processSynonymLine,removeKeyFromLine (line)
  (prog (mx)
    (return
      (seq
        line (dropLeadingBlanks line))
      (setq mx (maxindex line))
      (exit
        (do ((i 0 (1+ i)))
          ((> i mx) nil)
          (exit
            (if (char= (elt line i) \space)
              (exit
                (return
                  (do ((j (1+ i) (1+ j)))
                    ((> j mx) nil)
                    (seq
                      (exit
                        (if (char\= (elt line j) \space)
                          (exit
                            (return
                              (substring line j nil))))))))))))))

---
defun processSynonymLine

[processSynonymLine,removeKeyFromLine p834]

— defun processSynonymLine —

(defun |processSynonymLine| (line)
  (cons
    (string2id-n line 1)
    ([processSynonymLine,removeKeyFromLine line]))

|---|

This command is in the list of $noParseCommands 18.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 18.2
Chapter 50

)system help page Command

50.1 system help page man page

— system.help —

====================================================================
A.25. )system
====================================================================

User Level Required: interpreter

Command Syntax:

- )system cmdExpression

Command Description:

This command may be used to issue commands to the operating system while
remaining in AXIOM. The cmdExpression is passed to the operating system for
execution.

To get an operating system shell, issue, for example, )system sh. When you
enter the key combination, Ctrl-D (pressing and holding the Ctrl key and then
pressing the D key) the shell will terminate and you will return to AXIOM. We
do not recommend this way of creating a shell because Lisp may field some
interrupts instead of the shell. If possible, use a shell running in another
window.

If you execute programs that misbehave you may not be able to return to
AXIOM. If this happens, you may have no other choice than to restart AXIOM
and restore the environment via )history )restore, if possible.

837
Also See:
  o )boot
  o )fin
  o )lisp
  o )pquit
  o )quit

This command is in the list of $noParseCommands which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands.
Chapter 51

)tangle help page Command

51.1  tangle help page man page

--- tangle.help ---

User Level Required: interpreter

Command Syntax:
- )tangle [fileName]

Command Description:
This command is used to tangle pamphlet files.

)tangle matrix.input.pamphlet

will tangle the contents of the file matrix.input.pamphlet into matrix.input. The ‘’.input.pamphlet’’ is optional.

----

--- defun tangle ---

(defun |tangle| (arg)
(let ($InteractiveMode$ namestring dot1 dot2 outfile

839
(chunkname "*") (extension "input"))
(declare (special |$InteractiveMode| |$options|))
(setq |$InteractiveMode| t)
(setq namestring (symbol-name (car arg)))
(setq dot1 (position #\. namestring))
(if dot1
  (setq outfile
    (concatenate 'string (subseq namestring 0 dot1) "." extension))
  (setq outfile
    (concatenate 'string (subseq namestring 0) "." extension)))
(setq dot2 (position #\. namestring :from-end t))
(cond
  ((and (numberp dot1) (numberp dot2) (< dot1 dot2)))
   ((and (numberp dot1) (numberp dot2) (= dot1 dot2))
    (setq namestring (concatenate 'string namestring "pamphlet")))
   (t
    (setq namestring (concatenate 'string namestring ".input.pamphlet"))))
(if (probe-file namestring)
  (progn
    (tangle namestring chunkname outfile)
    (format t (concatenate 'string outfile " created from " namestring "\n")))
  (format t (concatenate 'string namestring " file not found\n"))))
Chapter 52

)trace help page Command

52.1 trace help page man page

— trace.help —

====================================================================
A.26. )trace
====================================================================

User Level Required: interpreter

Command Syntax:

- )trace
- )trace )off

- )trace function [options]
- )trace constructor [options]
- )trace domainOrPackage [options]

where options can be one or more of

- )after S-expression
- )before S-expression
- )break after
- )break before
- )cond S-expression
- )count
- )count n
- )depth n
- )local op1 [... opN]
- )nonquietly
842

CHAPTER 52. \texttt{\textbackslash trace} HELP PAGE COMMAND

\begin{itemize}
\item \texttt{\textbackslash nt}
\item \texttt{\textbackslash off}
\item \texttt{\textbackslash only \textit{listOfDataToDisplay}}
\item \texttt{\textbackslash ops}
\item \texttt{\textbackslash ops \textit{op1} [\ldots \textit{opN} ]}
\item \texttt{\textbackslash restore}
\item \texttt{\textbackslash stats}
\item \texttt{\textbackslash stats \textbackslash reset}
\item \texttt{\textbackslash timer}
\item \texttt{\textbackslash varbreak}
\item \texttt{\textbackslash varbreak \textit{var1} [\ldots \textit{varN} ]}
\item \texttt{\textbackslash vars}
\item \texttt{\textbackslash vars \textit{var1} [\ldots \textit{varN} ]}
\item \texttt{\textbackslash within \textit{executingFunction}}
\end{itemize}

\textbf{Command Description:}

This command is used to trace the execution of functions that make up the AXIOM system, functions defined by users, and functions from the system library. Almost all options are available for each type of function but exceptions will be noted below.

To list all functions, constructors, domains and packages that are traced, simply issue

\texttt{\textbackslash trace}

To untrace everything that is traced, issue

\texttt{\textbackslash trace \textbackslash off}

When a function is traced, the default system action is to display the arguments to the function and the return value when the function is exited. Note that if a function is left via an action such as a \texttt{THROW}, no return value will be displayed. Also, optimization of tail recursion may decrease the number of times a function is actually invoked and so may cause less trace information to be displayed. Other information can be displayed or collected when a function is traced and this is controlled by the various options. Most options will be of interest only to AXIOM system developers. If a domain or package is traced, the default action is to trace all functions exported.

Individual interpreter, lisp or boot functions can be traced by listing their names after \texttt{\textbackslash trace}. Any options that are present must follow the functions to be traced.

\texttt{\textbackslash trace \textit{f}}

traces the function \textit{f}. To untrace \textit{f}, issue
)trace f )off

Note that if a function name contains a special character, it will be necessary to escape the character with an underscore

)trace _/_D_,1

To trace all domains or packages that are or will be created from a particular constructor, give the constructor name or abbreviation after )trace.

)trace MATRIX
)trace List Integer

The first command traces all domains currently instantiated with Matrix. If additional domains are instantiated with this constructor (for example, if you have used Matrix(Integer) and Matrix(Float)), they will be automatically traced. The second command traces List(Integer). It is possible to trace individual functions in a domain or package. See the )ops option below.

The following are the general options for the )trace command.

)break after
causes a Lisp break loop to be entered after exiting the traced function.

)break before
causes a Lisp break loop to be entered before entering the traced function.

)break
is the same as )break before.

)count
causes the system to keep a count of the number of times the traced function is entered. The total can be displayed with )trace )stats and cleared with )trace )stats reset.

)count n
causes information about the traced function to be displayed for the first n executions. After the nth execution, the function is untraced.

)depth n
causes trace information to be shown for only n levels of recursion of the traced function. The command

)trace fib )depth 10

will cause the display of only 10 levels of trace information for the recursive execution of a user function fib.
>math
causes the function arguments and return value to be displayed in the
AXIOM monospace two-dimensional math format.

>nonquietly
causes the display of additional messages when a function is traced.

>nt
This suppresses all normal trace information. This option is useful if
the )count or )timer options are used and you are interested in the
statistics but not the function calling information.

>off
causes untracing of all or specific functions. Without an argument, all
functions, constructors, domains and packages are untraced. Otherwise,
the given functions and other objects are untraced. To immediately
retrace the untraced functions, issue )trace )restore.

>only listOfDataToDisplay
causes only specific trace information to be shown. The items are listed
by using the following abbreviations:

- a  display all arguments
- v  display return value
- 1  display first argument
- 2  display second argument
- 15 display the 15th argument, and so on

>restore
causes the last untraced functions to be retraced. If additional options
are present, they are added to those previously in effect.

>stats
causes the display of statistics collected by the use of the )count and
)timer options.

>stats reset
resets to 0 the statistics collected by the use of the )count and )timer
options.

>timer
causes the system to keep a count of execution times for the traced
function. The total can be displayed with )trace )stats and cleared with
)trace )stats reset.

>varbreak var1 [... varN]
causes a Lisp break loop to be entered after the assignment to any of the
listed variables in the traced function.

>vars
causes the display of the value of any variable after it is assigned in the traced function. Note that library code must have been compiled (see description of command )compile ) using the )vartrace option in order to support this option.

)vars var1 [... varN]
causes the display of the value of any of the specified variables after they are assigned in the traced function. Note that library code must have been compiled (see description of command )compile ) using the )vartrace option in order to support this option.

)within executingFunction
causes the display of trace information only if the traced function is called when the given executingFunction is running.

The following are the options for tracing constructors, domains and packages.

)local [op1 [... opN] ]
causes local functions of the constructor to be traced. Note that to untrace an individual local function, you must use the fully qualified internal name, using the escape character _ before the semicolon.

)trace FRAC )local
)trace FRAC_;cancelGcd )off

)ops op1 [... opN]
By default, all operations from a domain or package are traced when the domain or package is traced. This option allows you to specify that only particular operations should be traced. The command

)trace Integer )ops min max _+ _-

traces four operations from the domain Integer. Since + and - are special characters, it is necessary to escape them with an underscore.

Also See:
o )boot
o )lisp
o )ltrace

The trace global variables
This decides when to give trace and untrace messages.

---

1 “boot” (4.1 p 23) “lisp” (?? p ??) “ltrace” (39.1 p 632)
defvar $traceNoisely

— initvars —
(defvar |$traceNoisely| nil)

——

defvar $reportSpadtrace

This reports the traced functions
— initvars —
(defvar |$reportSpadtrace| nil)

——

defvar $optionAlist

— initvars —
(defvar |$optionAlist| nil)

——

defvar $tracedMapSignatures

— initvars —
(defvar |$tracedMapSignatures| nil)

——

defvar $traceOptionList

— initvars —
(defvar $traceOptionList
  '(|after| |before| |break| |cond| |count| |depth| |local| |mathprint|
    |nonquietly| |nt| |of| |only| |ops| |restore| |timer| |varbreak|
    |vars| |within|))

---

defun trace

[traceSpad2Cmd p847]

---

(defun trace (l)
  (traceSpad2Cmd l))

---

defun traceSpad2Cmd

[qcar p??]
[qcdr p??]
[getMapSubNames p869]
[trace1 p848]
[augmentTraceNames p872]
[traceReply p899]
[$mapSubNameAlist p??]

---

(defun traceSpad2Cmd (l)
  (let (tmp1 l1)
    (declare (special $mapSubNameAlist))
    (cond
     ((and (consp l)
           (eq (qcar l) '|Tuple|))
      (progn
       (setq tmp1 (qcdr l))
       (and (consp tmp1)
        (eq (qcdr tmp1) nil)
        (progn
         (setq 11 (qcar tmp1))
         t)))))
    (setq 1 11))
    (setq $mapSubNameAlist (|getMapSubNames| 1)))
defun trace1

[hasOption p451]
[throwKeyedMsg p??]
[unabbrev p??]
[isFunctor p??]
[getTraceOption p854]
[untraceDomainLocalOps p880]
[qlessp p1068]
[poundsign p??]
[untrace p862]
[centerAndHighlight p??]
[ptimers p859]
[say p??]
[pcounters p860]
[selectOptionLC p479]
[resetSpacers p858]
[resetTimers p858]
[resetCounters p858]
[qcar p??]
[qcdr p??]
[vecp p??]
[sayKeyedMsg p329]
[devaluate p??]
[lassoc p??]
[trace1 p848]
[delete p??]
[?t p902]
[seq p??]
[exit p??]
[transTraceItem p863]
[addassoc p??]
[getTraceOptions p852]
[/trace,0 p??]
[saveMapSig p853]
[$traceNoisely p846]
[$options p??]
[$lastUntraced p??]
[$optionAlist p846]
— defun trace1 —

(defun trace1 (arg)
  (prog (traceNoisely constructor ops lops templ opt a
        oldl newoptions domain tracelist optionlist domainlist
        oplist y varlist argument)
    (declare (special traceNoisely options lastUntraced
        optionList)))
    (return
      (seq
        (progn
          (setq traceNoisely nil)
          (cond
            ((hasOption options 'nonquietly)
              (setq traceNoisely t)))
            (cond
              ((hasOption options 'off)
               (cond
                ((or (setq ops (hasOption options 'ops))
                  (setq lops (hasOption options 'local)))
                (cond
                  ((null arg) (throwKeyedMsg 's2it0019 nil))
                  (t
                    (setq constructor
                      (unabbrev)
                      (cond
                        ((atom arg) arg)
                        (null (cdr arg))
                        (cond
                          ((atom (car arg)) (car arg))
                          (t (car (car arg))))))
                    (t nil)))
                (t nil)))
            (cond
              ((null (isFunctor constructor))
                (throwKeyedMsg 's2it0020 nil))
              (t
                (cond (ops (setq ops (getTraceOption ops)) nil))
                (cond
                  (lops
                    (setq lops (cdr (getTraceOption lops)))
                    (untraceDomainLocalOps))
                  (t nil)))))))
    (cond
      (and (qslessp 1 (#| options|))
        (null (hasOption options 'nonquietly)))
      (throwKeyedMsg 's2it0021 nil))
    (t (untrace arg))))
    (cond
      ((hasOption options 'stats)
       (cond
        (qslessp 1 (#| options|))
(throwKeyedMsg 's2it0001 (cons "trace " stats" nil)))
(t
(setq temp1 (car $options))
(setq opt (cdr temp1))
(cond
(null opt)
((centerAndHighlight) "Traced function execution times" 78 '-)
(pcounters)
(say " ")
((centerAndHighlight) "Traced function execution counts" 78 '-)
(pcounters))
(t
(selectOptionLC (car opt) '(reset) 'optionError)
(resetSpacers)
(resetTimers)
(resetCounters)
(throwKeyedMsg 's2it0002 nil))))
((setq a (hasOption $options) 'restore))
(unless (setq oldl $lastUntraced)
(setq newoptions (delete a $options))
(if (null arg)
(trace1 oldl)
(progn
(dolist (x arg)
(if (and (consp x)
(progn
 (setq domain (qcar x))
 (setq oplist (qcdr x))
 t)
 (vecp domain))
(sayKeyedMsg 's2it0003 (cons (devaluate domain) nil))
(progn
 (setq $options (append newoptions (assoc x $optionAlist)))
 (trace1 (list x)))))

((null arg) nil)
((and (consp arg) (eq (qcdr arg) nil) (eq (qcar arg) '?)) (?t))
(t
(setq tracelist
(or
 (prog (t1)
 (setq t1 nil)
 (return
 (do ((t2 arg (cdr t2)) (x nil))
 (or (atom t2)
 (prog (setq x (car t2)) nil))
 (reverse0 t1))
 (seq
 (exit
 (setq t1 (cons (transTraceItem x) t1)))))
 (return nil)))))
(do ((t3 tracelist (cdr t3)) (x nil))
    ((or (atom t3) (progn (setq x (car t3)) nil)) nil)
  (setq
    (exit
      (setq $optionAlist| (addassoc x $options| $optionAlist|)))
  (setq optionlist ($getTraceOptions| $options|))
  (setq argument
    (cond
      ((setq domainlist (lassoc 'of| optionlist|))
        (cond
          ((lassoc 'ops optionlist)
            (throwKeyedMsg| 's2it0004 nil))
          t
            (setq oplist
              (cond
                (tracelist (list (cons 'ops tracelist)))
                (t nil)))
          (setq varlist
            (cond
              ((setq y (lassoc 'vars| optionlist|)
                (list (cons 'vars| y)))
              (t nil)))
            (append domainlist (append oplist varlist))))
      (optionlist (append tracelist optionlist))
      (t tracelist))))
  (/TRACE,0|
    (prog (t4)
      (setq t4 nil)
      (return
        (do ((t5 argument (cdr t5)) (|funName| nil))
          ((or (atom t5)
              (progn (setq |funName| (car t5)) nil))
            (nreverse0 t4))
          (setq
            (exit
              (setq t4 (cons |funName| t4)))))
      (saveMapSig|)
      (prog (t6)
        (setq t6 nil)
        (return
          (do ((t7 argument (cdr t7)) (|funName| nil))
            ((or (atom t7)
                (progn (setq |funName| (car t7)) nil))
              (nreverse0 t6))
            (setq
              (exit
                (setq t6 (cons |funName| t6))))))))
defun getTraceOptions

[throwKeyedMsg p??]
[throwListOfKeyedMsgs p??]
[poundsign p??]
[seq p??]
[exit p??]
[getTraceOption p854]
[$traceErrorStack p??]

— defun getTraceOptions —

(defun |getTraceOptions| (|options|)
  (prog (|$traceErrorStack| optionlist temp1 key |parms|)
    (declare (special |$traceErrorStack|))
    (return
      (seq
        (progn
          (setq |$traceErrorStack| nil)
          (setq optionlist
            (prog (t0)
              (setq t0 nil)
              (return
                (do ((t1 |options| (cdr t1)) (x nil))
                  ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
                (seq
                  (exit
                    (setq t0 (cons (|getTraceOption| x) t0))))))))
          (cond
            (|$traceErrorStack|
              (cond
                ((null (cdr |$traceErrorStack|))
                  (setq temp1 (car |$traceErrorStack|))
                  (setq key (car temp1))
                  (setq |parms| (cadr temp1))
                  (|throwKeyedMsg| key (cons "" |parms|))
                (t
                  (|throwListOfKeyedMsgs| 's2it0017
                    (cons (|#| |$traceErrorStack|) nil)
                    (nreverse |$traceErrorStack|))))
            (t optionlist))))))

——
defun saveMapSig

(defun \textit{saveMapSig} (funnames)
  (let (map)
    (declare (special \$tracedMapSignatures \$mapSubNameAlist))
    (dolist (name funnames)
      (when (setq map (\textit{rassoc} name \$mapSubNameAlist))
        (setq \$tracedMapSignatures
          (addassoc name (\textit{getMapSig} map name) \$tracedMapSignatures))))

— defun saveMapSig —

defun getMapSig

(defun \textit{getMapSig} (mapname subname)
  (let (lmms sig)
    (declare (special \$InteractiveFrame))
    (when (setq lmms (\textit{get} mapname \text\'{\textit{localModemap}} \$InteractiveFrame))
      (do ((t0 lmms (cdr t0)) (mm nil) (t1 nil sig))
          ((or (atom t0) (progn (setq mm (car t0)) nil) t1 nil)
            (when (\textit{boot-equal} (cadr mm) subname) (setq sig (cdar mm))))
            sig)))

———

defun getTraceOption,hn

(defun \textit{getTraceOption},hn (seq)
  (exit)
  (isDomainOrPackage)

— defun getTraceOption,hn —


(defun getTraceOption (arg)
  (prog (1 opts key a |n|)
    (declare (special |$traceOptionList|))
    (return
      (setq opts (\(setq |getTraceOption,hn| x) (exit g))
    (exit
      (|stackTraceOptionError| (cons 's2it0013 (cons (cons x nil) nil)))))))

---

(defun getTraceOption

  (defun getTraceOption (arg)
    (prog (1 opts key a |n|)
      (declare (special |$traceOptionList|))
      (return

---

[stackTraceOptionError p861]
[domainToGenvar p861]

— defun getTraceOption,hn —

(defun getTraceOption,hn (x)
  (prog (g)
    (return
      (seq
        (if (and (atom x) (null (upper-case-p (elt (princ-to-string x) 0))))
          (exit
            (seq
              (if (|isDomainOrPackage| (eval x)) (exit x))
            (exit
              (|stackTraceOptionError|
                (cons 's2it0013 (cons (cons x nil) nil))))))))
        (if (setq g (|domainToGenvar| x)) (exit g))
      (exit
        (|stackTraceOptionError| (cons 's2it0013 (cons (cons x nil) nil))))))))

— defun getTraceOption —

(seq p??]
[exit p??]
[selectOptionLC p479]
[identp p1046]
[StackTraceOptionError p861]
[concat p1047]
[object2String p?]
[transOnlyOption p860]
[qcdr p?]
[qcar p?]
[getTraceOption,hn p853]
[isListOfIdentifiersOrStrings p869]
[isListOfIdentifiers p868]
[throwKeyedMsg p??]
[$traceOptionList p846]
(setq key (car arg))
(setq 1 (cdr arg))
(setq key
    (|selectOptionLC| key |$traceOptionList| '|traceOptionError|))
(setq arg (cons key 1))
(cond
    ((member key '(|nonquietly| |timer| |nt|)) arg)
        ((eq key '|break|)
            (cond
                ((null 1) (cons '|break| (cons '|before| nil)))
                t
                (setq |opts|
                    (prog (t0)
                        (setq t0 nil)
                        (return
                            (do ((t1 l (cdr t1)) (y nil))
                                ((or (atom t1)
                                    (progn (setq y (car t1)) nil))
                                (nreverse0 t0)))
                        (seq
                            (exit
                                (setq t0
                                    (cons
                                        (|selectOptionLC| y '(|before| |after|) nil) t0))))))))
                ((eq key '|restore|)
                    (cond
                        ((null l) arg)
                        (t
                            (|stackTraceOptionError| (cons 's2it0009 (cons nil nil))))))))
        ((eq key '|only|) (cons '|only| (|transOnlyOption| l))
        ((eq key '|within|)
            (cond
                ((and (consp 1))
                    (eq (qcdr 1) nil))
                (t
                    (null 1))
                t))
            (do ((t3 nil (null t2)) (t4 |opts| (cdr t4)) (y nil))
                                ((or t3 (atom t4) (progn (setq y (car t4)) nil))
                                t2)
                        (seq
                            (exit
                                (setq t2 (and t2 (identp y))))))))
                (cons '|break| |opts|))
          (t
            (|stackTraceOptionError| (cons 's2it0008 (cons nil nil))))))))
)
(progn (setq a (qcar l)) t)
(identp a))
(arg)
(t
(stackTraceOptionError|
(cons 's2it0010 (cons (cons "")within" nil) nil))))))
((member key '([cond] [before] [after]))
(setq key
(cond
((eq key 'cond) 'when)
(t key)))
(cond
((and (consp l)
  (eq (qcdr l) nil)
  (progn (setq a (qcar l)) t))
  (cons key 1))
(t
(stackTraceOptionError|
(cons 's2it0011
  (cons
    (cons (concat "")
      (object2String| key)) nil) nil))))))
((eq key 'depth)
(cond
((and (consp l)
  (eq (qcdr l) nil)
  (progn (setq n (qcar l)) t)
  (integerp n))
  arg)
(t
(stackTraceOptionError|
(cons 's2it0012 (cons (cons "")depth" nil) nil))))))
((eq key 'count)
(cond
((or (null l)
  (and (consp l)
    (eq (qcdr l) nil)
    (progn (setq n (qcar l)) t)
    (integerp n)))
  arg)
(t
(stackTraceOptionError|
(cons 's2it0012 (cons (cons "")count" nil) nil))))))
((eq key 'of)
(cons 'of
(prog (t5)
(setq t5 nil)
(return
(do ((t6 l (cdr t6)) (y nil))
  ((or (atom t6) (progn (setq y (car t6)) nil)) (nreverse0 t5))
  (setq t6 (cons y t6))))))
(setq t5 (cons (list (getTraceOption 'hn) y) t5)))))))
((member key '(local ops vars))
 (cond
 (or (null 1)
  (and (consp 1) (eq (qcdr 1) nil) (eq (qcar 1) '|all|))
  (cons key '|all|)
  ((isListOfIdentifiersOrStrings 1) arg)
  (t
   (stackTraceOptionError)
   (cons 's2it0015
     (cons
      (cons (concat "") (object2String key)) nil) nil)))))))))
((eq key '|varbreak|)
 (cond
 (or (null 1)
  (and (consp 1) (eq (qcdr 1) nil) (eq (qcar 1) '|all|))
  (cons '|varbreak| '|all|)
  ((isListOfIdentifiers 1) arg)
  (t
   (stackTraceOptionError)
   (cons 's2it0016
     (cons
      (cons (concat "") (object2String key)) nil) nil)))))))))
((eq key '|mathprint|)
 (cond
  ((null 1) arg)
  (t
   (stackTraceOptionError)
   (cons 's2it0009
     (cons
      (cons (concat "") (object2String key)) nil) nil)))))
(key (throwKeyedMsg 's2it0005 (cons key nil)))))))

---

defun traceOptionError

[stackTraceOptionError p861]
[commandAmbiguityError p452]

— defun traceOptionError —

(defun traceOptionError (opt keys)
 (if (null keys)
   (stackTraceOptionError)
   (cons 's2it0007 (cons (cons opt nil) nil)))))
CHAPTER 52. )TRACE HELP PAGE COMMAND

((commandAmbiguityError '([trace option] opt keys)))

---

defun resetTimers
[concat p1047]
[/timerlist p??]

--- defun resetTimers ---
(defun |resetTimers| ()
(declare (special /timerlist))
(dolist (timer /timerlist)
  (set (intern (concat timer ",TIMER")) 0)))

---

defun resetSpacers
[concat p1047]
[/spacelist p??]

--- defun resetSpacers ---
(defun |resetSpacers| ()
(declare (special /spacelist))
(dolist (spacer /spacelist)
  (set (intern (concat spacer ",SPACE")) 0)))

---

defun resetCounters
[concat p1047]
[/countlist p??]

--- defun resetCounters ---
(defun |resetCounters| ()
(declare (special /countlist))
(dolist (k /countlist)
  (set (intern (concat k ",COUNT")) 0)))
defun ptimers

[sayBrightly p??]
[bright p??]
[quotient p??]
[concat p1047]
[float p??]
[/timerlist p??]

---

(defun |ptimers| ()
  (declare (special /timerlist |$timerTicksPerSecond|))
  (if (null /timerlist)
    (|sayBrightly| " no functions are timed")
    (dolist (timer /timerlist)
      (|sayBrightly| " " ,@(|bright| timer) |:| " 
        ,,(quotient (eval (intern (concat timer ",TIMER")))
          ,(|float| |$timerTicksPerSecond|)) " sec."))))

---

defun pspacers

[sayBrightly p??]
[bright p??]
[concat p1047]
[/spacelist p??]

---

(defun |pspacers| ()
  (declare (special /spacelist))
  (if (null /spacelist)
    (|sayBrightly| " no functions have space monitored")
    (dolist (spacer /spacelist)
      (|sayBrightly| " " ,@(|bright| spacer) |:| 
        ,(eval (intern (concat spacer ",SPACE"))) " bytes"))))
defun pcounters

[sayBrightly p??]
[bright p??]
[concat p1047]
[/countlist p??]

— defun pcounters —

(defun |pcounters| ()
 (declare (special /countlist))
 (if (null /countlist)
     (|sayBrightly| " no functions are being counted")
     (dolist (k /countlist)
         (|sayBrightly| ((" " ,(|bright| k) |:| " " ,(eval (intern (concat k ",COUNT"))) " times")))))

—

defun transOnlyOption

[transOnlyOption p860]
[upcase p??]
[stackTraceOptionError p861]
[qcar p??]
[qcdr p??]

— defun transOnlyOption —

(defun |transOnlyOption| (arg)
 (let (y n)
   (when (and (consp arg) (progn (setq n (qcar arg)) (setq y (qcdr arg)) t))
     (cond
       ((integerp n) (cons n (|transOnlyOption| y)))
       ((member (setq n (upcase n)) '(v a c)) (cons n (|transOnlyOption| y)))
       (t
         (|StackTraceOptionError| (cons 's2it0006 (list (list n)))
         (|transOnlyOption| y))))))

——
defun stackTraceOptionError

[\$traceErrorStack\ p??]

— defun stackTraceOptionError —

(defun stackTraceOptionError (x)
  (declare (special \$traceErrorStack))
  (push x \$traceErrorStack)
  nil)

——

defun removeOption

— defun removeOption —

(defun removeOption (op options)
  (let (opt t0)
    (do ((t1 options (cdr t1)) (optentry nil)
         (or (atom t1)
             (progn (setq optentry (car t1)) nil)
             (progn (progn (setq opt (car optentry)) optentry) nil))
        (nreverse0 t0))
    (when (not (equal opt op)) (setq t0 (cons optentry t0))))))

——

defun domainToGenvar

[unabbrevAndLoad\ p??]
[getdatabase\ p1010]
[opOf\ p??]
genDomainTraceName\ p862]
evalDomain\ p913]
[\$doNotAddEmptyModeIfTrue\ p??]

— defun domainToGenvar —

(defun domainToGenvar (arg)
  (let ([\$doNotAddEmptyModeIfTrue\ y g]
        (declare (special \$doNotAddEmptyModeIfTrue)))
    (setq \$doNotAddEmptyModeIfTrue\ t))
(when
  (and (setq y (unabbrevAndLoad arg))
    (eq (getdatabase (opOf y) 'constructorKind) 'domain))
  (setq g (genDomainTraceName y))
  (set g (evalDomain y))
  g)))

---

defun genDomainTraceName

[lassoc p]
[genvar p]
[$domainTraceNameAssoc p]

---

defun genDomainTraceName

(defun |genDomainTraceName| (y)
  (let (u g)
    (declare (special |$domainTraceNameAssoc|))
    (if (setq u (lassoc y |$domainTraceNameAssoc|))
      u
      (progn
        (setq g (genvar))
        (setq |$domainTraceNameAssoc| (cons (cons y g) |$domainTraceNameAssoc|))
        g)))))

---

defun untrace

[copy p]
[transTraceItem p863]
[/untrace.0 p]
[lassocSub p871]
[removeTracedMapSigs p864]
[$lastUntraced p]
[$mapSubNameAlist p]
[/tracenames p]

---

defun untrace

(defun |untrace| (arg)
  (let (untraceList)
(declare (special |$lastUntraced| /tracenames |$mapSubNameAlist|))
(if arg
  (setq |$lastUntraced| arg)
  (setq |$lastUntraced| (copy /tracenames)))
(setq untracelist
  (do ((t1 arg (cdr t1)) (x nil) (t0 nil))
      ((or (atom t1) (progn (setq x (car t1)) nil))
       (nreverse0 t0))
    (push (|transTraceItem| x) t0)))
(\UNTRACE,0|
  (do ((t3 untracelist (cdr t3)) (|funName| nil) (t2 nil))
      ((or (atom t3) (progn (setq |funName| (car t3)) nil))
       (nreverse0 t2))
    (push (|lassocSub| |funName| |$mapSubNameAlist|) t2)))
(|removeTracedMapSigs| untracelist))

---

defun transTraceItem

[get p??]
  [member p1048]
  [objMode p??]
  [objVal p??]
  [domainToGenvar p861]
  [unabbrev p??]
  [constructor? p??]
  [vecp p??]
  [transTraceItem p863]
  [devaluate p??]
  [throwKeyedMsg p??]
  [$doNotAddEmptyModeIfTrue p??]

--- defun transTraceItem ---

(defun |transTraceItem| (x)
  (prog (|$doNotAddEmptyModeIfTrue| |value| y)
    (declare (special |$doNotAddEmptyModeIfTrue|))
    (return
      (progn
        (setq |$doNotAddEmptyModeIfTrue| t)
        (cond
          ((atom x)
            (cond
              ((and (setq |value| (|get| x '#|value| |$InteractiveFrame|))
                (|member| (|objModel| |value|)
                  '#(|Model|) '#(|Domain|) '#(SubDomain (|Domain|)))))))
```
(setq x (|objVal| |value|))
(cond
  ((setq y (|domainToGenvar| x)) y)
   (t x)))
((upper-case-p (elt (princ-to-string x) 0))
  (setq y (|unabbrev| x))
  (cond
    ((|constructor?| y) y)
    ((and (consp y) (|constructor?| (car y))) (car y))
    ((setq y (|domainToGenvar| x)) y)
    (t x)))
  (t x)))
((vecp (car x)) (|transTraceItem| (|devaluate| (car x))))
((setq y (|domainToGenvar| x)) y)
(t (|throwKeyedMsg| 's2it0018 (cons x nil)))))))

defun removeTracedMapSigs

[$tracedMapSignatures p846]

— defun removeTracedMapSigs —

(defun |removeTracedMapSigs| (untraceList)
  (declare (special |$tracedMapSignatures|))
  (dolist (name untraceList)
    (remprop name |$tracedMapSignatures|)))

— defun coerceTraceArgs2E —

defun coerceTraceArgs2E

[spadsysnamep p??]
[|name| p1045]
[coerceSpadArgs2E p865]
[|objValUnwrap| p??]
[coerceInteractive p??]
[|objNewWrap| p??]
[$OutputForm| p??]
[$mathTraceList| p??]
[$tracedMapSignatures p846]

— defun coerceTraceArgs2E —
(defun coerceTraceArgs2E (tracename subname args)
  (declare (ignore tracename))
  (let (name)
    (declare (special $OutputForm $mathTraceList $tracedMapSignatures))
    (cond
      ((member (setq name subname) $mathTraceList)
       (if (spadsysnamep (pname name))
           (coerceSpadArgs2E (reverse (cdr (reverse args))))
           (do ((t1 '(arg1 arg2 arg3 arg4 arg5 arg6 arg7 arg8
                        arg9 arg10 arg11 arg12 arg13 arg14 arg15
                        arg16 arg17 arg18 arg19) (cdr t1))
               (name nil)
               (t2 args (cdr t2))
               (arg nil)
               (t3 (cdr (lassoc subname $tracedMapSignatures)) (cdr t3))
               (type nil)
               (t0 nil))
            ((or (atom t1)
                 (progn (setq name (car t1)) nil)
                 (atom t2)
                 (progn (setq arg (car t2)) nil)
                 (atom t3)
                 (progn (setq type (car t3)) nil))
             (nreverse0 t0))
        (setq t0
          (cons
            (list '=' name
              (objValUnwrap
               (coerceInteractive
                (objNewWrap arg type) $OutputForm)))
             (((spadsysnamep (pname name)) (reverse (cdr (reverse args)))
               (t args)))))))

---

defun coerceSpadArgs2E

[seq p??]
[exit p??]
[objValUnwrap p??]
[coerceInteractive p??]
[objNewWrap p??]
[$streamCount p801]
[$OutputForm p??]
[$tracedSpadModemap p??]

— defun coerceSpadArgs2E —
(defun coerceSpadArgs2E (args)
  (let ((|$streamCount| 0))
    (declare (special |$streamCount| |$OutputForm| |$tracedSpadModemap|))
    (do ((t1 '(|arg1| |arg2| |arg3| |arg4| |arg5| |arg6| |arg7| |arg8|
          |arg9| |arg10| |arg11| |arg12| |arg13| |arg14| |arg15|
          |arg16| |arg17| |arg18| |arg19|) (cdr t1))
      (name nil)
      (t2 args (cdr t2))
      (arg nil)
      (t3 (cdr |$tracedSpadModemap|) (cdr t3))
      (type nil)
      (t0 nil))
    ((or (atom t1)
          (progn (setq name (car t1)) nil)
          (atom t2)
          (progn (setq arg (car t2)) nil)
          (atom t3)
          (progn (setq type (car t3)) nil))
     (nreverse0 t0)))
  (seq
    (exit
     (setq t0
       (cons
        (cons '='
          (cons name
            (cons ((|objValUnwrap|
               (|coerceInteractive|
              (|objNewWrap| arg type)
            |$OutputForm|))) nil)))))))))

---

defun subTypes

[|assoc p??|
 |seq p??|
 |exit p??|
 |subTypes p866|

— defun subTypes —

(defun subTypes (|mm| |sublist|)
  (prog (s)
    (return
     (seq
      (cond
```lisp
(defun coerceTraceFunValue2E
  (tracename
   subname
   value
  )
  (let (name u)
    (declare (special $tracedMapSignatures $OutputForm $mathTraceList))
    (if (member (setq name subname) $mathTraceList))
      (cond
        ((spadsysnamep (pname tracename)) (coerceSpadFunValue2E value))
        ((setq u (lassoc subname $tracedMapSignatures))
          (objValUnwrap)
          (coerceInteractive (objNewWrap value (car u) $OutputForm))
          (t value))
      value)))
```
defun coerceSpadFunValue2E

(defun coerceSpadFunValue2E (value)
  (let ((streamCount)
        (declare (special streamCount tracedSpadModemap OutputForm))
        (setq streamCount 0)
        (objValUnwrap
         (coerceInteractive
          (objNewWrap value (car tracedSpadModemap))
          OutputForm)))))

defun isListOfIdentifiers

(defun isListOfIdentifiers (arg)
  (prog ()
    (return
     (seq
      (prog (t0)
        (setq t0 t)
        (return
         (do ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
              ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
              (seq
               (exit
                (setq t0 (and t0 (identp x))))))))))))
defun isListOfIdentifiersOrStrings

(setq t0 t)
(setq t0 (and t0 (or (identp x) (stringp x)))))))))))

defun getMapSubNames

(let (lmm subs)
(declare (special /tracenames $lastUntraced $InteractiveFrame))
(setq subs nil)
(dolist (mapname arg)
  (when (setq lmm (get mapname 'localModemap $InteractiveFrame))
    (setq subs
      (append
       (do ((t2 lmm (cdr t2)) (t1 nil) (mm nil))
        ((or (atom t2)
          (progn (setq mm (CAR t2)) nil))
         (progn (setq mm (CAR t2)) nil)) (nreverse0 t1)))

/do  ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
  ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
  (seq
    (exit
      (setq t0 (and t0 (or (identp x) (stringp x)))))))))))

(unionq $lastUntraced $InteractiveFrame)
(union $lastUntraced $InteractiveFrame))
(get PreviousMapSubNames p870]
(unionp $lastUntraced p??]
[$lastUntraced p??]
[$InteractiveFrame p??]
[/tracenames p??]

---

(label_frame)
(setq t1 (cons (cons mapname (cadr |mm|)) t1))
(|union| subs))
(|getPreviousMapSubNames| (unionq /tracenames |$lastUntraced|))))

---

defun getPreviousMapSubNames

[get p??]
[exit p??]
[seq p??]

— defun getPreviousMapSubNames —

(defun getPreviousMapSubNames (traceNames)
  (prog (lmm subs)
    (return
     (seq
      (progn
        (setq subs nil)
        (seq
         (do ((t0 (assocleft (caar |$InteractiveFrame|)) (cdr t0))
          (mapname nil))
          ((or (atom t0) (progn (setq mapname (car t0)) nil)) nil)
        (seq
         (exit
          (cond
           ((setq lmm
            (|get| mapname '|localModemap| |$InteractiveFrame|))
             (exit
              (cond
               ((member (cadar lmm) |traceNames|)
                 (exit
                  (do ((t1 lmm (cdr t1)) (|mm| nil))
                    ((or (atom t1) (progn (setq |mm| (car t1)) nil)) nil)
                  (seq
                   (exit
                    (setq subs
                      (cons (cons mapname (cadr |mm|)) subs)))))
                  (exit subs)))))))))))))

---
defun lassocSub

(defun |lassocSub| (x subs)
  (let (y)
    (if (setq y (lassq x subs))
      y
      x)))

---

defun rassocSub

(defun |rassocSub| (x subs)
  (let (y)
    (if (setq y (|rassoc| x subs))
      y
      x)))

---

defun isUncompiledMap

(defun |isUncompiledMap| (x)
  (let (y)
    (declare (special |$InteractiveFrame|))
    (when (setq y (|get| x '|value| |$InteractiveFrame|))
      (and
       (eq (caar y) 'map)
       (null (|get| x '|localModemap| |$InteractiveFrame|))))))

---

52.1. TRACE HELP PAGE MAN PAGE
defun isInterpOnlyMap

[get p??]
[$InteractiveFrame p??]

— defun isInterpOnlyMap —

(defun |isInterpOnlyMap| (map)
  (let (x)
    (declare (special |$InteractiveFrame|))
    (when (setq x (|get| map '|localModemap| |$InteractiveFrame|))
      (eq (caaar x) '|interpOnly|)))))

defun augmentTraceNames

[get p??]
[$InteractiveFrame p??]

— defun augmentTraceNames —

(defun |augmentTraceNames| (arg)
  (let (mml res)
    (declare (special |$InteractiveFrame|))
    (dolist (tracename arg)
      (if (setq mml (|get| tracename '|localModemap| |$InteractiveFrame|))
        (setq res
          (append
            (prog (t1)
              (setq t1 nil)
              (return
                (do ((t2 mml (cdr t2)) (|mm| nil))
                  ((or (atom t2)
                      (progn (setq |mm| (CAR t2)) nil))
                    (nreverse0 t1))
                    (setq t1 (cons (cadr |mm|) t1))))
                    res))
                    (setq res (cons tracename res))))
        res))

—
defun isSubForRedundantMapName

[assocSub p871]
[member p1048]
[assocleft p??]
[$mapSubNameAlist p??]

— defun isSubForRedundantMapName —

(defun isSubForRedundantMapName (subname)
  (let (mapname tail)
    (declare (special $mapSubNameAlist))
    (when (setq mapname (rassocSub subname $mapSubNameAlist))
      (when (setq tail (member (cons mapname subname) $mapSubNameAlist))
        (member mapname (cdr (assocleft tail)))))))

———

defun untraceMapSubNames

[assocright p??]
[/untrace,2 p??]
[setdifference p??]
[getPreviousMapSubNames p870]
[$mapSubNameAlist p??]
[$lastUntraced p??]

— defun untraceMapSubNames —

(defun untraceMapSubNames (traceNames)
  (let ($mapSubNameAlist subs)
    (declare (special $mapSubNameAlist $lastUntraced))
    (if (null (setq $mapSubNameAlist (getPreviousMapSubNames |traceNames|))$lastUntraced))
      nil
    (dolist (name (setq subs (assocright $mapSubNameAlist))
      (when (member name /traceNames)
        (/UNTRACE,2 name nil)
      (setq $lastUntraced (setdifference $lastUntraced subs)))))))

———
defun funfind,LAM

&qcar p??]
[SEQ p??]
[isFunctor p??]
[exit p??]

— defun funfind,LAM —

(defun |funfind,LAM| (functor opname)
  (prog (ops tmp1)
    (return
     (seq
      (progn
       (setq ops (|isFunctor| functor))
       (prog t0)
       (setq t0 nil)
       (return
        (do ((t1 ops (cdr t1)) (u nil))
          ((or (atom t1) (progn (setq u (car t1)) nil)) (nreverse0 t0))
          (seq
           (exit
            (cond
             ((and (consp u)
               (progn
                (setq tmp1 (qcar u))
                (and (consp tmp1) (equal (qcar tmp1) opname))))
                (setq t0 (cons u t0))))))))))))

defmacro funfind

— defmacro funfind —

(defmacro |funfind| (&whole t0 &rest notused &aux t1)
  (declare (ignore notused))
  (dsetq t1 t0)
  (cons '|funfind,LAM| (wrap (cdr t1) '(quote quote))))
defun isDomainOrPackage

[refvecp p??]
[poundsign p??]
[isFunctor p??]
[opOf p??]

— defun isDomainOrPackage —

(defun isDomainOrPackage (dom)
  (and
    (refvecp dom)
    (> (|#| dom) 0)
    (isFunctor (opOf (elt dom 0))))

defun isTraceGensym

[gensymp p??]

— defun isTraceGensym —

(defun isTraceGensym (x)
  (gensym x))

defun spadTrace,g

— defun spadTrace,g —

(defun spadTrace,g (x)
  (if (stringp x) (intern x) x))

defun spadTrace,isTraceable

[seq p??]
[exit p??]
defun spadTrace, isTraceable

(defun spadTrace, isTraceable (x |domain|)
  (prog (n |functionSlot|)
    (return
     (seq
     (progn
      (setq n (caddr x))
      x
      (seq
       (if (atom (elt |domain| n)) (exit nil))
       (setq |functionSlot| (car (elt |domain| n)))
       (if (gensymp |functionSlot|)
        (exit (seq (|reportSpadTrace| '|Already Traced| x) (exit nil))))
       (if (null (bpiname |functionSlot|))
        (exit
         (seq
          (|reportSpadTrace| '|No function for| x)
          (exit nil))))
        (exit t))))))

defun spadTrace

[refvecp p??]
[aldorTrace p??]
[isDomainOrPackage p875]
[userError p??]
[seq p??]
[exit p??]
[spadTrace,g p875]
[getOption p892]
[removeOption p861]
[opOf p??]
[assoc p??]
[kdr p??]
[flattenOperationAlist p883]
[getOperationAlistFromLisplib p??]
[spadTrace,isTraceable p875]
[as-insert p??]
[bpiname p??]
(defun spadTrace (domain options)
  (let ((|$tracedModemap| listofoperations listofvariables
    listofbreakvars anyiftrue domainid currententry
    currentalist opstructurelist sig kind triple fn op
    mm n alias tracename sigslotnumberalist)
    (declare (special |$tracedModemap| /tracenames |$fromSpadTrace| |$letAssoc|
      |$reportSpadtrace| |$traceNoisely|)))
  (setq |$fromSpadTrace| t)
  (setq |$tracedModemap| nil)
  (cond
    ((and (consp domain)
      (refvecp (car domain))
      (eql (elt (car domain) 0) 0))
      (|aldorTrace| domain options))
    ((null (|isDomainOrPackage| domain))
      (|userError| "bad argument to trace"))
    (t
      (setq listofoperations
        (prog (t0)
          (setq t0 nil)
          (return
           (do ((t1 (|getOption| 'ops options) (cdr t1)) (x nil))
            ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
             (seq
               (exit
                (setq t0 (cons (|spadTrace,g| x) t0)))))))
    (cond
      ((setq listofvariables (|getOption| 'vars options))
        (setq options (|removeOption| 'vars options)))
      (cond
        ((setq options (|removeOption| 'vars options)))
        (cond
          ...)
((setq listofbreakvars (getOption 'varbreak options))
  (setq options (removeOption 'varbreak options)))
(setq anyiftrue (null listofoperations))
(setq domainid (opOf (elt domain 0)))
(setq currententry (assoc domain /tracenames))
(setq currentalist (kdr currententry))
(setq opstructurelist
  (flattenOperationAlist (getOperationAlistFromLisplib domainid)))
(setq sigslotnumberalist
  (prog (t2)
    (setq t2 nil)
    (return
     (do ((t3 opstructurelist (cdr t3)) (t4 nil))
       ((or (atom t3)
            (progn (setq t4 (CAR t3)) nil)
            (progn
              (setq op (car t4))
              (setq sig (cadr t4))
              (setq n (caddr t4))
              (setq kind (car (cddddr t4)))
              nil)
           (nreverse0 t2))
     (seq
      (exit
       (cond
        ((and (eq kind 'elt)
            (or anyiftrue (member op listofoperations))
            (integerp n)
            (spadTrace,isTraceable)
            (setq triple
              (cons op (cons sig (cons n nil)))) domain))
        (setq t2 (cons triple t2)))))))))
(cond
  (listofvariables
   (do ((t5 sigslotnumberalist (cdr t5)) (t6 nil))
      ((or (atom t5)
       (progn (setq t6 (car t5)) nil)
       (progn (progn
           (setq fn (car (elt domain n)))
           (setq $letAssoc (as-insert (bpiname fn) listofvariables $letAssoc)))))))))))
(cond
  (listofbreakvars
   (do ((t7 sigslotnumberalist (cdr t7)) (t8 nil))
      ((or (atom t7)
\begin{verbatim}
(progn (setq t8 (car t7)) nil)
(progn (progn (setq n (caddr t8)) t8) nil))

(seq
  (exit
    (progn
      (setq fn (car (elt domain n)))
      (setq |$letAssoc|
        (as-insert (bpiname fn)
          (cons (cons 'break listofbreakvars) nil) |$letAssoc|)))))

(do ((t9 sigslotnumberalist (cdr t9)) (|pair| nil))
  ((or (atom t9)
    (progn (setq |pair| (car t9)) nil))
   (progn
     (setq op (car |pair|))
     (setq mm (cadr |pair|))
     (setq n (caddr |pair|))
     (setq alias (|spadTraceAlias| domainid op n))
     (setq |$tracedModemap|
       (|subTypes| mm (|constructSubst| (elt domain 0))))
     (setq tracename
       (bpitrace (car (elt domain n)) alias options))
     (nconc |pair|)
     (cons listofvariables
       (cons (car (elt domain n))
         (cons tracename (cons alias nil)))))
   nil)
   nil)

(seq
  (exit
    (progn
      (setq alias (|spadTraceAlias| domainid op n))
      (setq |$tracedModemap|
        (|subTypes| mm (|constructSubst| (elt domain 0))))
      (setq tracename
        (bpitrace (car (elt domain n)) alias options))
      (nconc |pair|)
      (cons listofvariables
        (cons (car (elt domain n))
          (cons tracename (cons alias nil)))))
    (rplac (car (elt domain n)) tracename))))

(setq sigslotnumberalist
  (prog (t10)
    (setq t10 nil)
    (return
      (do ((t11 sigslotnumberalist (cdr t11)) (x nil))
      ((or (atom t11) (progn (setq x (car t11)) nil)) (nreverse0 t10))
        (seq
          (exit
            (cond ((cdddr x) (setq t10 (cons x t10))))))))))

(cond
  ($reportSpadtrace|
    (cond ($traceNoisely| (|printDashedLine|))
      (do ((t12 (orderBySlotNumber| sigslotnumberalist) (cdr t12))
        (x nil))
        ((or (atom t12)
          (progn (setq x (car t12)) nil))
          (return)
          (setq x (car (elt domain n))
            (setq tracename (|spadTraceAlias| domainid op n))
            (setq |$tracedModemap|
              (|subTypes| mm (|constructSubst| (elt domain 0)))))
            (setq Alias
              (bpitrace (car (elt domain n)) alias options))
            (nconc |pair|
              (cons listofvariables
                (cons (car (elt domain n))
                  (cons tracename (cons alias nil))))))
          (rplac (car (elt domain n)) tracename)))))

(cond
  nil)
\end{verbatim}
(seq (exit (reportSpadTrace 'tracing x))))))
(cond (|$letAssoc| (setq printflag t))
(cond
currententry
(rplac (cdr currententry)
  (append sigslotnumberalist currentalist))
(t
(setq /tracenames
  (cons (cons domain sigslotnumberalist) /tracenames))
  (spadReply))))))))

---

defun traceDomainLocalOps

[sayMSG p331]

— defun traceDomainLocalOps —

(defun |traceDomainLocalOps| ()
  (|sayMSG| '(" The )local option has been withdrawn")(|sayMSG| '(" Use )ltr to trace local functions.")))

---

defun untraceDomainLocalOps

[sayMSG p331]

— defun untraceDomainLocalOps —

(defun |untraceDomainLocalOps| ()
  (|sayMSG| '(" The )local option has been withdrawn")(|sayMSG| '(" Use )ltr to trace local functions.")))

---

defun traceDomainConstructor

[getOption p892]
[seq p??]
[exit p??]
— defun traceDomainConstructor —

(defun traceDomainConstructor (domainConstructor options)
  (prog (listOfLocalOps argl domain innerDomainConstructor)
    (declare (special $ConstructorCache))
    (return
      (seq
        (progn
          (loadFunctor domainConstructor)
          (setq listOfLocalOps (getOption 'local options))
          (when listOfLocalOps (traceDomainLocalOps))
          (cond
            ((and listOfLocalOps (null (getOption 'ops options))) nil)
            (t
             (do ((t2 (hget $ConstructorCache domainConstructor) (cdr t2))
                  (t3 nil))
                ((or (atom t2)
                     (progn (setq t3 (car t2)) nil)
                     (progn
                       (progn
                         (setq argl (car t3))
                         (setq domain (cddr t3)) t3)
                         nil))
                  nil))
            (seq
              (exit
                (spadTrace domain options)))
            (setq /tracenames (cons domainConstructor /tracenames))
            (setq innerDomainConstructor
              (intern (concat domainConstructor ";")))
            (cond
              ((fboundp innerDomainConstructor)
               (setq domainConstructor innerDomainConstructor)))
            (embed domainConstructor
              (cons 'lambda
                (cons
                  (cons '&rest
                    (cons 'args nil))
                  (cons
                    (cons 'prog
                      (cons
                        (cons 'call
                          (cons
                            (cons 'domain
                              (cons 'constructor
                                (cons 'black)
                                (cons 'domainConstructor)
                                (cons 'black)))))
                            (cons 'call
                              (cons 'domain
                                (cons 'constructor
                                  (cons 'black)
                                  (cons 'domainConstructor)
                                  (cons 'black)))))
                            (cons 'call
                              (cons 'domain
                                (cons 'constructor
                                  (cons 'black)
                                  (cons 'domainConstructor)
                                  (cons 'black))))))
              (setq domainConstructor 'black)))
            (seq
              (exit
                (spadTrace domain options)))))))
defun untraceDomainConstructor, keepTraced?

[seq p??]
[qcar p??]
isDomainOrPackage p875]
[boot-equal p??]
kar p??]
devaluate p??]
[exit p??]
[/untrace,0 p??]

— defun untraceDomainConstructor, keepTraced? —

(defun |untraceDomainConstructor, keepTraced?| (df domainConstructor) (prog (dc) (return (seq (if (and (and (cons df) (progn (setq dc (qcar df)) t)) (isDomainOrPackage dc) (boot-equal (kar (devaluate dc)) domainConstructor)) (exit (seq (|/UNTRACE,0| (cons dc nil)) (exit nil)))) (exit t)))))))
defun untraceDomainConstructor

(defun untraceDomainConstructor (domainConstructor)
  (prog (innerDomainConstructor)
    (declare (special /tracenames))
    (return
      (seq
        (progn
          (setq /tracenames
            (prog (t0)
              (setq t0 nil)
              (return
                (do ((t1 /tracenames (cdr t1)) (df nil))
                    ((or (atom t1) (progn (setq df (car t1)) nil)) (nreverse0 t0))
                  (seq
                    (exit
                      (cond (|(untraceDomainConstructor,keepTraced?|)
                            df domainConstructor)
                            (setq t0 (cons df t0)))))
            (setq innerDomainConstructor
              (intern (concat domainConstructor ";")))))
        (cond
          ((fboundp innerDomainConstructor) (unembed innerDomainConstructor))
          (t (unembed domainConstructor)))
        (setq /tracenames (|delete| domainConstructor /tracenames)))))))

defun flattenOperationAlist

(defun flattenOperationAlist (opAlist)
  (prog (op |mmList| |res|)
(return
  (seq
   (progn
     (setq |res| nil)
     (do ((t0 |opAlist| (cdr t0)) (t1 nil))
       ((or (atom t0)
          (progn (setq t1 (car t0)) nil)
          (progn
            (progn (setq op (car t1)) (setq |mmList| (cdr t1)) t1)
            nil))
       nil)
     (setq |res|
       (append |res|
         (prog (t2)
           (setq t2 nil)
           (return
             (do ((t3 |mmList| (cdr t3)) (mm nil))
               ((or (atom t3)
                 (progn (setq mm (car t3)) nil)) (nreverse0 t2))
               (seq
                (exit
                 (setq t2 (cons (cons op mm) t2))))))))
       |res|)))

---

defun mapLetPrint

[getAliasIfTracedMapParameter p889]
[getBpiNameIfTracedMap p890]
[letPrint p885]

--- defun mapLetPrint ---

(defun |mapLetPrint| (x val currentFunction)
  (setq x ([getAliasIfTracedMapParameter| x currentFunction])
  (setq currentFunction ([getBpiNameIfTracedMap| currentFunction])
  ([|letPrint| x val currentFunction])

---
defun letPrint

[lassoc p??]
[isgenvar p886]
[isSharpVarWithNum p886]
[gensymp p??]
[sayBrightlyNT p??]
[bright p??]
[shortenForPrinting p891]
[hasPair p891]
[pname p1045]
[break p906]
[$letAssoc p??]

— defun letPrint —

(defun |letPrint| (x |val| |currentFunction|)
  (prog (y)
    (declare (special |$letAssoc|))
    (return
      (progn
        (cond ($(and |$letAssoc|
          (or
            (setq y (lassoc |currentFunction| |$letAssoc|))
            (setq y (lassoc '|all| |$letAssoc|)))
          (cond
            ((and (or (eq y '|all|)
              (member x y))
              (null
                (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))
              (|sayBrightlyNT| (append (|bright| x) (cons '[: | nil])))
              (prin1 (|shortenForPrinting| |val|))
              (terpri)))))
        (cond
          ($(and (setq y (|hasPair| 'break y))
            (or (eq y '|all|)
              (and (member x y)
                (null (member (elt (pname x) 0) '($ |#|)))
                (null (gensymp x)))))))
          (break)
          (append
            (|bright| |currentFunction|)
            (cons "breaks after"
              (append
                (|bright| x)
                (cons ":= " (cons (|shortenForPrinting| |val|) nil))))))
          (t nil))))
    |val|))
defun Identifier beginning with a sharpsign-number?

This tests if x is an identifier beginning with # followed by a number. [isSharpVar p886]
[ adına p1045]
[qsizes p??]
[_digit p1045]
[dig2fix p??]

— defun isSharpVarWithNum —

(defun isSharpVarWithNum (x)
  (let (p n d ok c)
    (cond
      ((null (isSharpVar x)) nil)
      ((> 2 (setq n (qsizes (setq p ( adına x)))))) nil)
      (t
        (setq ok t)
        (setq c 0)
        (do ((t1 (1- n)) (i 1 (1+ i)))
          ((or (> i t1) (null ok)) nil)
          (setq d (elt p i))
          (when (setq ok (digitp d))
            (setq c (+ (* 10 c) (dig2fix d))))
        (when ok c)))))

— defun isSharpVar —

(defun isSharpVar (x)
  (and (identp x) (char= (schar (symbol-name x) 0) #\#))

— defun isgenvar —

[size p1045]
[digitp p1045]
— defun isgenvar —

(defun isgenvar (x)
  (and (identp x)
       (let ((y (symbol-name x)))
         (and (char= #\$ (elt y 0)) (> (size y) 1) (digitp (elt y 1))))))

— defun letPrint2 —

(defun letPrint2 (x printform currentFunction)
  (prog ($BreakMode flag y)
    (declare (special $BreakMode $letAssoc))
    (return
      (progn
        (setq $BreakMode nil)
        (cond
          ((and $letAssoc)
           (or (setq y (lassoc currentFunction $letAssoc))
            (setq y (lassoc 'all $letAssoc)))
            (cond
              ((and
                (or (eq y 'all) (member x y))
                (null (or (isgenvar x) (isSharpVarWithNum x) (gensymp x))))
              (setq $BreakMode 'letPrint2)
              (setq $BreakMode '')
              (setq flag nil)
              (catch 'letPrint2

```
defun letPrint3

This is the version for use when we have our hands on a function to convert the data into type "Expression" [letPrint2 p887]

<table>
<thead>
<tr>
<th>defun letPrint3</th>
</tr>
</thead>
</table>
| (defun |letPrint3| (x |xval| |printfn| |currentFunction|)
  (prog (|$BreakMode| |flag| y)
    (declare (special |$BreakMode| |$letAssoc|))
    (return

— defun letPrint3 —
(progn
  (setq $BreakMode| nil)
  (cond
    ((and |$letAssoc|
      (or (setq y (lassoc |currentFunction| |$letAssoc|))
        (setq y (lassoc '|all| |$letAssoc|)))))
    (cond
      ((and
        (or (eq y '|all|) (member x y))
        (null (or (isgenvar x) (isSharpVarWithNum x) (gensymp x))))
       (setq |$BreakMode| '|letPrint2|)
       (setq |flag| nil)
       (catch '|letPrint2|
         (mathprint
          (cons '= (cons x (cons (spadcall |xval| |printfn|) nil))))
         |flag|)
       (cond
        ((eq |flag| '|letPrint2|) (print |xval|))
        (t nil))))
    (cond
      ((and
        (setq y (hasPair 'break y))
        (or
         (eq y '|all|)
         (and
          (member x y)
          (null (member (elt (pname x) 0) '|$ |#|)))
          (null (gensymp x))))
       (break
        (append
         (bright |currentFunction|)
         (cons "breaks after"
          (append ((bright x) (cons ":= " (cons |xval| nil))))))
        (t nil)))))


defun getAliasIfTracedMapParameter

[isSharpVarWithNum p886]
[get p??]
[exit p??]
[spaddifference p??]
[string2pint-n p??]
[substring p??]
[pname p1045]
— defun getAliasIfTracedMapParameter —

(defun getAliasIfTracedMapParameter (x currentFunction)
  (prog (aliasList)
    (declare (special $InteractiveFrame))
    (return
      (seq
        (cond
          ((isSharpVarWithNum x)
            (cond
              ((setq aliasList (get currentFunction 'alias $InteractiveFrame))
                (exit (elt aliasList (spaddifference (string2pint-n (substring (pname x) 1 nil) 1) 1))))
            (t x))))))

——

defun getBpiNameIfTracedMap

(defun getBpiNameIfTracedMap (name)
  (prog (lmm bpiName)
    (declare (special $InteractiveFrame /tracenames))
    (return
      (seq
        (cond
          ((setq lmm (get name 'localModemap $InteractiveFrame))
            (cond
              ((member (setq bpiName (cadar lmm)) /tracenames)
                (exit bpiName))
              (t name)))))))

——
defun hasPair

[defun hasPair]

(defun hasPair (key arg)
  (prog (tmp1 a)
    (return
      (cond
        ((atom arg) nil)
        ((and (consp arg)
            (progn
              (setq tmp1 (qcar arg))
              (and (consp tmp1)
                (equal (qcar tmp1) key)
                (progn (setq a (qcdr tmp1)) t))))
             a)
        (t (hasPair key (cdr arg)))))))

---

defun shortenForPrinting

[defun shortenForPrinting]

(defun shortenForPrinting (val)
  (if (isDomainOrPackage val)
      (devaluate val)
      val))

---

defun spadTraceAlias

[defun spadTraceAlias]

(defun spadTraceAlias (domainid op n)
(internl domainid (intern "." "boot") op '1,| (princ-to-string n)))

---

defun getOption

[assoc p??]

— defun getOption —

(defun |getOption| (opt l)
  (let (y)
    (when (setq y (|assoc| opt l)) (cdr y)))))

---

defun reportSpadTrace

[qcar p??]

[sayBrightly p??]

[$traceNoisely p846]

— defun reportSpadTrace —

(defun |reportSpadTrace| (|header| t0)
  (prog (op sig n |t| |msg| |namePart| y |tracePart|)
    (declare (special |$traceNoisely|))
    (return
      (progn
        (setq op (car t0))
        (setq sig (cadr t0))
        (setq n (caddr t0))
        (setq |t| (cdddr t0))
        (cond
          ((null |$traceNoisely|) nil)
          (t
           (setq |msg|
             (cons |header|
               (cons '|%b|
                 (cons op
                   (cons '|:%d|
                     (cons (CDR sig)
                       (cons '|! -> |
                         (cons (car sig)))))))))
```
(consp |t|) (progn (setq y (qcar |t|)) t) (null (null y)))
  (cond
    ((eq y '|all|)
      (cons '|%b| (cons '|all| (cons '|%d| (cons '|vars| nil))))
      (t (cons '| vars: | (cons y nil))))
    (t nil)))
  (sayBrightly| (append |msg| (append |namePart| |tracePart|))))))))

---

defun orderBySlotNumber

[seq p??]
[assocright p??]
[orderList p??]
[exit p??]

—— defun orderBySlotNumber ——

(defun |orderBySlotNumber| (arg)
  (prog (n)
    (return
      (seq
        (assocright
          (orderList)
          (prog (t0)
            (setq t0 nil)
            (return
              (do ((t1 arg (cdr t1)) (x nil))
                ((or (atom t1)
                    (progn (setq x (car t1)) x nil))
                  (progn (progn (setq n (caddr x)) x) nil))
                (nreverse0 t0))
              (seq
                (exit
                  (setq t0 (cons (cons n x) t0))))))))))

———
defun /tracereply
[defun /tracereply ]
[defun /tracereply (]
(prog (|d| domainlist |functionList|)
  (declare (special /tracenames))
  (return
    (seq
      (cond
        ((null /tracenames) " Nothing is traced.")
        (t
          (do ((t0 /tracenames (cdr t0)) (x nil))
            ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
            (seq
              (exit
               (cond
                ((and (consp x)
                  (progn (setq |d| (qcar x)) t)
                  (|isDomainOrPackage| |d|))
                  (setq domainlist (cons (|devaluate| |d|) domainlist)))
                (t
                  (setq |functionList| (cons x |functionList|))))))))
            (append |functionList|
            (append domainlist (cons '|traced| nil)))))))))

— defun /tracereply —

defun spadReply,printName
[spadReply,printName ]
(defun |spadReply,printName| (x)
(prog (d)
  (return
   (seq
    (if (and (and (consp x) (progn (setq d (qcar x)) t))
        (isDomainOrPackage d))
      (exit (devaluate d)))
    (exit x))))

defun spadReply
  [seq p??]
  [exit p??]
  [spadReply,printName p894]
  [/tracenames p??]

  — defun spadReply —

  (defun |spadReply| ()
    (prog ()
      (declare (special /tracenames))
      (return
       (seq
        (prog (t0)
          (setq t0 nil)
          (return
           (do ((t1 /tracenames (cdr t1)) (x nil))
                ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
             (seq
              (exit
               (setq t0 (cons ([spadReply,printName] x) t0))))))))))

  ———

defun spadUntrace

  [isDomainOrPackage p875]
  [userError p??]
  [getOption p892]
  [devaluate p??]
  [assoc p??]
  [sayMSG p331]
  [bright p??]
--- defun spadUntrace ---

(defun |spadUntrace| (domain options)
  (prog (anyiftrue listofoperations domainid |pair| sigslotnumberalist
          op sig n |lv| |bpiPointer| tracename alias |assocPair|
          |newSigSlotNumberAlist|)
    (declare (special |$letAssoc| /tracenames))
    (return
     (seq
      (cond
       ((null (|isDomainOrPackage| domain))
        (|userError| "bad argument to untrace")))
       (t
        (setq anyiftrue (null options))
        (setq listofoperations (|getOption| '|ops:| options))
        (setq domainid (|devaluate| domain))
        (cond
         ((null (setq |pair| (|assoc| domain /tracenames)))
          (|sayMSG|
           (cons " No functions in"
             (append
              (|bright| (|prefix2String| domainid))
              (cons "are now traced." nil)))))))
       (t
        (setq sigslotnumberalist (cdr |pair|))
        (do ((t0 sigslotnumberalist (cdr t0)) (|pair| nil))
            (t (or (atom t0)
                (progn (setq |pair| (car t0)) nil)
                (progn (setq op (car |pair|))
                   (setq sig (cadr |pair|))
                   (setq n (caddr |pair|))
                   (setq |lv| (cadddr |pair|))
                   (setq |bpiPointer| (car (cddddr |pair|)))
                   (setq tracename (cdr (cddddr |pair|)))))

```
(setq alias (caddr (cddddr |pair|)))
(nil)
(nil)

(seq
(exit
(cond
((or anyiftrue (member op listofoperations))
(progn
(bpiiuntrace tracename alias)
(rplac (car (elt domain n)) |bpiPointer|)
(rplac (cdddr |pair|) nil)
(cond
((setq |assocPair|
  (assoc (bpiname |bpiPointer|) |$letAssoc|))
  (setq |$letAssoc| (remover |$letAssoc| |assocPair|))
  (cond
    ((null |$letAssoc|) (setletprintflag nil))
    (t nil)))
  (t nil)))))))))

(setq |newSigSlotNumberAlist|
  (prog (t1)
    (setq t1 nil)
    (return
      (do ((t2 sigslotnumberalist (cdr t2)) (x nil))
          ((or (atom t2) (progn (setq x (car t2)) nil)) (nreverse0 t1))
        (seq
          (exit
            (cond ((cdddr x) (setq t1 (cons x t1))))))))
    (cond
      (|newSigSlotNumberAlist|
       (rplac (cdr |pair|) |newSigSlotNumberAlist|))
      (t
       (setq /tracenames (delasc domain /tracenames)
         (/spadReply))))))))))))))))

(defun remover
[remover p897]

 — defun remover —

(defun remover (1st item)
  (cond
    ((null (consp 1st)) (cond ((equal 1st item) nil) (t 1st)))
    ((equal (car 1st) item) (cdr 1st))
    (t nil)))
(t
 (rplnode lst (remover (car lst) item) (remover (cdr lst) item))
 (rplaca lst (remover (car lst) item))
 (rplacd lst (remover (cdr lst) item))
 lst)))

defun prTraceNames,fn

(seq p??)
 [qcar p??]
 [qcdr p??]
 [isDomainOrPackage p875]
 [exit p??]
 [devaluate p??]

 ---

 — defun prTraceNames,fn —

 (defun |prTraceNames,fn| (x)
 (prog (|d| |t|)
 (return
 (seq
 (if (and (and (consp x)
 (progn (setq |d| (qcar x)) (setq |t| (qcdr x)) t))
 (isDomainOrPackage |d|))
 (exit (cons (|devaluate| |d|) |t|)))
 (exit x))))

 ---

defun prTraceNames

(seq p??)
 [exit p??]
 [prTraceNames,fn p898]
 [/tracenames p??]

 — defun prTraceNames —

 (defun |prTraceNames| ()
 (declare (special /tracenames))
 (seq
 (progn

(do ((t0 /tracenames (cdr t0)) (x nil))
    ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
  (seq
    (exit
      (print (|prTraceNames,fn| x)))))) nil)))

---

defvar $constructors

— initvars —

(defvar |$constructors| nil)

---

defun traceReply

[sayMessage p??]
[sayBrightly p??]
[qcar p??]
isDomainOrPackage p875]
[addTraceItem p902]
isFunctor p??]
isgenvar p886]
[ userError p??]
[seq p??]
[exit p??]
isSubForRedundantMapName p873]
rassocSub p871]
poundsign p??]
sayMSG p331]
[sayBrightlyLength p??]
[flowSegmentedMsg p??]
[concat p1047]
[preString p??]
[abbreviate p??]
[$domains p??]
[$packages p??]
[$constructors p899]
[$linelength p774]
[/tracenames p??]
--- defun traceReply ---

(defun traceReply ()
  (prog (domains packages constructors d functionList)
    (displayList)
    (declare (special domains packages constructors /tracenames $linelength))
    (return
      (seq
        (progn
          (setq domains nil)
          (setq packages nil)
          (setq constructors nil)
          (cond
            ((null /tracenames) (sayMessage " Nothing is traced now."))
            (t
              (sayBrightly " ")
              (do ((t0 /tracenames (cdr t0)) (x nil))
                (or (atom t0) (progn (setq x (car t0)) nil)) nil)
              (seq
                (exit
                  (cond
                    ((atom x)
                      (sayMessage "bad argument to trace")))
                    (t (userError "bad argument to trace"))))
              (setq functionList
                (prog (t1)
                  (setq t1 nil)
                  (return
                    (do ((t2 functionList (cdr t2)) (x nil))
                      (or (atom t2) (progn (setq x (car t2)) nil)) t1)
                    (seq
                      (exit
                        (cond
                          ((null (isSubForRedundantMapName x))
                          (setq t1
                            (append t1
                              (cons (rassocSub x $mapSubNameAlist)
                                (cons " " nil))))))))
                    (cond
                      (functionList)
                      (cond
                        ((eql 2 (|#| functionList))
                          (return)))))
            t))
          (setq functionList
            (prog (t1)
              (setq t1 nil)
              (return
                (do ((t2 functionList (cdr t2)) (x nil))
                  (or (atom t2) (progn (setq x (car t2)) nil)) t1)
                  (seq
                    (exit
                      (cond
                        ((null (isSubForRedundantMapName x))
                          (setq t1
                            (append t1
                              (cons (rassocSub x $mapSubNameAlist)
                                (cons " " nil))))))))
                      (cond
                        (functionList)
                        (cond
                          ((eq l 2 (|#| functionList))
                          (return))))
            ))))
    )))
52.1. TRACE HELP PAGE MAN PAGE

|sayMSG| (cons 'Function traced: |functionList|))

((<= (+ 22 (|sayBrightlyLength| |functionList|)) $lineLength)
  |sayMSG| (cons 'Functions traced: |functionList|)))

(t
  |sayBrightly| "Functions traced:"
  |sayBrightly|
  (|flowSegmentedMsg| |functionList| $lineLength 6))))

(cond
  (|$domains|
    (setq |displayList|
      (|concat|
        (|prefix2String| (CAR |$domains|))
        (prog (t3)
          (setq t3 nil)
          (return
            (do ((t4 (cdr |$domains|) (cdr t4)) (x nil))
              ((or (atom t4) (progn (setq x (car t4)) nil)) t3)
                (seq
                  (exit
                    (setq t3
                      (append t3 (|concat| "", " " (|prefix2String| x))))))))))))

  (atom |displayList|)
    (setq |displayList| (cons |displayList| nil))))
  (|sayBrightly| "Domains traced:"
    (|sayBrightly|
      (|flowSegmentedMsg| |displayList| $lineLength 6)))))

(cond
  (|$packages|
    (setq |displayList|
      (|concat|
        (|prefix2String| (CAR |$packages|))
        (prog (t5)
          (setq t5 nil)
          (return
            (do ((t6 (cdr |$packages|) (cdr t6)) (x nil))
              ((or (atom t6) (progn (setq x (car t6)) nil)) t5)
                (seq
                  (exit
                    (setq t5
                      (append t5 (|concat| ",", " " (|prefix2String| x))))))))))))

  (atom |displayList|)
    (setq |displayList| (cons |displayList| nil))))
  (|sayBrightly| "Packages traced:"
    (|sayBrightly|
      (|flowSegmentedMsg| |displayList| $lineLength 6)))))

(cond
  (|$constructors|
    (setq |displayList|
      (|concat|
        (|abbreviate| (CAR |$constructors|))
        (prog (t7)
(setq t7 nil)
(return
 (do ((t8 (cdr $constructors) (cdr t8)) (x nil))
     ((or (atom t8) (progn (setq x (car t8)) nil)) t7)
     (seq
      (exit
       (setq t7
           (append t7 (concat ', (abbreviate x)))))))
    (cond ((atom $displayList)
       (setq $displayList (cons $displayList nil)))
      (sayBrightly " Parameterized constructors traced:"
       (flowSegmentedMsg $displayList $linelength 6))
       (t nil)))))))

---

defun addTraceItem
[constructor? p??] [isDomain p??] [devaluate p??] [isDomainOrPackage p875] [$constructors p899] [$domains p??] [$packages p??]

— defun addTraceItem —

(defun |addTraceItem| (|d|)
 (declare (special $constructors $domains $packages))
 (cond
  (((constructor? |d|)
     (setq $constructors (cons |d| $constructors)))
   (((isDomain |d|)
     (setq $domains (cons (devaluate |d|) $domains)))
   (((isDomainOrPackage |d|)
     (setq $packages (cons (devaluate |d|) $packages))))

---

defun ?t
[isgenvar p886] [get p??] [sayMSG p331]
(defun |?t| ()
  (let (llm d suffix l)
    (declare (special /tracenames |$InteractiveFrame| |$mapSubNameAlist|))
    (if (null /tracenames)
      (|sayMSG| (|bright| "nothing is traced"))
      (progn
        (dolist (x /tracenames)
          (cond
            (((and (atom x) (null (isgenvar x)))
              (progn
                (cond
                  (((setq llm (|get| x '|localModemap| |$InteractiveFrame|))
                    (setq x (list (cadar llm))))))
                (|sayMSG|
                  ('("Function", Q(|bright| (|rassocSub| x |$mapSubNameAlist|))
                    "traced"))))))
            (dolist (x /tracenames)
              (cond
                (((and (consp x)
                  (progn (setq d (qcar x)) (setq l (qcdr x)) t)
                    (|isDomainOrPackage| d))
                  (progn
                    (setq suffix (cond ((|isDomain| d) "domain") (t "package")))
                    (|sayBrightly|
                      ('(" Functions traced in ", suffix |%b| , (|devaluate| d) |%d| ":")
                        (dolist (x (|orderBySlotNumber| l))
                          (|reportSpadTrace| '| (TAKE 4 x)))
                          (terpri))))))))))))
  (terpri)) )
defun tracelet

| gensym p?? |
| stupidIsSpadFunction p906 |
| bpiname p?? |
| lassoc p?? |
| union p?? |
| setletprintflag p?? |
| isgenvar p886 |
| compileBoot p907 |
| delete p?? |
| $traceletflag p?? |
| $QuickLet p?? |
| $letAssoc p?? |
| $traceletFunctions p?? |

— defun tracelet —

(defun |tracelet| (fn |vars|)
  (prog ($traceletflag |$QuickLet| l)
    (declare (special $traceletflag |$QuickLet| |$letAssoc| |
      $traceletFunctions|))
    (return
     (progn
      (cond
       ((and (gensym fn) (|stupidIsSpadFunction| (eval fn)))
        (setq fn (eval fn))
        (cond
         (((compiled-function-p fn) (setq fn (bpiname fn)))
          (t nil))))
      (cond
       ((eq fn '|Undef|) nil)
       (t
        (setq |vars|
          (cond
           ((eq |vars| '|all|) '|all|)
            ((setq l (lassoc fn |$letAssoc|)) (|union| |vars| l)
             (t |vars|)))
        (setq |$letAssoc| (cons (cons fn |vars|) |$letAssoc|))
        (cond (|$letAssoc| (setletprintflag t)))
        (setq $traceletflag t)
        (setq |$QuickLet| nil)
        (cond
         (((and (null (member fn |$traceletFunctions|)))
           (null (isgenvar fn)))
          (compiled-function-p (symbol-function fn))
          (null (|stupidIsSpadFunction| fn))
          (null (gensym fn)))))


(progn
  (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
  ([|compileBoot| fn]
   (setq |$traceletFunctions|
     ([|delete| fn |$traceletFunctions|])))))))))

---

defun breaklet

[gensymp p?]
[|stupidIsSpadFunction| p906]
[|bpiname| p?]
[|lassoc| p?]
[|assoc| p?]
[|union| p?]
[|setletprintflag| p?]
[|compileBoot| p907]
[|delete| p?]
[|$QuickLet| p?]
[|$letAssoc| p?]
[|$traceletFunctions| p?]

--- defun breaklet ---

(defun |breaklet| (fn |vars|)
  (prog ([|$QuickLet|] [fnEntry] [pair])
    (declare (special |$QuickLet| |$letAssoc| |$traceletFunctions|))
    (return
      (progn
        (cond
          ((and (gensymp fn) (|stupidIsSpadFunction|) (eval fn)))
            (setq fn (eval fn))
            (cond
              (((compiled-function-p fn) (setq fn (bpiname fn)))
              (t nil))))
          (cond
            ((eq fn '|Undef|) nil)
            (t
              (setq fn (assoc fn |$letAssoc|))
              (setq |vars|
                (cond
                  ((setq |pair| (assoc 'break |fnEntry|))
                    (union |vars| (cdr |pair|)))
                  (t |vars|))
                (setq |$letAssoc|
                  (cond
                    )))))))

defun stupidIsSpadFunction

(strpos p1045)
(pname p1045)

— defun stupidIsSpadFunction —

(defun stupidIsSpadFunction (fn)
  (strpos \\; (pname fn) 0 nil))

---

defun break

[MONITOR,EVALTRAN p??]
[enable-backtrace p??]
[sayBrightly p??]
[interrupt p??]
[/breakcondition p??]

— defun break —

(defun break (msg)
  (prog (condition)
    (declare (special /breakcondition))
    (return
      (progn
        (setq condition (\MONITOR,EVALTRAN/ /breakcondition nil))))
(when (eval condition)
  (|sayBrightly| msg)
  (interrupt)))))))

defun compileBoot

[/D,1 p??]

— defun compileBoot —

(defun |compileBoot| (fn)
  (|/D,1| (list fn) '(/comp) nil nil))
Chapter 53

)undo help page Command

53.1 undo help page man page

— undo.help —

====================================================================
A.27. )undo
====================================================================

User Level Required: interpreter

Command Syntax:

- )undo
- )undo integer
- )undo integer [option]
- )undo )redo

where option is one of

- )after
- )before

Command Description:

This command is used to restore the state of the user environment to an earlier point in the interactive session. The argument of an )undo is an integer which must designate some step number in the interactive session.

)undo n
)undo n )after
These commands return the state of the interactive environment to that immediately after step n. If n is a positive number, then n refers to step number n. If n is a negative number, it refers to the nth previous command (that is, undoes the effects of the last -n commands).

A )clear all resets the )undo facility. Otherwise, an )undo undoes the effect of )clear with options properties, value, and mode, and that of a previous undo. If any such system commands are given between steps n and n + 1 (n > 0), their effect is undone for )undo m for any 0 < m <= n.

The command )undo is equivalent to )undo -1 (it undoes the effect of the previous user expression). The command )undo 0 undoes any of the above system commands issued since the last user expression.

)undo n )before

This command returns the state of the interactive environment to that immediately before step n. Any )undo or )clear system commands given before step n will not be undone.

)undo )redo

This command reads the file redo.input. created by the last )undo command. This file consists of all user input lines, excluding those backtracked over due to a previous )undo.

The command )history )write will eliminate the ‘‘undone’’ command lines of your program.

Also See:
o )history

53.2 Evaluation

Some Antique Comments About the Interpreter

EVAL BOOT contains the top level interface to the Scratchpad-II interpreter. The Entry point into the interpreter from the parser is processInteractive.

The type analysis algorithm is contained in the file BOTMUP BOOT, and MODSEL boot, the map handling routines are in MAP BOOT and NEWMAP BOOT, and the interactive coerce routines are in COERGE BOOT and COERCEFN BOOT.

\(^1\)“history” (34.4 p 582)
Conventions: All spad values in the interpreter are passed around in triples. These are lists of three items:

\([\text{value}, \text{mode}, \text{environment}]\]

The value may be wrapped (this is a pair whose CAR is the atom WRAPPED and whose CDR is the value), which indicates that it is a real value, or unwrapped in which case it needs to be EVALed to produce the proper value. The mode is the type of value, and should always be completely specified (not contain $\text{$\$$EmptyMode}$). The environment is always empty, and is included for historical reasons.

Modemaps: Modemaps are descriptions of compiled Spad function which the interpreter uses to perform type analysis. They consist of patterns of types for the arguments, and conditions the types must satisfy for the function to apply. For each function name there is a list of modemaps in file modemap DATABASE for each distinct function with that name. The following is the list of the modemaps for “*” (multiplication. The first modemap (the one with the labels) is for module multiplication which is multiplication of an element of a module by a member of its scalar domain.

This is the signature pattern for the modemap, it is of the form:

\[(\text{DomainOfComputation TargetType <ArgumentType ...>})\]

\| This is the predicate that needs to be satisfied for the modemap to apply\|
\| \|\|
\| V \|
\| \|
\| \|/---------/\|
\| \| \( ( ( \ast1 \ast1 \ast2 \ast1) \)
\| \|\|/-----------------------------------------------------------/
\| \| \( ( \text{AND} (\text{ofCategory} \ast1 (\text{Module} \ast2)) (\text{ofCategory} \ast2 (\text{SimpleRing}))) ) \)
\| \| \text{CATDEF} \(\) This is the file where the function was defined
\| \| \( ( (\ast1 \ast1 \ast2 \ast1) \)
\| \| \( ( \text{AND} (\text{isDomain} \ast2 (\text{Integer})) (\text{ofCategory} \ast1 (\text{AbelianGroup}))) ) \)
\| \| \text{CATDEF}
\| \| \( ( (\ast1 \ast1 \ast2 \ast1) \)
\| \| \( ( \text{AND} \)
\| \| \( (\text{isDomain} \ast2 (\text{NonNegativeInteger})) \)
\| \| \( (\text{ofCategory} \ast1 (\text{AbelianMonoid}))) \)
\| \| \text{CATDEF}
\| \| \( ( (\ast1 \ast1 \ast1 \ast1) ((\text{ofCategory} \ast1 (\text{SemiGroup}))) ) \text{ CATDEF} )
\| \|\)

Environments: Environments associate properties with atoms.

Some common properties are:

- **modeSet**: During interpretation we build a modeSet property for each node in the expression. This is (in theory) a list of all the types possible for the node. In the current implementation these modeSets always contain a single type.
value: Value properties are always triples. This is where the values of variables are stored. We also build value properties for internal nodes during the bottom up phase.

mode: This is the declared type of an identifier.

There are several different environments used in the interpreter:

- $InteractiveFrame: this is the environment where the user values are stored. Any side effects of evaluation of a top-level expression are stored in this environment. It is always used as the starting environment for interpretation.
- $e: This is the name used for $InteractiveFrame while interpreting.
- $env: This is local environment used by the interpreter. Only temporary information (such as types of local variables is stored in $env. It is thrown away after evaluation of each expression.

Frequently used global variables:

- $genValue: if true then evaluate generated code, otherwise leave code unevaluated. If $genValue is false then we are compiling.
- $op: name of the top level operator (unused except in map printing)
- $mapList: list of maps being type analyzed, used in recursive map type analysis.
- $compilingMap: true when compiling a map, used to detect where to THROW when interpret-only is invoked
- $compilingLoop: true when compiling a loop body, used to control nesting level of interp-only loop CATCH points
- $interpOnly: true when in interpret only mode, used to call alternate forms of COLLECT and REPEAT.
- $inCOLLECT: true when compiling a COLLECT, used only for hacked stream compiler.
- $StreamFrame: used in printing streams, it is the environment where local stream variables are stored
- $declaredMode: Weak type propagation for symbols, set in upCOERCE and upLET. This variable is used to determine the alternate polynomial types of Symbols.
- $localVars: list of local variables in a map body
- $MapArgumentTypeList: hack for stream compilation
defun evalDomain

[sayMSG p331]
[concat p1047]
[prefix2String p??]
[startTimingProcess p??]
[eval p??]
[mkEvalable p913]
[stopTimingProcess p??]
[$evalDomain p913]

— defun evalDomain —

(defun evalDomain (form)
(let (result)
(declare (special $evalDomain))
(when $evalDomain
       (sayMSG (concat " instantiating" '|%b| (prefix2String form) '|%d|)))
(startTimingProcess 'instantiation)
(setq result (eval (mkEvalable form)))
(stopTimingProcess 'instantiation)
result))

— defun mkEvalable —

defun mkEvalable

[qcar p??]
[qcdr p??]
[mkEvalable p913]
[devaluate p??]
[mkEvalableRecord p915]
[mkEvalableUnion p915]
[mkEvalableMapping p915]
[loadIfNecessary p??]
[getdatabase p1010]
[mkq p??]
[constructor? p??]
[fbpip p??]
[bpiname p??]
[$Integer p??]
[$EmptyMode p??]

— defun mkEvalable —
(defun mkEvalable (form)
  (let ((op argl kind cosig))
    (declare (special $Integer| $EmptyMode|))
    (cond
      ((consp form) 
       (setq op (qcar form))
       (setq argl (qcdr form))
       (cond
         ((eq op 'quote) form)
         ((eq op 'wrapped) (mkEvalable (devaluate argl)))
         ((eq op '|Record|) (mkEvalableRecord form))
         ((eq op '|Union|) (mkEvalableUnion form))
         ((eq op '|Mapping|) (mkEvalableMapping form))
         ((eq op '|Enumeration|) form)
         (t
          (loadIfNecessary op)
          (setq kind (getdatabase op 'constructorkind))
          (cond
            ((setq cosig (getdatabase op 'cosig))
             (cons op
               (loop for x in argl for typeFlag in (rest cosig)
               collect
                 (cond
                   (typeFlag
                     (cond
                       ((eq kind '|category|) (mkq x))
                       ((vecp x) (mkq x))
                       (t
                        (loadIfNecessary x)
                        (mkEvalable x))))
                   ((and (consp x) (eq (qcar x) 'quote)) x)
                   ((and (consp x) (eq (qcar x) '|#|) (consp (qcdr x))
                    (eq (qcdr (qcdr x)) nil))
                    (list 'size (mkq (qcar (qcdr x))))
                    (t (mkq x)))
                   (t
                    (cons op
                      (loop for x in argl
destiny:collect (|mkEvalable| x))))))))
            ((equal form $EmptyMode| $Integer|)
             (and (identp form) (|constructor?| form))
             (list form))
            ((fbpip form) (bpiname form))
            (t form))))
defun mkEvalableUnion

(defun mkEvalableUnion (form)
  (cond
   ((isTaggedUnion form)
    (cons (car form)
          (loop for item in (rest form)
                collect (list ': (second item) (mkEvalable (third item))))))
   (t
    (cons (car form)
          (loop for d in (rest form)
                collect (mkEvalable d)))))))

defun mkEvalableRecord

(defun mkEvalableRecord (form)
  (cons (car form)
        (loop for item in (rest form)
              collect (list (quote ::) (second item) (mkEvalable (third item))))))

defun mkEvalableMapping

(defun mkEvalableMapping (form)
  (cons (car form)
        (loop for d in (rest form)
              collect (mkEvalable d))))
defun evaluateType

Takes a parsed, unabbreviated type and evaluates it, replacing type valued variables with
their values, and calling bottomUp on non-type valued arguemnts to the constructor and
finally checking to see whether the type satisfies the conditions of its modemap [isDomain-
ValuedVariable p959]

(qcar p??) [qcdr p??] [mkAtree p??] [bottomUp p??] [objVal p??] [getValue p??] [evaluateSignature p920] [member p1048] [evaluateType p916] [constructor? p??] [throwEvalTypeError p919] [$EmptyMode p??] [$expandSegments p??]

(defun |evaluateType| (form)
  (let (|$expandSegments| domain formp op argl)
    (declare (special |$expandSegments| |$EmptyMode|))
    (cond
      ((setq domain (|isDomainValuedVariable| form)) domain)
      ((equal form |$EmptyMode|) form)
      ((eq form '?) |$EmptyMode|)
      ((stringp form) form)
      ((eq form '$) form)
      (t
        (setq |$expandSegments| nil)
        (cond
          ((and (consp form) (eq (qcar form) '|typeOf|) (consp (qcdr form))
            (eq (qcdr (qcdr form)) nil))
          (setq formp (|mkAtree| form))
          (|bottomUp| formp)
          (|objVal| (|getValue| formp)))
          ((consp form)
            (setq op (qcar form))
            (setq argl (qcdr form))
            (cond
              ((eq op 'category)
                (cond
                  ((eq op 'category)
                    (cond
                      ((eq op 'category)
                        (cond
                          ((eq op 'category)
                            (cond
                              ((eq op 'category)
                                (cond
                                  ((eq op 'category)
                                    (cond
...))
                              ))
                        ))
                      ))
                  ))
                ))
            ))
          )))
    )))
  ))

— defun evaluateType —

(defun |evaluateType| (form)
  (let (|$expandSegments| domain formp op argl)
    (declare (special |$expandSegments| |$EmptyMode|))
    (cond
      ((setq domain (|isDomainValuedVariable| form)) domain)
      ((equal form |$EmptyMode|) form)
      ((eq form '?) |$EmptyMode|)
      ((stringp form) form)
      ((eq form '$) form)
      (t
        (setq |$expandSegments| nil)
        (cond
          ((and (consp form) (eq (qcar form) '|typeOf|) (consp (qcdr form))
            (eq (qcdr (qcdr form)) nil))
          (setq formp (|mkAtree| form))
          (|bottomUp| formp)
          (|objVal| (|getValue| formp)))
          ((consp form)
            (setq op (qcar form))
            (setq argl (qcdr form))
            (cond
              ((eq op 'category)
                (cond
...))
          ))
        )))
  ))
53.2. EVALUATION

```lisp
((consp argl)
 (cons op
  (cons (qcar argl)
    (loop for s in (qcdr argl)
      collect (|evaluateSignature| s))))
 (t form)))
(((|member| op '(|Join| |Mapping|))
 (cons op
 (loop for arg in argl
  collect (|evaluateType| arg))))
((eq op '|Union|)
 (cond
  ((and argl (consp (car argl)) (consp (qcdr (car argl))))
   (cons (qcdr (qcdr (car argl)))) nil)
  (|member| (qcar (car argl)) '|([: |Declare|])))
 (cons op
 (loop for item in argl
  collect (list '|:| (second item) (|evaluateType| (third item))))))
(t
 (cons op
 (loop for arg in argl
  collect (|evaluateType| arg))))
((eq op '|Record|)
 (cons op
 (loop for item in argl
  collect (list '|:| (second item) (|evaluateType| (third item))))))
((eq op '|Enumeration|) form)
(t (|evaluateType| form))
((|constructor?| form)
 (if (atom form)
   (|evaluateType| (list form))
   (|throwEvalTypeMsg| 'S2IE0003 (list form form)))))
```

defun Eval args passed to a constructor

Evaluates the arguments passed to a constructor [constructor? p??]
[getConstructorSignature p??]
[throwEvalTypeMsg p919]
[replaceSharps p958]
[categoryForm? p??]
[evaluateType p916]
[evalCategory p959]
[getDatabase p1010]
[mkAtree p?]
[putTarget p?]
[bottomUp p?]
[qcar p?]
[qcdr p?]
[getAndEvalConstructorArgument p958]
[coerceOrRetract p?]
[objValUnwrap p?]
[throwKeyedMsgCannotCoerceWithValue p?]
[makeOrdinal p920]
[$quadSymbol p?]
[$emptyMode p?]

--- defun evaluateType1 ---

(defun evaluateType1 (form)
  (let (op argl sig ml tmp1 m1 z1 zt zv v typeList (argnum 0))
    (declare (special $quadSymbol $emptyMode))
    (setq op (car form))
    (setq argl (cdr form))
    (cond
      (constructor? op)
        (cond
          ((null (setq sig (getConstructorSignature form))
            (throwEvalTypeMsg 'S2IE0005 (list form))))
          (t
            (setq ml (cdr sig))
            (setq ml (replaceSharps ml form))
            (cond
              ((not (eql (|#| argl) (|#| ml)))
                (throwEvalTypeMsg 'S2IE0003 (list form form))
              (t
                (loop for x in argl for m in ml
                   do
                     (setq typeList
                       (cons
                         (cond
                          ((categoryForm? m)
                            (setq m (evaluateType (subst x '$ m)))
                          (if (evalCategory (setq xp (evaluateType x)) m)
                             xp
                             (throwEvalTypeMsg 'S2IE0004 (list form))))))
              (t
                (setq m (evaluateType m))
                (cond
                  ((and (eq (getDatabase (opOf m) 'constructorkind) 'domain) ...)...)
(setq tree (|mkAtree| x))
(putTarget tree m)
(progn
  (setq tmp1 (|bottomUp| tree))
  (and (consp tmp1)
      (eq (qcdr tmp1) nil)))
(setq m1 (qcar tmp1))
(setq z1 (|getAndEvalConstructorArgument| tree))
(setq zt (car z1))
(setq zv (cdr z1))
(if (setq v (|coerceOrRetract| z1 m))
    (objValUnwrap v)
    (|throwKeyedMsgCannotCoerceWithValue| zv zt m)))
(t
  (when (equal x |$EmptyMode|) (setq x |$quadSymbol|))
  (throwEvalTypeMsg| S2I|E0006
    (list (|makeOrdinal| (incf argnum)) m form))))
(typeList))
(cons op (nreverse typeList))))
(t (throwEvalTypeMsg| S2I|E0007 (list op))))

---------

defvar $noEvalTypeMsg

— initvars —

(defun throwEvalTypeMsg
  (spadThrow p??)
  (throwKeyedMsg p??)
  ($noEvalTypeMsg p919)

— defun throwEvalTypeMsg —

(defun |throwEvalTypeMsg| (msg args)
  (declare (special |$noEvalTypeMsg|))
  (if |$noEvalTypeMsg|
      (|spadThrow|)
      (|throwKeyedMsg| msg args)))
defun makeOrdinal

— defun makeOrdinal —

(defun makeOrdinal (i)
  (elt '(*first* *second* *third* *fourth* *fifth* *sixth* *seventh*
       *eighth* *ninth* *tenth*)
  (1- i)))

defun evaluateSignature

Calls evaluateType on a signature [evaluateType p916]

— defun evaluateSignature —

(defun evaluateSignature (sig)
  (cond
    ((and (consp sig) (eq (qcar sig) 'signature) (consp (qcdr sig))
      (consp (qcdr (qcdr sig))) (eq (qcdr (qcdr (qcdr sig))) nil))
      (cons 'signature (cons (qcar (qcdr sig))
        (list
          (loop for z in (qcar (qcdr (qcdr sig)))
            collect (if (eq z '$) z (evaluateType z)))))))
    (t sig)))

53.3 Data Structures

\$frameRecord = [delta1, delta2, ...] \ where delta(i) contains changes in the “backwards” direction. Each delta(i) has the form ((var . proplist)...) \ where proplist denotes an ordinary proplist. For example, an entry of the form ((x (value) (mode (Integer)))...) indicates that to undo 1 step, x’s value is cleared and its mode should be set to (Integer).

A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a a system command is given which alters the environment.
53.4 FUNCTIONS

recordFrame('system) is called before a command is executed, and recordFrame('normal) is called after (see processInteractive1). If no changes are found for former, no special entry is given.

The $previousBindings is a copy of the CAAR $InteractiveFrame. This is used to compute the delta(i)s stored in $frameRecord.

53.4 Functions

Initial Undo Variables

$undoFlag := true  --Default setting for undo is "on"
$frameRecord := nil --Initial setting for frame record
$previousBindings := nil

defvar $undoFlag

    — initvars —

    (defvar |$undoFlag| t "t means we record undo information")

    |

defvar $frameRecord

    — initvars —

    (defvar |$frameRecord| nil "a list of value changes")

    |

defvar $previousBindings

    — initvars —

    (defvar |$previousBindings| nil "a copy of Interactive Frame info for undo")

    |
defvar $reportundo

— initvars —

(defvar $reportundo nil "t means we report the steps undo takes")

——

defun undo
[stringPrefix? p??] [pname p1045] [read p642] [userError p??] [qcdr p??] [qcar p??] [spaddifference p??] [identp p1046] [undoSteps p930] [undoCount p929] [$options p??] [$InteractiveFrame p??]

— defun undo —

(defun undo (l)
(let (tmp1 key s undoWhen n)
(declare (special $options $InteractiveFrame))
(setq undoWhen '|after|)
(when
 (and (consp $options))
 (eq (qcdr $options) nil)
 (progn
 (setq tmp1 (qcar $options))
 (and (consp tmp1)
 (eq (qcdr tmp1) nil)
 (progn (setq key (qcar tmp1)) t)))
(cond
 ((|stringPrefix?| (setq s (pname key)) "redo")
 (setq $options nil)
 (|read| '(|redo.input|)))
((null (|stringPrefix?| s "before"))
 (|userError| "only option to undo is ")redo""))
(t
 (setq undoWhen '|before|))))
(if (null 1)
  (setq n (spaddifference 1))
  (setq n (car l)))
(when (identp n)
  (setq n (parse-integer (pname n)))
  (unless (integerp n)
    (userError| "undo argument must be an integer")))
(setq |$InteractiveFrame| (|undoSteps| (|undoCount| n) undoWhen))
nil))

---

defun recordFrame

[kar p??]
[diffAlist p924]
[seq p??]
[exit p??]
[$undoFlag p921]
[$frameRecord p921]
[$InteractiveFrame p??]
[$previousBindings p921]

— defun recordFrame —

(defun recordFrame (systemNormal)
  (prog (currentAlist delta)
    (declare (special |$undoFlag| |$frameRecord| |$InteractiveFrame|
      |$previousBindings|))
    (return
      (seq
        (cond
          ((null |$undoFlag|) nil)
          (t
            (setq currentAlist (kar |$frameRecord|))
            (setq delta
              (|diffAlist| (caar |$InteractiveFrame|) |$previousBindings|)))
            (cond
              ((eq systemNormal '|system|)
                (cond
                  ((null delta)
                    (return nil))
                  (t
                    (setq delta (cons '|systemCommand| delta))))))
            (setq |$frameRecord| (cons delta |$frameRecord|))
            (setq |$previousBindings|
              (prog (tmp0)
(setq tmp0 nil)
(return
(do ((tmp1 (caar $InteractiveFrame$) (cdr tmp1)) (x nil))
   ((or (atom tmp1)
        (progn (setq x (car tmp1)) nil))
    (nreverse0 tmp0))
  (seq
    (exit
     (setq tmp0
       (cons
        (cons
         (car x)
         (prog (tmp2)
           (setq tmp2 nil)
           (return
            (do ((tmp3 (cdr x) (cdr tmp3)) (y nil))
                ((or (atom tmp3)
                    (progn (setq y (car tmp3)) nil))
                 (nreverse0 tmp2))
              (seq
               (exit
                (setq tmp2
                  (cons
                    (cons
                     (car y)
                     (cdr y)
                    )
                  )
               )))
           ))))
     (car $frameRecord$)))))))

defun diffAlist

diffAlist(new,old) ==
--record only those properties which are different
for (pair := [name,:proplist]) in new repeat
  -- name has an entry both in new and old world
  -- (1) if the old world had no proplist for that variable, then
  --     record NIL as the value of each new property
  -- (2) if the old world does have a proplist for that variable, then
  --     a) for each property with a value: give the old value
    --     b) for each property missing: give NIL as the old value
oldPair := ASSQ(name,old) =>
  null (oldProplist := CDR oldPair) => --record old values of new properties as NIL
  acc := [ [name,:[ [prop] for [prop,:.] in proplist ] ],:acc]
deltas := nil
for (propval := [prop,:val]) in proplist repeat
  null (oldPropval := ASSOC(prop,oldProplist)) => --missing property
  deltas := [ [prop,:deltas]
EQ(CDR oldPropval,val) => 'skip
deltas := [oldPropval,:deltas]
53.4. FUNCTIONS

\[
deltas => \text{acc} := \left[ \left[ \text{name}, :N\text{REVERSE} \deltas \right], :\text{acc} \right]
\]
\[
\text{acc} := \left[ \left[ \text{name}, :\left[ \left[ \text{prop} \right] \text{for } \left[ \text{prop}, . \right] \text{in proplist} \right] \right], :\text{acc} \right]
\]
-- record properties absent on new list (say, from a \text{cl all})
for (oldPair := \left[ \text{name}, :r \right]) \text{in old repeat}
\quad r \text{ and null LASSQ(name,new) =>}
\quad \text{acc} := \left[ \text{oldPair}, :\text{acc} \right]
-- name has an entry both in new and old world
-- (1) if the new world has no proplist for that variable
-- (a) if the old world does, record the old proplist
-- (b) if the old world does not, record nothing
-- (2) if the new world has a proplist for that variable, it has
-- been handled by the first loop.
\text{res} := N\text{REVERSE} \text{acc}
\text{if BOUNDP '$reportundo and $reportundo then reportUndo res}
\text{res}

[\text{assq p1050}]
[\text{tmp1 p??}]
[\text{seq p??}]
[\text{exit p??}]
[\text{assoc p??}]
[\text{lassq p??}]
[\text{reportUndo p927}]

— defun diffAlist —

(defun \text{diffAlist} (new old)
  (prog (proplist oldPair oldProplist val oldPropval deltas prop name r acc res)
    (return
      \text{seq}
      \text{progn}
      \text{(do ((tmp0 new (cdr tmp0)) (pair nil))}
      \text{(or (atom tmp0)}
      \text{\text{(progn (setq pair (car tmp0)) nil)}}
      \text{\text{progn}}
      \text{\text{(progn}}
      \text{\text{(setq name (car pair))}}
      \text{\text{(setq proplist (cdr pair))}}
      \text{\text{pair)}}
      \text{\text{nil)}}
      \text{nil)}
    \text{\text{seq}}
    \text{\text{exit}}
    \text{\text{(cond}}
    \text{\text{(setq oldPair (assq name old))}}
    \text{\text{(cond}}
    \text{\text{((null (setq oldProplist (cdr oldPair)))}}
    \text{\text{(setq acc}}
    \text{\text{(cons}}}
(cons
 name
 (prog (tmp1)
   (setq tmp1 nil)
   (return
    (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
        ((or (atom tmp2)
             (progn (setq tmp3 (car tmp2)) nil)
             (progn
              (progn
               (setq prop (car tmp3))
               tmp3)
             nil))
     (nreverse0 tmp1))
   (seq
    (exit
     (setq tmp1 (cons (cons prop nil) tmp1)))))))
 acc)))
(t
 (setq deltas nil)
 (do ((tmp4 proplist (cdr tmp4)) (|propval| nil))
     ((or (atom tmp4)
          (progn (setq |propval| (car tmp4)) nil)
          (progn
           (setq prop (car |propval|))
           (setq val (cdr |propval|))
           |propval|)
          nil)
     nil)
   (seq
    (exit
     (cond
      ((null (setq oldPropval (|assoc| prop oldProplist)))
       (setq deltas (cons (cons prop nil) deltas)))
      ((eq (cdr oldPropval) val) '|skip|)
      (t (setq deltas (cons oldPropval deltas))))))
 (when deltas
  (setq acc
    (cons (cons name (nreverse deltas)) acc)))))
(t
 (setq acc
 (cons
 (cons
 name
 (prog (tmp5)
   (setq tmp5 nil)
   (return
    (do ((tmp6 proplist (cdr tmp6)) (tmp7 nil))
        ((or (atom tmp6)
             (progn (setq tmp7 (CAR tmp6)) nil)
             (progn
              (setq tmp7 (CAR tmp6))
              (nreverse0 tmp7))
             nil)
    nil)
   (seq
    (exit
     (setq tmp6 (cons (cons prop nil) tmp6)))))))))
defun reportUndo

This function is enabled by setting $\texttt{reportundo}$ to a non-nil value. An example of the output generated is:

\[
r := \text{binary}(22/7)
\]

\[
(1) \quad 11.001
\]

Type: BinaryExpansion

Properties of \% ::
value was: NIL
value is: (\((\text{BinaryExpansion})\) \text{WRAPPED} . #(1 1 1 NIL (0 0 1)))

Properties of r ::
value was: NIL
value is: ((|BinaryExpansion|) WRAPPED . #(1 1) NIL (0 0 1))

---

(defun reportUndo (acc)
  (prog (name proplist curproplist prop value)
    (declare (special |$InteractiveFrame|))
    (return
     (seq
      (do ((tmp0 acc (cdr tmp0)) (tmp1 nil))
        ((or (atom tmp0)
            (progn (setq tmp1 (car tmp0)) nil)
            (progn
              (setq name (car tmp1))
              (setq proplist (cdr tmp1))
              tmp1)
            nil))
      nil)
      (seq
       (exit
        (progn
         (sayBrightly
          (concat '|Properties of | (pname name) " ::"))
         (setq curproplist (lassoc name (caar |$InteractiveFrame|))))
         (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
           ((or (atom tmp2)
               (progn (setq tmp3 (car tmp2)) nil)
               (progn
                 (setq prop (car tmp3))
                 (setq value (cdr tmp3))
                 tmp3)
               nil))
            nil))
      nil)
      (seq
       (exit
        (progn
        ...)))
      nil)))
  nil))
      (seq
       (exit
        (progn
        ...)))
  nil)))


(defun clearFrame
  (clearCmdAll p
    [frameRecord p921]
    [previousBindings p921]
  )
  (defun clearFrame ()
    (declare (special [frameRecord] [previousBindings]))
    (clearCmdAll)
    (setq [frameRecord] nil)
    (setq [previousBindings] nil)
  )

Undo previous n commands

(defun undoCount (n)
  " Undo previous n commands"
  (prog (m)
    (return
      (progn
        (setq m
          (cond
            (>= n 0) (spaddifference (spaddifference [IOindex] n) 1))
            (t (spaddifference n)))))
    (cond
CHAPTER 53. \textit{undo help page command}

\begin{verbatim}
(()= m $IOindex)
( |userError|
  (concat "Magnitude of undo argument must be less than step number (" 
    (princ-to-string $IOindex") ")."))
(t m)))

\end{verbatim}

\begin{verbatim}
defun undoSteps
  -- undoes m previous commands; if )before option, then undo one extra at end
  -- Example: if $IOindex now is 6 and m = 2 then general layout of $frameRecord,
  -- after the call to recordFrame below will be:
  -- (\langle change for systemcommands\rangle
  -- (\langle change for #5\rangle \langle change for system commands\rangle
  -- (\langle change for #4\rangle \langle change for system commands\rangle
  -- (\langle change for #3\rangle \langle change for system commands\rangle
  -- (\langle change for #2\rangle \langle change for system commands\rangle
  -- (\langle change for #1\rangle \langle change for system commands\rangle\rangle) where system
  -- command entries are optional and identified by (systemCommand, change).
  -- For a "undo 3 )after", m = 2 and undoStep will restore the environment
  -- up to, but not including \langle change for #3\rangle.
  -- An "undo 3 )before" will additionally restore \langle change for #3\rangle.
  -- Thus, the later requires one extra undo at the end.

  [writeInputLines p587]
  [spaddifference p??]
  [recordFrame p923]
  [copy p??]
  [undoSingleStep p931]
  [qcdr p??]
  [qcar p??]
  [$IOindex p10]
  [$InteractiveFrame p??]
  [$frameRecord p921]

  --- defun undoSteps ---

(defun undoSteps (m beforeOrAfter)
  (let (tmp1 tmp2 systemDelta lastTailSeen env)
    (declare (special $IOindex $InteractiveFrame $frameRecord))
    (writeInputLines "\texttt{redo} (spaddifference $IOindex m))
    (recordFrame "\texttt{normal})
    (setq env (copy (caar $InteractiveFrame)))
    (do ((i 0 (1+ i)) (framelist $frameRecord (cdr framelist)))
      ((or (> i m) (atom framelist)) nil)
  
\end{verbatim}
53.4. FUNCTIONS

(defun undoSingleStep (changes env) ==
  -- Each change is a name-proplist pair. For each change:
  -- (1) if there exists a proplist in env, then for each prop-value change:
  -- (a) if the prop exists in env, RPLAC in the change value
  -- (b) otherwise, CONS it onto the front of prop-values for that name
  -- (2) add change to the front of env
  -- pp "-------Undoing 1 step-------"
  -- pp changes

  [assq p1050]
  [seq p??]
  [exit p??]
  [lassoc p??]
  [undoLocalModemapHack p933]

  — defun undoSingleStep —

  (defun |undoSingleStep| (changes env) (changes env)
    (prog (name changeList pairlist proplist prop value node)
      (return
       (seq


(progn
  (do ((tmp0 changes (cdr tmp0)) (|change| nil))
      ((or (atom tmp0)
        (progn (setq |change| (car tmp0)) nil)
        (progn
          (setq name (car |change|))
          (setq changeList (cdr |change|))
          |change| nil)
        nil)
      nil)
    (seq
     (exit
      (progn
        (when (lassoc '|localModemap| changeList)
          (setq changeList (|undoLocalModemapHack| changeList)))
        (setq changeList (|undoLocalModemapHack| changeList)))
        (cond
          ((setq pairlist (assq name env))
            (cond
              ((setq proplist (cdr pairlist))
                (do ((tmp1 changeList (cdr tmp1)) (pair nil))
                  ((or (atom tmp1)
                    (progn (setq pair (car tmp1)) nil)
                    (progn
                      (setq prop (car pair))
                      (setq value (cdr pair))
                      pair)
                    nil))
                  nil)
                nil)
              (seq
                (exit
                  (cond
                    ((setq node (assq prop proplist))
                      (rplacd node value))
                    (t (rplacd proplist
                      (cons (car proplist) (cdr proplist))))
                      (rplaca proplist pair))))))))
            (t (rplacd pairlist changeList))))
            (t (setq env (cons |change| env))))))))
  env))))}}
defun undoLocalModemapHack

[seq p??]
[exit p??]

--- defun undoLocalModemapHack ---

(defun undoLocalModemapHack (changeList)
  (prog (name value)
    (return
      (seq
        (prog (tmp0)
          (setq tmp0 nil)
          (return
            (do ((tmp1 changeList (cdr tmp1)) (pair nil))
                ((or (atom tmp1)
                    (progn (setq pair (car tmp1)) nil)
                    (progn
                      (setq name (car pair))
                      (setq value (cdr pair))
                      (pair) nil))
                   (nreverse0 tmp0))
        (seq
          (exit
            (cond
              ((cond
                  ((eq name '|localModemap|) (cons name nil))
                  (t pair))
                (setq tmp0
                  (cons
                    (cond
                      ((eq name '|localModemap|) (cons name nil))
                      (t pair)) tmp0))))))))))

Remove undo lines from history write

Removing undo lines from )hist )write linelist [stringPrefix? p??]
[seq p??]
[exit p??]
[trimString p??]
[substring p??]
[charPosition p??]
[maxindex p??]
— defun removeUndoLines —

(defun removeUndoLines (u)
  "Remove undo lines from history write"
  (prog (xtra savedIOindex s1 m s2 x code c n acc)
    (declare (special $currentLine $IOindex))
    (return
     (seq
      (progn
        (setq xtra
          (cond
            ((stringp $currentLine) (cons $currentLine nil))
            (t (reverse $currentLine)))
            (setq xtra
              (prog (tmp0)
                (setq tmp0 nil)
                (return
                 (do ((tmp1 xtra (cdr tmp1)) (x nil))
                   ((or (atom tmp1) (progn (setq x (car tmp1)) nil))
                     (nreverse0 tmp0))
                 (seq
                  (exit
                   (cond
                     ((null (stringPrefix? "history" x))
                       (setq tmp0 (cons x tmp0)))))))
                (setq u (append u xtra))
              (cond
                ((null (prog (tmp2)
                      (setq tmp2 nil)
                      (return
                       (do ((tmp3 nil tmp2) (tmp4 u (cdr tmp4)) (x nil))
                         ((or tmp3 (atom tmp4) (progn (setq x (car tmp4)) nil))
                           (setq u (append u (做梦Prefix? "undo" x)))))))
                    (setq savedIOindex $IOindex)
                    (setq $IOindex 1)
                    (do ((y u (cdr y)))
                      (atom y) nil))
                (t
                 (setq savedIOindex $IOindex)
                 (setq $IOindex 1)
                 (do ((y u (cdr y)))
                   (atom y) nil))
                (t
                 (setq savedIOindex $IOindex)
                 (setq $IOindex 1)
                 (do ((y u (cdr y)))
                   (atom y) nil)))
      (eq xtra nil) u)
    (eq savedIOindex nil)
    (do ((y u (cdr y)))
      (atom y) nil))
    (eq savedIOindex nil)
    (do ((y u (cdr y)))
      (atom y) nil))
    (eq savedIOindex nil)
    (do ((y u (cdr y)))
      (atom y) nil)))
(seq
  (exit
    (cond
      ((eql (elt (setq x (car y)) 0) #\>)
        (cond
          ((stringPrefix? "undo"
              (setq s (trimString x))
            (setq s1 (trimString (substring s 5 nil)))
            (cond
              ((not (string= s1 "redo"))
                (setq m (charPosition #\# s1 0))
                (setq code
                  (cond
                    ((> (maxindex s1) m) (elt s1 (1+ m)))
                    (t #\a))
                  (setq s2 (trimString (substring s1 0 m)))))
            (setq n
              (cond
                ((string= s1 "redo") 0)
                ((not (string= s2 ""))
                  (undoCount (parse-integer s2))
                  (t -1))
                (rplaca y
                  (concat ">") code (princ-to-string n)))
                (t nil)))
            (t (setq $IOindex (1+ $IOindex))))))))
  (setq acc nil)
  (do ((y (nreverse u) (cdr y)))
    ((atom y) nil)
    (seq
      (exit
        (cond
          ((eql (elt (setq x (car y)) 0) #\>)
            (setq code (elt x 1))
            (setq n (parse-integer (substring x 2 nil)))
            (setq y (cdr y))
            (do ()
              (null y) nil)
          (seq
            (exit
              (progn
                (setq c (car y))
                (cond
                  ((or (eql (elt c 0) #\))
                    (eql (elt c 0) #\>)
                    (setq y (cdr y)))
                  ((eql n 0)
                    (return nil))
                  (t
(setq n (spaddifference n 1))
(setq y (cdr y)))))))
(cond
  ((and y (not (eql code #\b)))
   (setq acc (cons c acc)))
  (t (setq acc (cons x acc)))))))))
(setq |$IOindex| savedIOindex)
(cons |$IOindex| savedIOindex)
(setq acc))))))

_______
Chapter 54

)what help page Command

54.1 what help page man page

— what.help —

====================================================================
A.28. )what
====================================================================

User Level Required: interpreter

Command Syntax:

- )what categories pattern1 [pattern2 ...]
- )what commands pattern1 [pattern2 ...]
- )what domains pattern1 [pattern2 ...]
- )what operations pattern1 [pattern2 ...]
- )what packages pattern1 [pattern2 ...]
- )what synonym pattern1 [pattern2 ...]
- )what things pattern1 [pattern2 ...]
- )apropos pattern1 [pattern2 ...]

Command Description:

This command is used to display lists of things in the system. The patterns are all strings and, if present, restrict the contents of the lists. Only those items that contain one or more of the strings as substrings are displayed. For example,

)what synonym

displays all command synonyms,
CHAPTER 54. \texttt{WHAT HELP PAGE COMMAND}

\texttt{WHAT SYNONYM VER}

displays all command synonyms containing the substring \texttt{`ver'},

\texttt{WHAT SYNONYM VER PR}

displays all command synonyms containing the substring \texttt{`ver'} or the substring \texttt{`pr'}. Output similar to the following will be displayed

\textbf{--------------- System Command Synonyms ---------------}

user-defined synonyms satisfying patterns:

\texttt{ver pr}

\texttt{APR ......................... \texttt{WHAT THINGS}}
\texttt{APROPOS ...................... \texttt{WHAT THINGS}}
\texttt{PROMPT ....................... \texttt{SET MESSAGE PROMPT}}
\texttt{VERSION ..................... \texttt{LISP *YEARWEEK*}}

Several other things can be listed with the \texttt{WHAT} command:

- \texttt{CATEGORIES} displays a list of category constructors.
- \texttt{COMMANDS} displays a list of system commands available at your user-level. Your user-level is set via the \texttt{SET USERLEVEL} command. To get a description of a particular command, such as \texttt{WHAT}, issue \texttt{HELP WHAT}.
- \texttt{DOMAINS} displays a list of domain constructors.
- \texttt{OPERATIONS} displays a list of operations in the system library. It is recommended that you qualify this command with one or more patterns, as there are thousands of operations available. For example, say you are looking for functions that involve computation of eigenvalues. To find their names, try \texttt{WHAT OPERATIONS EIG}. A rather large list of operations is loaded into the workspace when this command is first issued. This list will be deleted when you clear the workspace via \texttt{CLEAR ALL} or \texttt{CLEAR COMPLETELY}. It will be re-created if it is needed again.
- \texttt{PACKAGES} displays a list of package constructors.
- \texttt{SYNONYM} lists system command synonyms.
- \texttt{THINGS} displays all of the above types for items containing the pattern strings as substrings. The command synonym \texttt{APROPOS} is equivalent to \texttt{WHAT THINGS}.

Also See:
- \texttt{DISPLAY}
- \texttt{SET}
- \texttt{SHOW}
54.1. WHAT HELP PAGE MAN PAGE

defvar $whatOptions

— initvars —

(defun $whatOptions
  \'(\|operations\| \|categories\| \|domains\| \|packages\|
   \|commands\| \|synonyms\| \|things\|))

defun what
[whatSpad2Cmd p940]

— defun what —

(defun \|what\| (1)
  \(|\|whatSpad2Cmd\| 1\|))

defun whatSpad2Cmd,fixpat
[qcar p??]
[downcase p??]

— defun whatSpad2Cmd,fixpat —

(defun \|whatSpad2Cmd,fixpat\| (x)
  (let (xp)
    (if (and (consp x) (progn (setq xp (qcar x)) t))
      (downcase xp)
      (downcase x))))

---

\footnote{\text{"display" (29.2 p 535) \text{"set" (45.36 p 808) \text{"show" (46.1 p 814)}}}}
defun whatSpad2Cmd

(reportWhatOptions p941)
(selectOptionLC p479)
(sayKeyedMsg p329)
(seq p??)
(exit p??)
(whatSpad2Cmd, fixpat p939)
(whatSpad2Cmd p940)
(filterAndFormatConstructors p944)
(whatCommands p941)
(apropos p945)
(printSynonyms p474)
($e p??)
($whatOptions p939)

— defun whatSpad2Cmd —

(defun whatSpad2Cmd (arg)
  (prog (|$e| |key0| key args)
    (declare (special |$e| |$whatOptions|))
    (return
      (seq
        (progn
          (setq |$e| |$EmptyEnvironment|)
          (cond
            ((null arg) (|reportWhatOptions|))
            (t
             (setq |key0| (car arg))
             (setq args (cdr arg))
             (setq key (|selectOptionLC| |key0| |$whatOptions| nil))
             (cond
              ((null key) (|sayKeyedMsg| 's2iz0043 nil))
              (t
               (setq args
                 (prog (t0)
                   (setq t0 nil)
                   (setq t0 nil)
                   (return
                    (do ((t1 args (cdr t1)) (p nil))
                      ((or (atom t1)
                          (progn (setq p (car t1)) nil))
                      (nreverse0 t0))
                    (seq
                      (exit
                        (setq t0 (cons (|whatSpad2Cmd,fixpat| p) t0)))))))))))
            (seq
              (cond
                ((eq key '|things|)
defun Show keywords for )what command

[sayBrightly p??]
[$whatOptions p939]

— defun reportWhatOptions —

(defun |reportWhatOptions| ()
  (let (optlist)
    (declare (special |$whatOptions|))
    (setq optlist
      (reduce #'append
        (mapcar #'(lambda (x) `(|%l| " " ,x)) |$whatOptions|)))
    (sayBrightly|
      "(|%b| " )what" |%d| "argument keywords are" |%b| ,@optlist |%d| |%l| " or abbreviations thereof." |%l| |%l| " Issue" |%b| ")what ?" |%d| "for more information.")))))

—

defun The )what commands implementation

[centerAndHighlight p??]
[strconc p??]
(defun whatCommands (patterns)
  (let (label ell)
    (declare (special $systemCommands $linelength $UserLevel))
    (setq label
          (strconc 'System Commands for User Level: |
                    (princ-to-string $UserLevel)))
    (centerAndHighlight label $linelength (specialChar 'hbar))
    (setq ell
          (filterListOfStrings patterns
                      (mapcar #'princ-to-string (commandsForUserLevel $systemCommands))))
    (when patterns
      (if ell
        (sayMessage
         "System commands at this level matching patterns:" %l " " %b
         ,(append (blankList patterns) (list '%d)))
        (sayMessage
         "No system commands at this level matching patterns:" %l " " %b
         ,(append (blankList patterns) (list '%d))))
    (when ell
      (sayAsManyPerLineAsPossible ell))
    (say " ")
    (unless patterns (sayKeyedMsg 's2iz0046 nil))))

(defun filterListOfStrings (patterns names)
  ;; Find all names contained in a pattern
  ;; Names and patterns are lists of strings. This returns a list of strings in
  ;; names that contains any of the strings in the patterns
  ;; (satisfiesRegularExpressions p93)
  ;;
  ;; (defun filterListOfStrings (patterns names)

  ;; This returns a list of strings in names that contains any of the strings in
  ;; the patterns
  ;; (satisfiesRegularExpressions p93)
  ;;
  ;; (defun filterListOfStrings (patterns names)

defun Find function of names contained in pattern

The argument names and patterns are lists of strings. The argument fn is something like
CAR or CADR This returns a list of strings in names that contains any of the strings in
patterns [satisfiesRegularExpression p943]

— defun filterListOfStringsWithFn —

(defun filterListOfStringsWithFn (patterns names fn)
  (let (result)
    (if (or (null patterns) (null names))
      names
      (dolist (name (reverse names) result)
        (when (satisfiesRegularExpression (funcall fn name) patterns)
          (push name result))))))

— defun satisfiesRegularExpression —

(defun satisfiesRegularExpression (name patterns)
  (let ((dname (downcase (copy name))))
    (dolist (pattern patterns)
      (when (strpos pattern dname 0 "@")
        (return-from nil t)))))
defun filterAndFormatConstructors

(defun filterAndFormatConstructors (constrType label patterns)
  (prog (l)
    (declare (special $linelength))
    (return
      (progn (|centerAndHighlight| label $linelength (|specialChar| '|hbar|))
        (setq l
          ([filterListOfStringsWithFn patterns
            ([whatConstructors| constrType)
              ([function| cdr]))]
          (cond (patterns
                    (cond
                      ((null l)
                        ([sayMessage|
                          (cons " No "
                          (cons label
                            (cons " with names matching patterns:"
                              (cons '|%l|
                                (cons " "
                                  (cons '|%b|
                                    (append (|blankList| patterns)
                                      (cons '|%d| nil)))))
                                )
                              ))))))
                    (t
                        ([sayMessage|
                          (cons label
                            (cons " with names matching patterns:"
                              (cons '|%l|
                                (cons " "
                                  (cons '|%b|
                                    (append (|blankList| patterns)
                                      (cons '|%d| nil))))))))
                      )]])
          (cond (l ([pp2Cols| l]))))))

— defun filterAndFormatConstructors —
54.1. WHAT HELP PAGE MAN PAGE

defun whatConstructors

; [boot-equal p??]
; [getdatabase p1010]
; [seq p??]
; [msort p??]
; [exit p??]

— defun whatConstructors —

(defun |whatConstructors| (constrType)
  (prog nil
    (return
      (seq
        (msort
          (prog (t0)
            (setq t0 nil)
            (return
              (do ((t1 (|allConstructors|) (cdr t1)) (|con| nil))
                  ((or (atom t1) (progn (setq |con| (car t1)) nil)) (nreverse0 t0))
                (seq
                  (exit
                    (cond
                      ((equal (getdatabase |con| 'constructorkind) constrType)
                        (setq t0
                          (cons
                            (cons
                              (getdatabase |con| 'abbreviation)
                              (string |con|))
                            t0)))))))))))

Display all operation names containing the fragment

Argument 1 is a list of operation name fragments. This displays all operation names containing these fragments [allOperations p1033]

[filterListOfStrings p942]
[seq p??]
[exit p??]
[downcase p??]
[sayMessage p??]
[sayAsManyPerLineAsPossible p??]
[msort p??]
[sayKeyedMsg p329]
— defun apropos —

(defun apropos (arg)
"Display all operation names containing the fragment"
(prog (ops)
  (return
   (seq
    (progn
      (setq ops
        (cond
          ((null arg) (allOperations))
          (t
           (filterListOfStrings
             (progn (t0)
               (setq t0 nil)
               (return
                 (do ((t1 arg (cdr t1)) (p nil))
                     ((or (atom t1) (progn (setq p (car t1)) nil))
                      (nreverse0 t0))
                 (seq (exit (setq t0 (cons (downcase (princ-to-string p)) t0))))))
             (allOperations)))))
        (cond
          (ops
            (sayMessage "Operations whose names satisfy the above pattern(s):")
            (sayAsManyPerLineAsPossible (msort ops))
            (sayKeyedMsg 's2if0011 (cons (car ops) nil)))
          (t
            (sayMessage "There are no operations containing those patterns")
            nil))))))
Chapter 55

)with help page Command

55.1 with help page man page

— with.help —

This command is obsolete.
This has been renamed )library.

See also:
o )library

— defun with —

(defun |with| (args)
  ([library| args])

———

1 “library” (66.1 p 1013)
Chapter 56

)workfiles help page Command

56.1 workfiles help page man page

defun workfiles

[workfilesSpad2Cmd ]

|— defun workfiles —|

(defun |workfiles| (l)
  (|workfilesSpad2Cmd| l))

—

defun workfilesSpad2Cmd

[throwKeyedMsg ]

[selectOptionLC ]

[pathname ]

[delete ]

[makeInputFilename ]

[sayKeyedMsg ]

[namestring ]

[updateSourceFiles ]

[say ]

[centerAndHighlight ]

[specialChar ]

[sortby ]

[sayBrightly ]
# Defun workfilesSpad2Cmd

```lisp
(defun workfilesSpad2Cmd (args)
  (let (deleteflag type flist type1 fl)
    (declare (special $options $sourceFiles $linelength))
    (cond
      (args (throwKeyedMsg 's2iz0047 nil))
      (t
        (setq deleteflag nil)
        (do ((t0 $options (cdr t0)) (t1 nil))
          ((or (atom t0) (progn (setq t1 (car t0)) nil) (progn (setq type (car t1)) t1) nil)
          nil)
        (setq type1
          (selectOptionLC type '(boot lisp meta delete) nil))
        (cond
          ((null type1) (throwKeyedMsg 's2iz0048 (cons type nil)))
          ((eq type1 'delete) (setq deleteflag t)))
        (do ((t2 $options (cdr t2)) (t3 nil))
          ((or (atom t2) (progn (setq t3 (CAR t2)) nil) (progn (setq type (car t3)) (setq flist (cdr t3)) t3) nil)) nil)
        (setq type1 (selectOptionLC type 'boot lisp meta delete) nil)
        (unless (eq type1 'delete)
          (dolist (file flist)
            (setq fl (pathname (list file type1 ")")))
          (cond
            (deleteflag
              (setq $sourceFiles (delete fl $sourceFiles)))
            (null (makeInputFilename fl1))
            (null (sayKeyedMsg 's2iz0035 (namestring (pathname fl)) nil)
              (null (updateSourceFiles fl))))
          (null nil))
        nil)
    (say " ")
    (centerAndHighlight
      'User-specified work files $linelength
        ('specialChar |hbar|)
      (say " ")
    (if (null $sourceFiles)
      nil)))
```

---

**Options:**

- `$options` - List of options passed to the `workfilesSpad2Cmd` function.
- `$sourceFiles` - List of source files.
- `$linelength` - Line length for printing.

**Example Usage:**

```lisp
(defun workfilesSpad2Cmd (args)
  ;; Example usage of workfilesSpad2Cmd
  (workfilesSpad2Cmd 'boot 'lisp 'meta 'delete)
```

**Purpose:**

The `workfilesSpad2Cmd` function is used to manage workfiles for Spad2Cmd, a tool for managing source files in a specific context. It processes options, source files, and line length to manage file operations such as deleting, listing, or updating source files.

**Components:**

- The function initializes variables `deleteflag`, `type`, `flist`, and `type1`.
- It iterates over the option list, processing each option and its corresponding list of source files.
- It uses `selectOptionLC` to select an option from a set of choices (boot, lisp, meta, delete).
- It handles deletion of source files based on the selected option.
- It prints a message indicating the processed work files with the specified line length.
(say "no files specified")
(progn
  (setq |$sourceFiles| (sortby '|pathnameType| |$sourceFiles|))
  (do ((t5 |$sourceFiles| (cdr t5)) (fl nil))
      ((or (atom t5) (progn (setq fl (car t5)) nil)) nil)
    (|sayBrightly| (list " " (|namestring| fl))))))
Chapter 57

)zsyste<caret>mdevelopment help page Command

57.1  zsystemdevelopment help page man page

defun zsystemdevelopment

[zsystemDevelopmentSpad2Cmd p953]

— defun zsystemdevelopment —

(defun |zsystemdevelopment| (arg)
  (|zsystemDevelopmentSpad2Cmd| arg))

———

defun zsystemDevelopmentSpad2Cmd

[zsystemdevelopment1 p954]
[|$InteractiveMode p22]

— defun zsystemDevelopmentSpad2Cmd —

(defun |zsystemDevelopmentSpad2Cmd| (arg)
  (declare (special |$InteractiveMode|)))
  (|zsystemdevelopment1| arg |$InteractiveMode|)))

———
defun zsystemdevelopment1
  (defun zsystemdevelopment1 (arg im)
    (let ((|$InteractiveMode| fromopt opt optargs newopt opt1 constream upf fun)
      (declare (special |$InteractiveMode| /wsname /version |$options|))
      (setq |$InteractiveMode| im)
      (setq fromopt nil)
      (do ((t0 |$options| (cdr t0)) (t1 nil))
           ((or (atom t0)
                (progn (setq t1 (car t0)) nil)
                (progn
                  (setq opt (CAR t1))
                  (setq optargs (CDR t1))
                  (setq t1 nil))
                nil)
        (setq opt1 (|selectOptionLC| opt (|from|) nil))
        (when (eq opt1 (|from|)) (setq fromopt (cons (cons 'from optargs) nil)))
      (do ((t2 |$options| (cdr t2)) (t3 nil))
          ((or (atom t2)
               (progn (setq t3 (car t2)) nil)
               (progn
                 (setq opt (car t3))
                 (setq optargs (cdr t3))
                 t3)
             nil)
          nil))
    — defun zsystemdevelopment1 —
  )
(cond
  ((eq opt1 '|from|) nil)
  ((eq opt '|c|) (/D,1| newopt ((COMP) nil nil))
   (|/D,1| newopt 'define nil nil))
  ((eq opt '|d|) (|/D,1| newopt 'define nil t))
  ((eq opt '|dt|) (|/D,1| newopt ((COMP) nil t))
   (|/D,1| newopt ((COMP) nil 'tracelet))
  ((eq opt '|ct|) (|/D,1| newopt ((COMP) t nil))
   (|/D,1| newopt ((COMP) t t))
  ((eq opt '|ctl|) (|/D,1| newopt nil t nil))
  ((eq opt '|version|) (|version|))
  ((eq opt '|pause|)
   (setq constream
     (defiostream '((device . console) (qual . v)) 120 0))
   (next constream)
   (shut constream))
  ((or
     (eq opt '|update|)
     (eq opt '|patch|))
   (setq |$InteractiveMode| nil)
   (setq upf
     (cons
      (or (kar optargs) /version)
      (cons
       (or (kadr optargs) /wsname)
       (cons (or (kaddr optargs) '* nil))))))
  (setq fun
    (cond
      ((eq opt '|patch|) '/update-lib-1)
      (t '/update-1)))
  (catch 'filenam (funcall fun upf))
  (|sayMessage| " Update/patch is completed.")
  ((null optargs)
   (|sayBrightly| "(" An argument is required for" ,0(|bright| opt)))
  (t
   (|sayMessage|
    "(" Unknown option:" ,0(|bright| opt)
    |%l| " Available options are" ,0(|bright| "c ct e ec ect cls pause update patch compare record")\)))))))
CHAPTER 57. )ZSYSTEMDEVELOPMENT HELP PAGE COMMAND
Chapter 58

Handlers for Special Forms

This file contains the functions which do type analysis and evaluation of special functions in the interpreter. Special functions are ones which are not defined in the algebra code, such as assignment, construct, COLLECT and declaration.

Operators which require special handlers all have a LISP “up” property which is the name of the special handler, which is always the word “up” followed by the operator name. If an operator has this “up” property the handler is called automatically from bottomUp instead of general modemap selection.

The up handlers are usually split into two pieces, the first is the up function itself, which performs the type analysis, and an “eval” function, which generates (and executes, if required) the code for the function.

The up functions always take a single argument, which is the entire attributed tree for the operation, and return the modeSet of the node, which is a singleton list containing the type computed for the node.

The eval functions can take any arguments deemed necessary. Actual evaluation is done if $genValue$ is true, otherwise code is generated.

(See the function analyzeMap for other things that may affect what is generated in these functions.)

These functions are required to do two things:

1. do a putValue on the operator vector with the computed value of the node, which is a triple. This is usually done in the eval functions.

2. do a putModeSet on the operator vector with a list of the computed type of the node. This is usually done in the up functions.

There are several special modes used in these functions:

1. Void is the mode that should be used for all statements that do not otherwise return values, such as declarations, loops, IF-THEN’s without ELSE’s, etc..
2. $NoValueMode$ and $ThrowAwayMode$ used to be used in situations where Void is now used, and are being phased out completely.

```lisp
(defun getAndEvalConstructorArgument
  [getValue p]
  [objMode p]
  [isWrapped p]
  [objVal p]
  [isLocalVar p]
  [compFailure]
  [objNewWrap p]
  [timedEVALFUN p]

  — defun getAndEvalConstructorArgument —

(let (triple)
  (setq triple (getValue tree))
  (cond
    ((eq (objMode triple) '(Domain)) triple)
    ((isWrapped (objVal triple)) triple)
    ((isLocalVar (objVal triple))
      (compFailure "Local variable or parameter used in type")
    (t
      (objNewWrap (timedEVALFUN (objVal triple) (objMode triple))))))

  — — — — — — — — — — — — — — — — — — — — — —

(defun replaceSharps
  Replaces all sharps in x by the arguments of domain d. Replaces all replaces the triangle variables [subCopy p]
  [$TriangleVariableList p]
  [$FormalMapVariableList p]

  — defun replaceSharps —

(let (sl)
  (declare (special $TriangleVariableList $FormalMapVariableList))
  (loop for e in (rest d) for var in $FormalMapVariableList
    do (setq sl (cons (cons var e) sl)))
  (setq x (subCopy x sl))
  (setq sl nil))
```
(loop for e in (rest d) for var in $TriangleVariableList$
do (setq sl (cons (cons e sl)))
  (|subCopy| x sl))

---

defun isDomainValuedVariable

Returns the value of form if form is a variable with a type value [identp p1046]
[get p??]
[member p1048]
[objMode p??]
[objValUnwrap p??]
[$e p??]
[$env p??]
[$InteractiveFrame p??]

| defun isDomainValuedVariable |

(defun |isDomainValuedVariable| (form)
  (let (val)
    (declare (special |$e| |$env| |$InteractiveFrame|))
    (when (and (identp form)
      (setq val
        (or (|get| form '|value| |$InteractiveFrame|)
          (and (consp |$env|) (|get| form '|value| |$env|))
          (and (consp |$e|) (|get| form '|value| |$e|)))
          (|member| (|objMode| val) '(('|Domain|) '|SubDomain| ('|Domain|))))
          (|objValUnwrap| val)))))

---

defun evalCategory

[ofCategory p419]
[isPartialMode p420]

| defun evalCategory |

(defun |evalCategory| (d c)
  (or (|isPartialMode| d) (|ofCategory| d c)))
Chapter 59

Handling input files

defun Handle .axiom.input file

[/editfile p515]

  — defun readSpadProfileIfThere —

(defun |readSpadProfileIfThere| ()
  (let ((file (list '|.axiom| '|input|)))
    (declare (special /editfile))
    (when (makeInputFilename file) (setq /editfile file) (/rq))))

defvar $boot-line-stack

  — initvars —

  (defvar boot-line-stack nil "List of lines returned from preparse")

defvar $in-stream

  — initvars —

  (defvar in-stream t "Current input stream.")
defvar $out-stream

— initvars —
(defvar out-stream t "Current output stream.")

defvar $file-closed

— initvars —
(defvar file-closed nil "Way to stop EOF tests for console input.")

defvar $echo-meta

— initvars —
(defvar echo-meta nil "T if you want a listing of what has been read.")

defvar $noSubsumption

— initvars —
(defvar |$noSubsumption| t)
defvar $envHashTable

The $envHashTable variable is a hashtable that optimizes lookups in the environment, which normally involve search. This gets populated in the addBinding function.

— initvars —

(defvar |$envHashTable| nil)

defun Dynamically add bindings to the environment

[getProplist p964]
[addBindingInteractive p967]
[hput p1044]
|$InteractiveMode p22]
[$envHashTable p963]

— defun addBinding —

(defun |addBinding| (var proplist e)
  (let (tailContour tailEnv tmp1 curContour lx)
    (declare (special |$InteractiveMode| |$envHashTable|))
    (setq curContour (caar e))
    (setq tailContour (cdar e))
    (setq tailEnv (cdr e))
    (cond
     ((eq proplist (|getProplist| var e)) e)
     (t
      (when |$envHashTable|
        (do ((prop proplist (cdr prop)) (u nil))
          ((or (atom prop)
              (progn (setq u (car prop)) nil))
            nil)
        (hput |$envHashTable| (list var (car u)) t)))
      (cond
       (|$InteractiveMode| (|addBindingInteractive| var proplist e))
       (t
        (when (and (consp curContour)
                    (progn
                      (setq tmp1 (qcar curContour))
                      (and (consp tmp1) (equal (qcar tmp1) var)))
                        (setq curContour (cdr curContour)))
          (setq lx (cons var proplist))
          (cons (cons (cons lx curContour) tailContour) tailEnv)))))))

— —
defun Fetch a property list for a symbol from CategoryFrame

(defun getProplist (p)
  (search p $CategoryFrame)
)

— defun getProplist —

(defun getProplist (x e)
  (let (u pl)
    (declare (special $CategoryFrame))
    (cond
      ((null (atom x)) (getProplist (car x) e))
      ((setq u (search x e)) u)
      ((setq pl (search x $CategoryFrame)) pl))))

________

defun Search for a binding in the environment list

(defun searchCurrentEnv (x currentEnv)
  (for contour in currentEnv repeat
    if u:= assq (x, contour) then return (signal:= u)
  )
  (kdr signal)
)

— defun searchCurrentEnv —

(defun searchCurrentEnv (x currentEnv)
  (let ((curEnv (car e)) (tailEnv (cdr e)))
    (or (searchCurrentEnv x curEnv) (searchTailEnv x tailEnv))))

________

defun Search for a binding in the current environment

searchCurrentEnv(x, currentEnv) ==
  for contour in currentEnv repeat
    if u:= assq (x, contour) then return (signal:= u)
  KDR signal

— defun searchCurrentEnv —

(defun searchCurrentEnv (x currentEnv)
(prog (u signal)
   (return
    (seq
     (progn
      (do ((thisenv currentEnv (cdr thisenv)) (contour nil))
        ((or (atom thisenv) (progn (setq contour (car thisenv)) nil)) nil)
      (seq
       (exit
        (cond
         ((setq u (assq x contour)) (return (setq signal u)))
         (t nil))))))
     (kdr signal))))

---

defun searchTailEnv

;searchTailEnv(x, e) ==
; for env in e repeat
;   signal :=
;       for contour in env repeat
;         if (u := ASSQ(x, contour)) and ASSQ("FLUID", u) then return (signal := u)
;       if signal then return signal
;    KDR signal

[assq p1050]
[kdr p??]

--- defun searchTailEnv ---

(defun |searchTailEnv| (x e)
 (prog (u signal)
   (return
    (seq
     (progn
      (do ((thise e (cdr thise)) (env nil))
        ((or (atom thise) (progn (setq env (car thise)) nil)) nil)
      (seq
       (exit
        (setq signal
          (progn
            (do ((cone env (cdr cone)) (contour nil))
              ((or (atom cone) (progn (setq contour (car cone)) nil)) nil)
            (seq
             (exit
              (cond
               ((and (setq u (assq x contour)) (assq 'fluid u))
                nil)))))))))))
     (kdr signal)))))
(return (setq signal u)))
(t nil)))))))
(cond
(signal (return signal))
(t nil))))))))
(kdr signal))))))}
Chapter 60

File Parsing

defun Bind a variable in the interactive environment

(defun addBindingInteractive)
(let ((curContour (caar e)) u)
  (cond
    ((setq u (assq var curContour)) (rplacd u proplist) e)
    (t (rplac (caar e) (cons (cons var proplist) curContour)) e))))

defvar $line-handler

— initvars —
(deparameter line-handler 'next-META-line "Who grabs lines for us.")

defvar $spad-errors

— initvars —
(defvar $spad_errors (vector 0 0 0))

---

defvar $xtokenreader

— initvars —

(defvar xtokenreader 'spadtok)

---

defun Initialize the spad reader

(defun Initialize the spad reader ()
  (declare (special $spad_errors spaderrorstream *standard-output* xtokenreader line-handler meta-error-handler file-closed boot-line-stack))
  (setq $spad_errors (vector 0 0 0))
  (setq spaderrorstream *standard-output*)
  (setq xtokenreader 'get-BOOT-token)
  (setq line-Handler 'next-BOOT-line)
  (setq meta-error-handler 'spad-syntax-error)
  (setq file-closed nil)
  (next-lines-clear)
  (ioclear))

---
defun spad-syntax-error

[bumpercount p969]
[consoleinput p969]
[spad-long-error p969]
[spad-short-error p970]
[ioclear p972]
[debugmode p972]
[spad-reader p972]

---

(defun spad-syntax-error (&rest byebye)
  "Print syntax error indication, underline character, scrub line."
  (declare (special debugmode))
  (bumperrorcount '|'syntax|)
  (cond ((and (eq debugmode 'yes) (not(consoleinput in-stream)))
    (spad-long-error))
    ((spad-short-error)))
  (ioclear)
  (throw 'spad_reader nil))

---

defun spad-long-error

[spad-error-loc p970]
[iostat p970]
[out-stream p962]
[spaderrorstream p970]

---

(defun spad-long-error ()
  (declare (special spaderrorstream))
  (spad-error-loc spaderrorstream)
  (iostat)
  (unless (equal out-stream spaderrorstream)
    (spad-error-loc out-stream)
    (terpri out-stream)))

---
defun spad-short-error

(line-past-end-p p)
(line-print p)
($current-line p)

— defun spad-short-error —

(defun spad-short-error ()
  (if (line-past-end-p Current-Line)
      (format t "The current line is empty.\n")
      (progn
        (format t "The current line is:\n\n")
        (line-print current-line))))

——

defun spad-error-loc

defun spad-error-loc

— defun spad-error-loc —

(defun spad-error-loc (str)
  (format str "******** Boot Syntax Error detected ********"))

——

defun iostat

(token-stack-show p)
(next-lines-show p)
($boot p)
($spad p)
($current-line p)

— defun iostat —

(defun iostat ()
  "Tell me what the current state of the parsing world is."
  (declare (special $boot $spad))
  (if (line-past-end-p Current-Line)
      (format t "\nThe current line is empty.\n")
(progn
  (format t "The current line is:\"%\%\")
  (line-print current-line)))
(if (or $boot $spad) (next-lines-show)
(token-stack-show)
nil)

defun next-lines-show
[boot-line-stack p961]

— defun next-lines-show —

(defun next-lines-show ()
  (declare (special boot-line-stack))
  (and boot-line-stack (format t "Currently preparsed lines are:\"\%\")
    (mapcar #'(lambda (line)
      (format t \"\%> \%A\%\" (car line) (cdr Line)))
    boot-line-stack))

defun token-stack-show
[token-type p??]
[valid-tokens p??]
[current-token p??]
[next-token p??]
[prior-token p??]

— defun token-stack-show —

(defun token-stack-show ()
  (if (= valid-tokens 0)
    (format t \"There are no valid tokens.\")
    (format t \"The number of valid tokens is \%S.\" valid-tokens))
  (when (> valid-tokens 0)
    (format t \"The current token is\")
    (describe current-token))
  (when (> valid-tokens 1)
    (format t \"The next token is\")
    (describe next-token))
  (when (token-type prior-token)
defun ioclear

The IO state manipulation routines assume that

- one I/O stream pair is in effect at any moment
- there is a current line
- there is a current token and a next token
- there is a reduction stack

(defun ioclear (&optional (in t) (out t))
  (declare (special current-fragment current-line $boot $spad)
           (ignore in out))
  (setq current-fragment nil)
  (line-clear current-line)
  (token-install nil nil current-token nil)
  (token-install nil nil next-token nil)
  (token-install nil nil prior-token nil)
  (reduce-stack-clear)
  (if (or $boot $spad) (next-lines-clear)) nil)

---

defun Set boot-line-stack to nil

[boot-line-stack p961]

---

---

---
(defun next-lines-clear ()
  (setq boot-line-stack nil))
Chapter 61

Handling output

61.1 Special Character Tables

defvar $defaultSpecialCharacters

— initvars —

(defvar |$defaultSpecialCharacters| (list
(int-char 28) ; upper left corner
(int-char 27) ; upper right corner
(int-char 30) ; lower left corner
(int-char 31) ; lower right corner
(int-char 79) ; vertical bar
(int-char 45) ; horizontal bar
(int-char 144) ; APL quad
(int-char 173) ; left bracket
(int-char 189) ; right bracket
(int-char 192) ; left brace
(int-char 208) ; right brace
(int-char 59) ; top box tee
(int-char 62) ; bottom box tee
(int-char 63) ; right box tee
(int-char 61) ; left box tee
(int-char 44) ; center box tee
(int-char 224))) ; back slash

———
defvar $plainSpecialCharacters0

— initvars —

(defvar $plainSpecialCharacters0 (list
  (int-char 78) ; upper left corner (+)
  (int-char 78) ; upper right corner (+)
  (int-char 78) ; lower left corner (+)
  (int-char 78) ; lower right corner (+)
  (int-char 79) ; vertical bar
  (int-char 96) ; horizontal bar (-)
  (int-char 111) ; APL quad (?)
  (int-char 173) ; left bracket
  (int-char 189) ; right bracket
  (int-char 192) ; left brace
  (int-char 208) ; right brace
  (int-char 78) ; top box tee (+)
  (int-char 78) ; bottom box tee (+)
  (int-char 78) ; right box tee (+)
  (int-char 78) ; left box tee (+)
  (int-char 224) ; back slash)

——

defvar $plainSpecialCharacters1

— initvars —

(defvar $plainSpecialCharacters1 (list
  (int-char 107) ; upper left corner (,)
  (int-char 107) ; upper right corner (,)
  (int-char 125) ; lower left corner (')
  (int-char 125) ; lower right corner (')
  (int-char 79) ; vertical bar
  (int-char 96) ; horizontal bar (-)
  (int-char 111) ; APL quad (?)
  (int-char 173) ; left bracket
  (int-char 189) ; right bracket
  (int-char 192) ; left brace
  (int-char 208) ; right brace
  (int-char 78) ; top box tee (+)
  (int-char 78) ; bottom box tee (+)
  (int-char 78) ; right box tee (+)
  (int-char 78) ; left box tee (+)
defvar $plainSpecialCharacters2

— initvars —

(defvar |$plainSpecialCharacters2| (list
(int-char 79) ; upper left corner (|)
(int-char 79) ; upper right corner (|)
(int-char 79) ; lower left corner (|)
(int-char 79) ; lower right corner (|)
(int-char 79) ; vertical bar
(int-char 96) ; horizontal bar (-)
(int-char 111) ; APL quad (?)
(int-char 173) ; left bracket
(int-char 189) ; right bracket
(int-char 192) ; left brace
(int-char 208) ; right brace
(int-char 78) ; top box tee (+)
(int-char 78) ; bottom box tee (+)
(int-char 78) ; right box tee (+)
(int-char 78) ; left box tee (+)
(int-char 78) ; center box tee (+)
(int-char 224))) ; back slash

defvar $plainSpecialCharacters3

— initvars —

(defvar |$plainSpecialCharacters3| (list
(int-char 96) ; upper left corner (-)
(int-char 96) ; upper right corner (-)
(int-char 96) ; lower left corner (-)
(int-char 96) ; lower right corner (-)
(int-char 79) ; vertical bar
(int-char 96) ; horizontal bar (-)
(int-char 111) ; APL quad (?)
(int-char 173) ; left bracket

(int-char 78) ; center box tee (+)
(int-char 224))) ; back slash
(int-char 189) ; right bracket
(int-char 192) ; left brace
(int-char 208) ; right brace
(int-char 78) ; top box tee (+)
(int-char 78) ; bottom box tee (+)
(int-char 78) ; right box tee (+)
(int-char 78) ; left box tee (+)
(int-char 78) ; center box tee (+)
(int-char 224)); ; back slash

defvar $plainRTspecialCharacters
— initvars —

(defvar |$plainRTspecialCharacters| (list
(QUOTE +) ; upper left corner (+)
(QUOTE +) ; upper right corner (+)
(QUOTE +) ; lower left corner (+)
(QUOTE +) ; lower right corner (+)
(QUOTE \|) ; vertical bar
(QUOTE -) ; horizontal bar (-)
(QUOTE ?) ; APL quad (?)
(QUOTE [) ; left bracket
(QUOTE ]) ; right bracket
(QUOTE {) ; left brace
(QUOTE }) ; right brace
(QUOTE +) ; top box tee (+)
(QUOTE +) ; bottom box tee (+)
(QUOTE +) ; right box tee (+)
(QUOTE +) ; left box tee (+)
(QUOTE +) ; center box tee (+)
(QUOTE \|\|))) ; back slash

defvar $RTspecialCharacters
— initvars —

(defvar |$RTspecialCharacters| (list
(intern (string (code-char 218))) ;-- upper left corner (+)
(intern (string (code-char 191))) ;-- upper right corner (+)
(intern (string (code-char 192))) ;-- lower left corner (+)
(intern (string (code-char 217))) ;-- lower right corner (+)
(intern (string (code-char 179))) ;-- vertical bar
(intern (string (code-char 196))) ;-- horizontal bar (-)
(list (code-char #x1d) (code-char #xe2)) ;-- APL quad (?)
(QUOTE []) ;-- left bracket
(QUOTE ]) ;-- right bracket
(QUOTE {{}) ;-- left brace
(QUOTE }}) ;-- right brace
(intern (string (code-char 194))) ;-- top box tee (+)
(intern (string (code-char 193))) ;-- bottom box tee (+)
(intern (string (code-char 180))) ;-- right box tee (+)
(intern (string (code-char 195))) ;-- left box tee (+)
(intern (string (code-char 197))) ;-- center box tee (+)
(QUOTE |\|)) ;-- back slash

defvar $specialCharacters

— initvars —

(defvar |$specialCharacters| |$RTspecialCharacters|)

defvar $specialCharacterAlist

— initvars —

(defvar |$specialCharacterAlist|)

'((|ulc| . 0)
 (|urc| . 1)
 (|llc| . 2)
 (|lrc| . 3)
 (|vbar| . 4)
 (|hbar| . 5)
 (|quad| . 6)
 (|lbrk| . 7)
 (|rbrk| . 8)
 (|lbrc| . 9)
defun Look up a special character code for a symbol

This function looks up a symbol in $specialCharacterAlist, gets the index into the EBCDIC table, and returns the appropriate character. TPDHERE: Make this more international, not EBCDIC [ifcdr p??]

[assq p1050]
[$specialCharacters p979]
[$specialCharacterAlist p979]

— defun specialChar —

(defun specialChar (symbol)
  (let (code)
    (declare (special |$specialCharacters| |$specialCharacterAlist|)))
    (if (setq code (ifcdr (assq symbol |$specialCharacterAlist|)))
      (elt |$specialCharacters| code)
      "?"))

———
Chapter 62

Stream and File Handling

defun make-instream

[makeInputFilename p983]

— defun make-instream —

(defvar make-instream (defmake 输入文件路径 &optional (recnum 0))
 (declare (ignore recnum))
 (cond ((numberp filespec) (make-synonym-stream '*terminal-io*)))
 ((null filespec) (error "not handled yet"))
 (t (open (makeInputFilename spec)
 :direction :input :if-does-not-exist nil))))

———

defun make-outstream

[make-filename p??]

— defun make-outstream —

(defvar make-outstream (defmake 输出文件路径 &optional (width nil) (recnum 0))
 (declare (ignore width) (ignore recnum))
 (cond ((numberp filespec) (make-synonym-stream '*terminal-io*)))
 ((null filespec) (error "not handled yet"))
 (t (open (make-filename spec) :direction :output))))

———
defun make-appendstream

[make-filename ?]

— defun make-appendstream —

(defun make-appendstream (filespec &optional (width nil) (recnum 0))
  "fortran support"
  (declare (ignore width) (ignore recnum))
  (cond
    ((numberp filespec) (make-synonym-stream '*terminal-io*))
    ((null filespec) (error "make-appendstream: not handled yet"))
    ('else (open (make-filename filespec) :direction :output
                 :if-exists :append :if-does-not-exist :create))))

defun defiostream

— defun defiostream —

(defun defiostream (stream-alist buffer-size char-position)
  (declare (ignore buffer-size))
  (let ((mode (or (cdr (assoc 'mode stream-alist)) 'input))
        (filename (cdr (assoc 'file stream-alist)))
        (dev (cdr (assoc 'device stream-alist))))
    (if (eq dev 'console) (make-synonym-stream '*terminal-io*)))
  (let ((strm (case mode
                 ((output o) (open (make-filename filename)
                              :direction :output))
                 ((input i) (open (makeInputFilename filename)
                                  :direction :input)))))
    (if (and (numberp char-position) (> char-position 0))
        (file-position strm char-position)
        strm)))

defun shut

[shut is-console (vol9)]

— defun shut —
(defun shut (st)
  (if (is-console st)
      st
      (if (streamp st) (close st) -1))))

defun eofp

— defun eofp —

(defun eofp (stream) (null (peek-char nil stream nil nil)))

defun makeStream

[make-appendstream p982]
[make-outstream p981]

— defun makeStream —

(defun makeStream (append filename i j)
  (if append
      (make-appendstream filename i j)
      (make-outstream filename i j)))

defun Construct a new input file name

— defun makeInputFilename —

(defun makeInputFilename (filearg &optional (filetype nil))
  (let*
    ((filename (make-filename filearg filetype))
     (dirname (pathname-directory filename))
     (ft (pathname-type filename))
     (dirs (getDirectoryList ft))
     (newfn nil))
    (if (or (null dirname) (eqcar dirname :relative)))
(dolist (dir dirs (probeName filename))
  (when (probe-file (setq newfn (concatenate 'string dir filename)))
    (return newfn))
  (proberName filename))))

---

defun getDirectoryList

[$current-directory p5]
[$UserLevel p807]
[$library-directory-list p7]
[$directory-list p6]

— defun getDirectoryList —

(defun getDirectoryList (ft &aux (cd (namestring $current-directory)))
  (declare (special $current-directory |$UserLevel| $library-directory-list
                     $directory-list))
  (if (member ft '("nrlib" "daase" "exposed") :test #'string=)
      (if (eq |$UserLevel| '|development|)
          (cons cd $library-directory-list)
          $library-directory-list)
      (adjoin cd
               (adjoin (namestring (user-homedir-pathname)) $directory-list
                        :test #'string=)
               :test #'string=)))

---

defun probeName

Sometimes we are given a file and sometimes we are given the name of an Axiom KAF
(Keyed-Access File). KAF files are actually directories with a single file called “index.kaf”. We check for the latter case and return the directory name as the filename, per Axiom convention.

— defun probeName —

(defun probeName (file)
  (when (or (probe-file file)
            (probe-file (concatenate 'string (namestring file) "/index.kaf")))
    (namestring file)))

---
defun makeFullNamestring

— defun makeFullNamestring —

(defun makeFullNamestring (filearg &optional (filetype nil))
  (namestring (merge-pathnames (make-filename filearg filetype))))

-----

defun Replace a file by erase and rename

[makeFullNamestring p985]

— defun replaceFile —

(defun replaceFile (filespec1 filespec2)
  ($erase (setq filespec1 (makeFullNamestring filespec1)))
  (rename-file (makeFullNamestring filespec2) filespec1))

-----
The Spad Server Mechanism

defun openserver

This is a cover function for the C code used for communication interface.

(defun openserver (name)
  (open_server name))
Chapter 64

Axiom Build-time Functions

defun spad-save

The spad-save function is just a cover function for more lisp system specific save functions. There is no standard name for saving a lisp image so we make one and conditionalize it at compile time.

This function is passed the name of an image that will be saved. The saved image contains all of the loaded functions.

This is used in the src/interp/Makefile.pamphlet in three places:

- creating depsys, an image for compiling axiom.
  Some of the Common Lisp code we compile uses macros which are assumed to be available at compile time. The DEPSYS image is created to contain the compile time environment and saved. We pipe compile commands into this environment to compile from Common Lisp to machine dependent code.

  DEPSYS=${OBJ}/${SYS}/bin/depsys

- creating savesys, an image for running axiom.
  Once we’ve compile all of the Common Lisp files we fire up a clean lisp image called LOADSYS, load all of the final executable code and save it out as SAVESYS. The SAVESYS image is copied to the ${MNT}/${SYS}/bin subdirectory and becomes the axiom executable image.

  LOADSYS= ${OBJ}/${SYS}/bin/lisp
  SAVESYS= ${OBJ}/${SYS}/bin/interpsys
  AXIOMSYS= ${MNT}/${SYS}/bin/AXIOMsys

- creating debugsys, an image with all interpreted functions loaded.
Occasionally we need to really get into the system internals. The best way to do this is to run almost all of the lisp code interpreted rather than compiled (note that cfuns.lisp and sockio.lisp still need to be loaded in compiled form as they depend on the loader to link with lisp internals). This image is nothing more than a load of the file src/interp/debugsys.lisp.pamphlet. If you need to make test modifications you can add code to that file and it will show up here.

`DEBUGSYS=${OBJ}/${SYS}/bin/debugsys`

```
[save-system p?]
[$SpadServer p10]
[$openServerIfTrue p8]

— defun spad-save —

(defun user::spad-save (save-file)
  (declare (special |$SpadServer| $openServerIfTrue))
  (setq |$SpadServer| nil)
  (setq $openServerIfTrue t)
  #+:AKCL
  (system::save-system save-file)
  #+:allegro
  (if (fboundp 'boot::restart)
      (excl::dumplisp :name save-file :restart-function #'boot::restart)
      (excl::dumplisp :name save-file))
  #+:Lucid
  (if (fboundp 'boot::restart)
      (sys::disksave save-file :restart-function #'boot::restart)
      (sys::disksave save-file))
  #+:CCL
  (preserve)
)
```
Chapter 65

Exposure Groups

Exposure groups are a way of controlling the namespace available to the user. Certain algebra files are only useful for internal purposes but they contain functions have common names (like “map”. In order to separate the user visible functions from the internal functions the algebra files are collected into “exposure groups”. These large groups are grouped into sets in the variable $\text{globalExposureGroupAlist}$.

Exposure group information is kept in the local frame. For more information “The Frame Mechanism” 32.3 on page 552.
Chapter 66

Databases

66.1 Database structure

In order to understand this program you need to understand some details of the structure of the databases it reads. Axiom has 5 databases, the interp.daase, operation.daase, category.daase, and browse.daase.

kaf File Format

This documentation refers to kaf files which are random access files. nrlib files are kaf files (look for nrlib/index.kaf) The format of a random access file is

```
byte-offset-of-key-table
first-entry
second-entry
...
last-entry
((key1 . first-entry-byte-address)
 (key2 . second-entry-byte-address)
 ...
 (keyN . last-entry-byte-address))
```

The key table is a standard lisp alist.

To open a database you fetch the first number, seek to that location, and (read) which returns the key-data alist. To look up data you index into the key-data alist, find the ith-entry-byte-address, seek to that address, and (read).

For instance, see src/share/algebra/users.daase/index.kaf

One existing optimization is that if the data is a simple thing like a symbol then the nth-entry-byte-address is replaced by immediate data.
Another existing one is a compression algorithm applied to the data so that the very long names don’t take up so much space. We could probably remove the compression algorithm as 64k is no longer considered 'huge'. The database-abbreviation routine handles this on read and write-compress handles this on write.

Indeed, a faster optimization is to simply read the whole database into the image before it is saved. The system would be easier to understand and the interpreter would be faster.

The fastest optimization is to fix the time stamp mechanism which is currently broken. Making this work requires a small bit of coordination at 'make' time which I forgot to implement.

Database Files

Database files are very similar to kaf files except that there is an optimization (currently broken) which makes the first item a pair of two numbers. The first number in the pair is the offset of the key-value table, the second is a time stamp. If the time stamp in the database matches the time stamp in the image the database is not needed (since the internal hash tables already contain all of the information). When the database is built the time stamp is saved in both the gcl image and the database.

Regarding the 'ancestors field in a category: At database build time there exists a *ancestors-hash* hash table that gets filled with CATEGORY (not domain) ancestor information. This later provides the information that goes into interp.daase This *ancestors-hash* does not exist at normal runtime (it can be made by a call to genCategoryTable). Note that the ancestor information in *ancestors-hash* (and hence interp.daase) involves #1, #2, etc instead of R, Coef, etc. The latter thingies appear in all .nrlib/index.kaf files. So we need to be careful when we )lib categories and update the ancestor info.

This file contains the code to build, open and access the .daase files. This file contains the code to )library nrlibs and asy files.

There is a major issue about the data that resides in these databases. the fundamental problem is that the system requires more information to build the databases than it needs to run the interpreter. in particular, modemap.daase is constructed using properties like "modemaps" but the interpreter will never ask for this information.

So, the design is as follows:

- the modemap.daase needs to be built. this is done by doing a )library on ALL of the nrlib files that are going into the system. this will bring in "modemap" information and add it to the *modemaps-hash* hashtable.
- database build proceeds, accessing the "modemap" property from the hashtables. once this completes this information is never used again.
- the interp.daase database is built. this contains only the information necessary to run the interpreter. note that during the running of the interpreter users can extend the system by do a )library on a new nrlib file. this will cause fields such as "modemap" to be read and hashed.
66.1. DATABASE STRUCTURE

Each constructor (e.g. LIST) had one library directory (e.g. LIST.nrlib). This directory contained a random access file called the index.kaf file. These files contain runtime information such as the operationAlist and the ConstructorModemap. At system build time we merge all of these .nrlib/index.kaf files into one database, INTERP.data. Requests to get information from this database are cached so that multiple references do not cause additional disk i/o.

This database is left open at all times as it is used frequently by the interpreter. One minor complication is that newly compiled files need to override information that exists in this database.

The design calls for constructing a random read (kaf format) file that is accessed by functions that cache their results. When the database is opened the list of constructor-index pairs is hashed by constructor name. A request for information about a constructor causes the information to replace the index in the hash table. Since the index is a number and the data is a non-numeric sexpr there is no source of confusion about when the data needs to be read.

The format of this new database is as follows:

- **First entry:**
  - an integer giving the byte offset to the constructor alist at the bottom of the file

- **Second and subsequent entries (one per constructor):**
  - (operationAlist)
  - (constructorModemap)

- **Last entry:** (pointed at by the first entry)
  - an alist of (constructor . index) e.g.
    - (PI offset-of-operationAlist offset-of-constructorModemap)
    - (NNI offset-of-operationAlist offset-of-constructorModemap)
    - ....

This list is read at open time and hashed by the car of each item.

The system has been changed to use the property list of the symbols rather than hash tables. Since we already hashed once to get the symbol we need only an offset to get the property list. This also has the advantage that eq hash tables no longer need to be moved during garbage collection.

There are 3 potential speedups that could be done.

- The best would be to use the value cell of the symbol rather than the property list but I'm unable to determine all uses of the value cell at the present time.

- A second speedup is to guarantee that the property list is a single item, namely the database structure. This removes an assoc but leaves one open to breaking the system if someone adds something to the property list. This was not done because of the danger mentioned.

- A third speedup is to make the getdatabase call go away, either by making it a macro or eliding it entirely. This was not done because we want to keep the flexibility of changing the database forms.
The new design does not use hash tables. The database structure contains an entry for each item that used to be in a hash table. Initially the structure contains file-position pointers and these are replaced by real data when they are first looked up. The database structure is kept on the property list of the constructor, thus, `(get ’DenavitHartenbergMatrix— ’database)` will return the database structure object.

Each operation has a property on its symbol name called ’operation which is a list of all of the signatures of operations with that name.

**defstruct $database**

--- initvars ---

```
(defstruct database
  abbreviation ; interp.
  ancestors ; interp.
  constructor ; interp.
  constructorcategory ; interp.
  constructorkind ; interp.
  constructormodemap ; interp.
  cosig ; interp.
  defaultdomain ; interp.
  modems ; interp.
  niladic ; interp.
  object ; interp.
  operationalist ; interp.
  documentation ; browse.
  constructorform ; browse.
  attributes ; browse.
  predicates ; browse.
  sourcefile ; browse.
  parents ; browse.
  users ; browse.
  dependents ; browse.
  spare ; superstition
) ; database structure
```

**defvar *defaultdomain-list***

There are only a small number of domains that have default domains. Rather than keep this slot in every domain we maintain a list here.

--- initvars ---

```
(defvar *defaultdomain-list* '(
```
66.1. DATABASE STRUCTURE

{(|MultisetAggregate| |Multiset|)
 (|FunctionSpace| |Expression|)
 (|AlgebraicallyClosedFunctionSpace| |Expression|)
 (|ThreeSpaceCategory| |ThreeSpace|)
 (|DequeueAggregate| |Dequeue|)
 (|ComplexCategory| |Complex|)
 (|LazyStreamAggregate| |Stream|)
 (|AssociationListAggregate| |AssociationList|)
 (|QuaternionCategory| |Quaternion|)
 (|PriorityQueueAggregate| |Heap|)
 (|PointCategory| |Point|)
 (|PlottableSpaceCurveCategory| |Plot3D|)
 (|PermutationCategory| |Permutation|)
 (|StringCategory| |String|)
 (|FileNameCategory| |FileName|)
 (|OctonionCategory| |Octonion|))

defvar $*operation-hash*

— initvars —

(ddefvar *operation-hash* nil "given an operation name, what are its modemaps?")

——

defvar $*hasCategory-hash*

This hash table is used to answer the question“does domain x have category y?”. This is answered by constructing a pair of (x . y) and doing an equal hash into this table.

— initvars —

(ddefvar *hasCategory-hash* nil "answers x has y category questions")

——

defvar $*miss*

This variable is used for debugging. If a hash table lookup fails and this variable is non-nil then a message is printed.

— initvars —
(defvar *miss* nil "print out cache misses on getdatabase calls")

Note that constructorcategory information need only be kept for items of type category. this will be fixed in the next iteration when the need for the various caches are reviewed

Note that the *modemaps-hash* information does not need to be kept for system files. these are precomputed and kept in modemap.data however, for user-defined files these are needed. Currently these are added to the database for 2 reasons; there is a still-unresolved issue of user database extensions and this information is used during database build time

Database streams

This are the streams for the databases. They are always open. There is an optimization for speeding up system startup. If the database is opened and the ..-stream-stamp* variable matches the position information in the database then the database is NOT read in and is assumed to match the in-core version

defvar $*interp-stream*

— initvars —

(defvar *interp-stream* nil "an open stream to the interpreter database")

defvar $*interp-stream-stamp*

— initvars —

(defvar *interp-stream-stamp* 0 "*interp-stream* (position . time)"

defvar $*operation-stream*

This is indexed by operation, not constructor

— initvars —
66.1. DATABASE STRUCTURE

(defvar *operation-stream* nil "the stream to operation.daase")

---

defvar *operation-stream-stamp*

— initvars —

(defvar *operation-stream-stamp* 0 "*operation-stream* (position . time)")

---

defvar *browse-stream*

— initvars —

(defvar *browse-stream* nil "an open stream to the browser database")

---

defvar *browse-stream-stamp*

— initvars —

(defvar *browse-stream-stamp* 0 "*browse-stream* (position . time)")

---

defvar *category-stream*

This is indexed by (domain . category)

— initvars —

(defvar *category-stream* nil "an open stream to the category table")

---
defvar $*category-stream-stamp*

— initvars —
(defvar *category-stream-stamp* 0 "*category-stream* (position . time)"

——

defvar $*allconstructors*

— initvars —
(defvar *allconstructors* nil "a list of all the constructors in the system"

——

defvar $*allOperations*

— initvars —
(defvar *allOperations* nil "a list of all the operations in the system"

——

defun Reset all hash tables before saving system

[interpopen p??]  [operationopen p??]
[browseopen p??]  [categoryopen p??]
[initial-getdatabase p1001]
[*sourcefiles* p??]  [*interp-stream* p998]
[*operation-stream* p998]  [*category-stream* p999]
[*browse-stream* p999]  [*category-stream-stamp* p1000]
[*operation-stream-stamp* p999]  [*interp-stream-stamp* p998]
66.1. DATABASE STRUCTURE

["allconstructors" p1000]  
["operation-hash" p997]  
["hascategory-hash" p??]

— defun resethashtables —

(defun resethashtables ()
"set all -hash* to clean values. used to clean up core before saving system"
(declare (special *sourcefiles* *interp-stream* *operation-stream*
 *category-stream* *browse-stream* *category-stream-stamp*
 *operation-stream-stamp* *interp-stream-stamp*
 *allconstructors* *operation-hash* *hascategory-hash*))
(setq *hascategory-hash* (make-hash-table :test #'equal))  
(setq *operation-hash* (make-hash-table))
(setq *allconstructors* nil)
(setq *sourcefiles* nil)
(setq *interp-stream-stamp* '(0 . 0))
(interpopen)
(setq *operation-stream-stamp* '(0 . 0))
(operationopen)
(setq *browse-stream-stamp* '(0 . 0))
(browseopen)
(setq *category-stream-stamp* '(0 . 0))
(categoryopen) ;note: this depends on constructorform in browse.daase
(initial-getdatabase)
(close *interp-stream*)
(close *operation-stream*)
(close *category-stream*)
(close *browse-stream*)
(gbc t))

———

defun Preload algebra into saved system

[getdatabase p1010]  
[getEnv p??]

— defun initial-getdatabase —

(defun initial-getdatabase ()
"fetch data we want in the saved system"
(let (hascategory constructormodemapAndoperationalist operation constr)
(format t "Initial getdatabase"%")
(setq hascategory '(
  (|Equation| . |Ring|)
  (|Expression| . |CoercibleTo|))
  (|Expression| . |CommutativeRing|))
  (|Expression| . |Ring|))
  (|Expression| . |CommutativeRing|))
  (gbc t))
((|Expression| . |IntegralDomain|) (|Expression| . |Ring|))
((|Float| . |RetractableTo|))
((|Fraction| . |Algebra|) (|Fraction| . |CoercibleTo|))
((|Fraction| . |OrderedSet|) (|Fraction| . |RetractableTo|))
((|Integer| . |Algebra|) (|Integer| . |CoercibleTo|))
((|Integer| . |ConvertibleTo|) (|Integer| . |LinearlyExplicitRingOver|))
((|Integer| . |RetractableTo|))
((|List| . |CoercibleTo|) (|List| . |FiniteLinearAggregate|))
((|List| . |OrderedSet|))
((|Polynomial| . |CoercibleTo|) (|Polynomial| . |CommutativeRing|))
((|Polynomial| . |ConvertibleTo|) (|Polynomial| . |OrderedSet|))
((|Polynomial| . |RetractableTo|))
((|Symbol| . |CoercibleTo|) (|Symbol| . |ConvertibleTo|))
((|Variable| . |CoercibleTo|)))
(dolist (pair hascategory) (getdatabase pair 'hascategory))
(setq constructormodemapAndoperationalist '(
  |BasicOperator| |Boolean|
  |CardinalNumber| |Color| |Complex|
  |Database|
  |Equation| |EquationFunctions2| |Expression|
  |Float| |Fraction| |FractionFunctions2|
  |Integer| |IntegralDomain|
  |Kernel|
  |List|
  |Matrix| |MappingPackage1|
  |Operator| |OutputForm|
  |NonNegativeInteger|
  |ParametricPlaneCurve| |ParametricSpaceCurve| |Point| |Polynomial|
  |PolynomialFunctions2| |PositiveInteger|
  |Ring|
  |SetCategory| |SegmentBinding| |SegmentBindingFunctions2| |DoubleFloat|
  |SparseMultivariatePolynomial| |SparseUnivariatePolynomial| |Segment|
  |String| |Symbol|
  |UniversalSegment|
  |Variable| |Vector|))
(dolist (con constructormodemapAndoperationalist)
  (getdatabase con 'constructormodemap)
  (getdatabase con 'operationalist))
(setq operation '(
  |+| |-| |*| |/| |**| |coerce| |convert| |elt| |equation|
  |float| |sin| |cos| |map| |SEGMENT|))
(dolist (op operation) (getdatabase op 'operation))
(setq constr '( ;these are sorted least-to-most freq. delete early ones first
  |Factored| |SparseUnivariatePolynomialFunctions2| |TableAggregate&|
  |RetractableTo&| |RecursiveAggregate&| |UserDefinedPartialOrdering|
  |None| |UnivariatePolynomialCategoryFunctions2| |IntegerPrimesPackage|
  |SetCategory&| |IndexedExponents| |QuotientFieldCategory&| |Polynomial|
  |EltableAggregate&| |PartialDifferentialRing&| |Set|
  |UnivariatePolynomialCategory&| |FlexibleArray|
  |SparseMultivariatePolynomial| |PolynomialCategory&|)
defun Open the interp database

Format of an entry in interp.daase:

(constructor-name
 operationalist
 constructormodemap
 modenames -- this should not be needed. eliminate it.
 object -- the name of the object file to load for this con.
 constructorcategory -- note that this info is the cadar of the
 constructormodemap for domains and packages so it is stored
 as NIL for them. it is valid for categories.
 niladic -- t or nil directly
 unused
 cosig -- kept directly
(defun interpOpen ()
"open the interpreter database and hash the keys"
(declare (special $spadroot *allconstructors* *interp-stream* *interp-stream-stamp*)
(let (constructors pos stamp dbstruct)
  (setq *interp-stream* (open (DaaseName "interp.daase" nil)))
  (setq stamp (read *interp-stream*))
  (unless (equal stamp *interp-stream-stamp*)
    (format t " Re-reading interp.daase")
    (setq *interp-stream-stamp* stamp)
    (setq pos (car stamp))
    (file-position *interp-stream* pos)
    (setq constructors (read *interp-stream*))
  (dolist (item constructors)
    (setq *allconstructors* (adjoin (first item) *allconstructors*))
    (setq dbstruct (make-database))
    (setf (get (car item) 'database) dbstruct)
    (setf (database-operationalist dbstruct) (second item))
    (setf (database-constructormodemap dbstruct) (third item))
    (setf (database-modemaps dbstruct) (fourth item))
    (setf (database-object dbstruct) (fifth item))
    (setf (database-constructorcategory dbstruct) (sixth item))
    (setf (database-niladic dbstruct) (seventh item))
    (setf (database-abbreviation dbstruct) (eighth item))
    (setf (get (eighth item) 'abbreviationfor) (first item)) ;invert
    (setf (database-cosig dbstruct) (ninth item))
    (setf (database-constructorkind dbstruct) (tenth item))
    (setf (database-ancestors dbstruct) (nth 11 item)))
  (format t "&")))  

This is an initialization function for the constructor database it sets up 2 hash tables, opens the database and hashes the index values.
66.1. DATABASE STRUCTURE

There is a slight asymmetry in this code. The sourcefile information for system files is only the filename and extension. For user files it contains the full pathname. When the database is first opened the sourcefile slot contains system names. The lookup function has to prefix the "$spadroot" information if the directory-namestring is null (we don’t know the real root at database build time).

An object-hash table is set up to look up urlib and ao information. This slot is empty until a user does a library call. We remember the location of the urlib or ao file for the user's local library at that time. A NIL result from this probe means that the library is in the system-specified place. When we get into multiple library locations this will also contain system files.

defun Open the browse database

Format of an entry in browse.daase:

( constructorname
  sourcefile
  constructorform
  documentation
  attributes
  predicates
)

[$spadroot p9]
[*allconstructors* p1000]
[*browse-stream* p999]
[*browse-stream-stamp* p999]

— defun browseOpen —

(defun browseOpen ()
  "open the constructor database and hash the keys"
  (declare (special $spadroot *allconstructors* *browse-stream*
    *browse-stream-stamp*))
  (let (constructors pos stamp dbstruct)
    (setq *browse-stream* (open (DaaseName "browse.daase" nil)))
    (setq stamp (read *browse-stream*))
    (unless (equal stamp *browse-stream-stamp*)
      (format t " Re-reading browse.daase")
      (setq *browse-stream-stamp* stamp)
      (setq pos (car stamp))
      (file-position *browse-stream* pos)
      (setq constructors (read *browse-stream*))
    (dolist (item constructors)
      (unless (setq dbstruct (get (car item) 'database))
        (format t "browseOpen:~%")
        (format t "the browse database contains a constructor "~a" item")
      )
    )
  )
(format t "that is not in the interp.daase file. we cannot")
(format t "get the database structure for this constructor and")
(warn "will create a new one")
(setf (get (car item) 'database) (setq dbstruct (make-database)))
(setq *allconstructors* (adjoin item *allconstructors*))
(setf (database-sourcefile dbstruct) (second item))
(setf (database-constructorm form dbstruct) (third item))
(setf (database-documentation dbstruct) (fourth item))
(setf (database-attributes dbstruct) (fifth item))
(setf (database-predicates dbstruct) (sixth item))
(setf (database-parents dbstruct) (seventh item))))
(format t "-&")

---

defun Open the category database

[$spadroot p9]
[*hasCategory-hash* p997]
[*category-stream* p999]
[*category-stream-stamp* p1000]
--- defun categoryOpen ---

(defun categoryOpen ()
  "open category.daase and hash the keys"
  (declare (special $spadroot *hasCategory-hash* *category-stream* *
    *category-stream-stamp*))
  (let (pos keys stamp)
    (setq *category-stream* (open (DaaseName "category.daase" nil)))
    (setq stamp (read *category-stream*))
    (unless (equal stamp *category-stream-stamp*)
      (format t " Re-reading category.daase")
      (setq *category-stream-stamp* stamp)
    (setq pos (car stamp))
    (file-position *category-stream* pos)
    (setq keys (read *category-stream*))
    (setq *hasCategory-hash* (make-hash-table :test #'equal))
    (dolist (item keys)
      (setf (gethash (first item) *hasCategory-hash*) (second item)))
    (format t "-&")
  )
---
defun Open the operations database

(defun operationOpen ()
  "read operation database and hash the keys"
  (declare (special $spadroot *operation-hash* *operation-stream*
                       *operation-stream-stamp*))
  (let (operations pos stamp)
    (setq *operation-stream* (open (DaaseName "operation.daase" nil)))
    (setq stamp (read *operation-stream*))
    (unless (equal stamp *operation-stream-stamp*)
      (format t " Re-reading operation.daase")
      (setq *operation-stream-stamp* stamp)
      (setq pos (car stamp))
      (file-position *operation-stream* pos)
      (setq operations (read *operation-stream*))
      (dolist (item operations)
         (setf (gethash (car item) *operation-hash*) (cdr item))))
    (format t "&")
    )

---

defun Add operations from newly compiled code

(defun addoperations (constructor oldmaps)
  "add ops from a )library domain to *operation-hash*
   (declare *operation-hash*))
  (dolist (map oldmaps) ; out with the old
    (let (oldop op)
      (setq op (car map))
      (setq oldop (getdatabase op 'operation))
      (setq oldop (lisp::delete (cdr map) oldop :test #'equal))
      (setf (gethash op *operation-hash*) oldop))
    (dolist (map (getdatabase constructor 'modemaps)) ; in with the new
      (let (op newmap)
        (setq op (car map))
        (setq newmap (gethash op *operation-hash*))
        (setq newmap (lisp::delete (cdr map) newmap :test #'equal))
        (setf (gethash op newmap))
        )
      )
(setq newmap (getdatabase op 'operation))
(setf (gethash op *operation-hash*) (cons (cdr map) newmap)))

defun Show all database attributes of a constructor

[getdatabase p1010]

defun showdatabase —

(defun showdatabase (constructor)
  (format t "\%a: \%a\n" 'constructorkind
          (getdatabase constructor 'constructorkind))
  (format t "\%a: \%a\n" 'cosig
          (getdatabase constructor 'cosig))
  (format t "\%a: \%a\n" 'operation
          (getdatabase constructor 'operation))
  (format t "\%a: \%a\n" 'constructormodemap
          (pprint (getdatabase constructor 'constructormodemap))
  (format t "\%a: \%a\n" 'constructorcategory
          (pprint (getdatabase constructor 'constructorcategory))
  (format t "\%a: \%a\n" 'operationalist
          (pprint (getdatabase constructor 'operationalist))
  (format t "\%a: \%a\n" 'modemaps
          (pprint (getdatabase constructor 'modemaps))
  (format t "\%a: \%a\n" 'hascategory
          (getdatabase constructor 'hascategory))
  (format t "\%a: \%a\n" 'object
          (getdatabase constructor 'object))
  (format t "\%a: \%a\n" 'niladic
          (getdatabase constructor 'niladic))
  (format t "\%a: \%a\n" 'abbreviation
          (getdatabase constructor 'abbreviation))
  (format t "\%a: \%a\n" 'constructor?
          (getdatabase constructor 'constructor?))
  (format t "\%a: \%a\n" 'constructor
          (getdatabase constructor 'constructor))
  (format t "\%a: \%a\n" 'defaultdomain
          (getdatabase constructor 'defaultdomain))
  (format t "\%a: \%a\n" 'ancestors
          (getdatabase constructor 'ancestors))
  (format t "\%a: \%a\n" 'sourcefile
          (getdatabase constructor 'sourcefile))
  (format t "\%a: \%a\n" 'constructorform
          (getdatabase constructor 'constructorform))
  (format t "\%a: \%a\n" 'constructorargs
          (getdatabase constructor 'constructorargs))
  (format t "\%a: \%a\n" 'constructorform
          (getdatabase constructor 'constructorform))
  (format t "\%a: \%a\n" 'constructorargs
          (getdatabase constructor 'constructorargs))
defun Set a value for a constructor key in the database

[make-database p??]

— defun setdatabase —

(defun setdatabase (constructor key value)
  (let (struct)
    (when (symbolp constructor)
      (unless (setq struct (get constructor 'database))
        (setq struct (make-database))
      (case key
        (abbreviation
          (setf (database-abbreviation struct) value))
        (when (symbolp value)
          (setf (get value 'abbreviationfor) constructor))
        (constructorkind
          (setf (database-constructorkind struct) value))))))

defun Delete a value for a constructor key in the database

— defun deldatabase —

(defun deldatabase (constructor key)
  (when (symbolp constructor)
    (case key
      (abbreviation
        (setf (get constructor 'abbreviationfor) nil)))))
defun Get constructor information for a database key

| defun getdatabase |
| (defun getdatabase (constructor key) |
| (declare (special $spadroot) (special *miss*)) |
| (when (eq *miss* t) (format t "getdatabase call: ~20a ~a~%" constructor key)) |
| (let (data table stream ignore struct) |
| (declare (ignore ignore) |
| (special *hascategory-hash* *operation-hash* |
| *browse-stream* *defaultdomain-list* *interp-stream* |
| *category-stream* *hasCategory-hash* *operation-stream*)) |
| (when (or (symbolp constructor) |
| (and (eq key 'hascategory) (consp constructor))) |
| (case key |
| (abbreviation (setq stream *interp-stream*) |
| (when (setq struct (get constructor 'database)) |
| (setq data (database-abbreviation struct)))) |
| (constructorkind (setq stream *interp-stream*) |
| (when (setq struct (get constructor 'database)) |
| (setq data (database-constructorkind struct)))) |
| (cosig (setq stream *interp-stream*) |
| (when (setq struct (get constructor 'database)) |
| (setq data (database-cosig struct)))) |
| (operation (setq stream *operation-stream*) |
| (setq data (gethash constructor *operation-hash*))) |
| (constructormodemap (setq stream *interp-stream*) |
| (when (setq struct (get constructor 'database)) |
| (setq data (database-constructormodemap struct)))) |
| (constructorcategory |
(setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-constructorcategory struct))
    (when (null data) ;domain or package then subfield of constructor-modemap
      (setq data (cadar (getdatabase constructor 'constructor-modemap))))))

(operationalist
  (setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-operationalist struct))))

(modemaps
  (setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-modemaps struct))))

(hascategory
  (setq table *hasCategory-hash*)
  (setq stream *category-stream*)
  (setq data (gethash constructor table)))

(object
  (setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-object struct))))

(asharp?
  (setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-object struct))))

(niladic
  (setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-niladic struct))))

(constructor?
  (when (setq data (get constructor 'abbreviationfor)))
  (defaultdomain
    (setq data (cadr (assoc constructor *defaultdomain-list*))))

(ancestors
  (setq stream *interp-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-ancestors struct))))

(sourcefile
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-sourcefile struct))))
(constructorform
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-constructorform struct)))))
(constructorargs
 (setq data (cdr (getdatabase constructor 'constructorform))))
(attributes
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-attributes struct)))))
(predicates
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-predicates struct)))))
(documentation
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-documentation struct)))))
(parents
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-parents struct)))))
(users
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-users struct)))))
(dependents
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-dependents struct)))))
(otherwise (warn "%(GETDATABASE ~a ~a) failed~%" constructor key)))
(when (numberp data) ; fetch the real data
 (when *miss* (format t "getdatabase miss: ~20a ~a~%" constructor key))
(file-position stream data)
(setq data (read stream))
(case key ; cache the result of the database read
   operation (setf (gethash constructor *operation-hash*) data))
   hascategory (setf (gethash constructor *hascategory-hash*) data))
   constructorkind (setf (database-constructorkind struct) data))
   cosig (setf (database-cosig struct) data))
   constructormodemap (setf (database-constructormodemap struct) data))
   constructorcATEGORY (setf (database-constructorcATEGORY struct) data))
   operationalist (setf (database-operationalist struct) data))
   modemaps (setf (database-modemaps struct) data))
   object (setf (database-object struct) data))
   niladic (setf (database-niladic struct) data))
   abbreviation (setf (database-abbreviation struct) data))
   constructor (setf (database-constructor struct) data))
   ancestors (setf (database-ancestors struct) data))
   constructorform (setf (database-constructorform struct) data))
66.1. DATABASE STRUCTURE

(attributes (setf (database-attributes struct) data))
(predicates (setf (database-predicates struct) data))
(documentation (setf (database-documentation struct) data))
(parents (setf (database-parents struct) data))
(users (setf (database-users struct) data))
(dependents (setf (database-dependents struct) data))
(sourcefile (setf (database-sourcefile struct) data)))

(case key ; fixup the special cases
(sourcefile
  (when (and data (string= (directory-namestring data) "")
    (string= (pathname-type data) "spad"))
    (setq data
      (concatenate 'string $spadroot "/../src/algebra/" data))))
(asharp? ; is this ashad code?
  (if (consp data)
    (setq data (cdr data))
    (setq data nil)))
(object ; fix up system object pathname
  (if (consp data)
    (setq data
      (if (string= (directory-namestring (car data)) "")
        (concatenate 'string $spadroot "/algebra/" (car data) ".o")
        (car data))
      (when (and data (string= (directory-namestring data) ""))
        (setq data (concatenate 'string $spadroot "/algebra/" data ".o")))))
)

---

defun The }library top level command

[localdatabase p1014]
[extendLocalLibdb p??]
tersyscommand p452
$newConlist p??
$options p??

— defun library —

(defun |library| (args)
  (let (original-directory)
    (declare (special |$options| |$newConlist|))
    (setq original-directory (get-current-directory))
    (setq |$newConlist| nil)
    (localdatabase args |$options|)
    (extendLocalLibdb |$newConlist|)
    (system::chdir original-directory)
defun Read a local filename and update the hash tables

The localdatabase function tries to find files in the order of:

- nrlib/index.kaf
- .asy
- .ao,
- asharp to .asy

---

(defun localdatabase (filelist options &optional (make-database? nil))
"read a local filename and update the hash tables"
(labels (
  (processOptions (options)
    (let (only dir noexpose)
      (when (setq only (assoc '|only| options))
        (setq options (lisp::delete only options :test #'equal))
        (setq only (cdr only)))
      (when (setq dir (assoc '|dir| options))
        (setq options (lisp::delete dir options :test #'equal))
        (setq dir (second dir))
      (when (null dir)
        (|sayKeyedMsg| 'S2IU0002 nil))
      (when (setq noexpose (assoc '|noexpose| options))
        (setq options (lisp::delete noexpose options :test #'equal)))
      (when options
        (format t " Ignoring unknown )library option: ~a~%" options))
      (values only dir noexpose)))
  (processDir (dirarg thisdir)
    (let (allfiles)
      (declare (special vmlisp::*index-filename*))
      (system::chdir (string dirarg)))
defun Update the database from an nrlib index.kaf file

[getdatabase p1010]
[make-database p??]
[addoperations p1007]
[sublis p??]
[updateDatabase p1017]
[installConstructor p??]
[updateCategoryTable p??]
[categoryForm? p??]
[setExposeAddConstr p700]
[startTimingProcess p??]
[loadLibNoUpdate p1037]
[sayKeyedMsg p329]
--- defun localnrlib ---

(defun localnrlib (key nrlib object make-database? noexpose)
  "given a string pathname of an index.kaf and the object update the database"
  (labels (
    (fetchdata (alist in index)
      (let (pos)
        (setq pos (third (assoc index alist :test #'string=)))
        (when pos
          (file-position in pos)
          (read in))))
    (let (alist kind (systemdir? nil) pos constructorform oldmaps abbrev dbstruct)
      (declare (special *allOperations* *allconstructors* |$FormalMapVariableList|))
      (with-open-file (in nrlib)
        (file-position in (read in))
        (setq alist (read in))
        (setq pos (third (assoc "constructorForm" alist :test #'string=)))
        (file-position in pos)
        (setq constructorform (read in))
        (setq key (car constructorform))
        (setq oldmaps (getdatabase key 'modemaps))
        (setq dbstruct (make-database))
        (setq *allconstructors* (adjoin key *allconstructors*))
        (setf (get key 'database) dbstruct) ; store the struct, side-effect it...
        (setf (database-constructorform dbstruct) constructorform)
        (setq *allOperations* nil) ; force this to recompute
        (setf (database-object dbstruct) object)
        (setq abbrev
          (intern (pathname-name (first (last (pathname-directory object)))))))
      (setf (database-abbreviation dbstruct) abbrev)
      (setf (get abbrev 'abbreviationfor) key)
      (setf (database-operationalist dbstruct) nil)
      (setf (database-operationalist dbstruct) (fetchdata alist in "operationAlist"))
      (setf (database-constructormodemap dbstruct)
        (fetchdata alist in "constructorModemap"))
      (setf (database-modemaps dbstruct) (fetchdata alist in "modemaps"))
      (setf (database-sourcefile dbstruct) (fetchdata alist in "sourceFile"))
      (when make-database?
        (setf (database-sourcefile dbstruct)
          (file-namestring (database-sourcefile dbstruct)))
        (setf (database-constructorkind dbstruct)
          (setq kind (fetchdata alist in "constructorKind")))
        (setf (database-constructorcategory dbstruct)
          (setq kind (fetchdata alist in "constructorCategory"))))
    (labels (fetchdata alist in index)))
  (labels (
    (fetchdata (alist in index)
      (let (pos)
        (setq pos (third (assoc index alist :test #'string=)))
        (when pos
          (file-position in pos)
          (read in)))))
  (let (alist kind (systemdir? nil) pos constructorform oldmaps abbrev dbstruct)
    (declare (special *allOperations* *allconstructors* |$FormalMapVariableList|))
    (with-open-file (in nrlib)
      (file-position in (read in))
      (setq alist (read in))
      (setq pos (third (assoc "constructorForm" alist :test #'string=)))
      (file-position in pos)
      (setq constructorform (read in))
      (setq key (car constructorform))
      (setq oldmaps (getdatabase key 'modemaps))
      (setq dbstruct (make-database))
      (setq *allconstructors* (adjoin key *allconstructors*))
      (setf (get key 'database) dbstruct) ; store the struct, side-effect it...
      (setf (database-constructorform dbstruct) constructorform)
      (setq *allOperations* nil) ; force this to recompute
      (setf (database-object dbstruct) object)
      (setq abbrev
        (intern (pathname-name (first (last (pathname-directory object)))))))
      (setf (database-abbreviation dbstruct) abbrev)
      (setf (get abbrev 'abbreviationfor) key)
      (setf (database-operationalist dbstruct) nil)
      (setf (database-operationalist dbstruct) (fetchdata alist in "operationAlist"))
      (setf (database-constructormodemap dbstruct)
        (fetchdata alist in "constructorModemap"))
      (setf (database-modemaps dbstruct) (fetchdata alist in "modemaps"))
      (setf (database-sourcefile dbstruct) (fetchdata alist in "sourceFile"))
      (when make-database?
        (setf (database-sourcefile dbstruct)
          (file-namestring (database-sourcefile dbstruct)))
        (setf (database-constructorkind dbstruct)
          (setq kind (fetchdata alist in "constructorKind")))
        (setf (database-constructorcategory dbstruct)
          (setq kind (fetchdata alist in "constructorCategory"))))
    (labels (fetchdata alist in index)
      (let (pos)
        (setq pos (third (assoc index alist :test #'string=)))
        (when pos
          (file-position in pos)
          (read in)))))
(fetchdata alist in "constructorCategory"))
(setf (database-documentation dbstruct)
  (fetchdata alist in "documentation"))
(setf (database-attributes dbstruct)
  (fetchdata alist in "attributes"))
(setf (database-predicates dbstruct)
  (fetchdata alist in "predicates"))
(setf (database-niladic dbstruct)
  (when (fetchdata alist in "NILADIC") t))
(addoperations key oldmaps)
(unless make-database?
  (if (eq kind '|category|)
    (setf (database-ancestors dbstruct)
      (sublislis |$FormalMapVariableList|
       (cdr constructorform) (fetchdata alist in "ancestors"))))))
(updateDatabase| key key systemdirp) ; makes many hashtables???
(installConstructor| key kind) ; used to be key cname ...
(updateCategoryTable| key kind)
(if |$InteractiveMode| (setq |$CategoryFrame| |$EmptyEnvironment|))
(setf (database-cosig dbstruct)
  (cons nil (mapcar #'|categoryForm?|
   (cddar (database-constructormodemap dbstruct))))))
(remprop key 'loaded)
(if (null noexpose) ([setExposeAddConstr| (cons key nil)])
  (setf (symbol-function key) ; sets the autoload property for cname 
    #'(lambda (&rest args)
      (unless (get key 'loaded)
        ([startTimingProcess] 'load)
                      ([loadLibNoUpdate] key key object)); used to be key cname key
                      (apply key args)))
                  ([sayKeyedMsg] 'S2IU0001 (list key object))))))))

defun updateDatabase

For now in NRUNTIME do database update only if forced [constructor? p??]
[clearClams p??]
[clearAllSlams p??]
[$forceDatabaseUpdate p??]

defun updateDatabase —

(defun |updateDatabase| (fname cname systemdirp)
  (declare (ignore fname))
  (declare (special |$forceDatabaseUpdate|))
  (when |$forceDatabaseUpdate|
    — defun updateDatabase —
    ...
defun Make new databases

Making new databases consists of:

1. reset all of the system hash tables
2. set up Union, Record and Mapping
3. map )library across all of the system files (fills the databases)
4. loading some normally autoloaded files
5. making some database entries that are computed (like ancestors)
6. writing out the databases
7. write out 'warm' data to be loaded into the image at build time

Note that this process should be done in a clean image followed by a rebuild of the system image to include the new index pointers (e.g. *interp-stream-stamp*)

The system will work without a rebuild but it needs to re-read the databases on startup. Rebuilding the system will cache the information into the image and the databases are opened but not read, saving considerable startup time. Also note that the order the databases are written out is critical. The interp.daase depends on prior computations and has to be written out last.

The build-name-to-pamphlet-hash builds a hash table whose key-value is:

- abbreviation - pamphlet file name
- abbreviation-line - pamphlet file position
- constructor - pamphlet file name
- constructor-line - pamphlet file position
is the symbol of the constructor name and whose value is the name of the source file without any path information. We hash the constructor abbreviation to pamphlet file name. [local-database p1014]

```lisp
(defun make-databases (ext dirlist)
  (labels ((build-name-to-pamphlet-hash (dir)
               (let ((ht (make-hash-table)) (eof '(done)) point mark abbrev name file ns)
                 (dolist (fn (directory dir))
                   (with-open-file (f fn)
                     (do ((ln (read-line f nil eof) (read-line f nil eof))
                          (line 0 (incf line)))
                         ((eq ln eof))
                       ((eq ln eof))
                         (when (and (setq mark (search "\)abb" ln)) (= mark 0))
                         (setq mark (position "\space" ln :from-end t))
                         (setq name (intern (string-trim "\space" (subseq ln mark))))
                         (cond ((setq mark (search "domain" ln)) (setq mark (+ mark 7)))
                               ((setq mark (search "package" ln)) (setq mark (+ mark 8)))
                               ((setq mark (search "category" ln)) (setq mark (+ mark 9))))
                         (setq point (position "\space" ln :start (+ mark 1)))
                         (setq abbrev (intern (string-trim "\space" (subseq ln mark point)))))))
```

— defun make-databases —
(setq ns (namestring fn))
(setq mark (position "/" ns :from-end t))
(setq file (subseq ns (+ mark 1)))
(setq (gethash abbrev ht) file)
(setq (gethash (format nil ""-a-line" abbrev) ht) line)
(setq (gethash name ht) file)
(setq (gethash (format nil ""-a-line" name) ht) line)))

;; these are types which have no library object associated with them.
;; we store some constructed data to make them perform like library
;; objects, the *operationalist-hash* key entry is used by allConstructors
(withSpecialConstructors ()
  (declare (special *allconstructors*))
  ; note: if item is not in *operationalist-hash* it will not be written
  ; Category
  (setf (get '|Category| 'database)
    (make-database :operationalist nil :niladic t))
  (push '|Category| *allconstructors*)
  ; UNION
  (setf (get '|Union| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Union| *allconstructors*)
  ; RECORD
  (setf (get '|Record| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Record| *allconstructors*)
  ; MAPPING
  (setf (get '|Mapping| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Mapping| *allconstructors*)
  ; ENUMERATION
  (setf (get '|Enumeration| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Enumeration| *allconstructors*)
)
(final-name (root)
  (format nil "".daase"-a" root ext)))
)
(let (d)
  (declare (special |$constructorList| *sourcefiles*
    *allconstructors* *operation-hash*))
  (do-symbols (symbol)
    (when (get symbol 'database)
      (setf (get symbol 'database) nil)))
  (setq *hascategory-hash* (make-hash-table :test #'equal))
  (setq *operation-hash* (make-hash-table))
  (setq *allconstructors* nil)
  (withSpecialConstructors)
  (localdatabase nil
    (list (list '|dir| (namestring (truename "/"))))))
'make-database)
(dolist (dir dirlist)
  (localdatabase nil
    (list (list '|dir| (namestring (truename (format nil "./a" dir)))))))
'make-database))

;browse.daase
(load (concatenate 'string (|getEnv| "AXIOM") "/autoload/topics")) ;; hack
(|browserAutoloadOnceTrigger|)
(|mkTopicHashTable|)
(setq $constructorList nil) ;; affects buildLibdb
(setq *sourcefiles* (build-name-to-pamphlet-hash
  (concatenate 'string (|getEnv| "AXIOM") "/.../src/algebra/*.pamphlet"))
  (|buildLibdb|)
(|dbSplitLibdb|)

;; mkConstructorDataTable
(|mkUsersHashTable|)
(|saveUsersHashTable|)
(|mkDependentsHashTable|)
(|saveDependentsHashTable|)

; (|buildGloss|)
(write-browsedb)
(write-operationdb)

; note: genCategoryTable creates a new *hascategory-hash* table
; this smashes the existing table and regenerates it.
; write-categorydb does getdatabase calls to write the new information
(write-categorydb)
(dolist (con (|allConstructors|))
  (let (dbstruct)
    (when (setq dbstruct (get con 'database))
      (setf (database-cosig dbstruct)
        (cons nil (mapcar #'|categoryForm?|
          (cddar (database-constructormodemap dbstruct))))))
    (when (and (|categoryForm?| con)
        (= (length (setq d (|domainsOf| (list con) NIL NIL))) 1))
      (setq d (caar d))
    (when (= (length d) (length (|getConstructorForm| con)))
      (format t "a has a default domain of "a"%" con (car d))
      (setf (database-defaultdomain dbstruct) (car d)))))))
    ; note: genCategoryTable creates *ancestors-hash*. write-interpdb
    ; does gethash calls into it rather than doing a getdatabase call.
(write-interpdb)
(write-warmdata)

; when (probe-file (final-name "interp")
  (delete-file (final-name "interp")))
(rename-file "interp.build" (final-name "interp"))

; when (probe-file (final-name "operation")
  (delete-file (final-name "operation")))
(rename-file "operation.build" (final-name "operation"))

; when (probe-file (final-name "browse")

(delete-file (final-name "browse"))
(rename-file "browse.build"
  (final-name "browse"))
(when (probe-file (final-name "category"))
  (delete-file (final-name "category"))
  (rename-file "category.build"
    (final-name "category"))))

;;

(defun saveDependentsHashTable ()
  (let (stream)
    (declare (special $depTb $erase))
    ($erase 'dependents 'database 'a)
    (setq stream (|writeLib1| 'dependents 'database 'a))
    (dolist (k (msort (hkeys $depTb)))
      (|rwrite| k (hget $depTb k) stream))
    (rshut stream)))

;;

(defun saveUsersHashTable ()
  (erase p)
  (writeLib1 p)
  (msort p)
  (hkeys p1044)
  (rwrite p604)
  (hget p1044)
  (rshut p)
  ($depTb p)
  ($erase p))
66.1. DATABASE STRUCTURE

(defun saveUsersHashTable ()
  (let (stream)
    (declare (special $usersTb $erase))
    ($erase 'users 'database '|a|)
    (setq stream (|writeLib1| '|users| 'database '|a|))
    (dolist (k (msort (hkeys $usersTb)))
      (|rwrite| k (HGET $usersTb k) stream))
    (rshut stream)))

(defun Construct the proper database full pathname

(defun DaaseName (name erase?)
  (let (daase filename)
    (declare (special $spadroot))
    (if (setq daase (|getEnv| "DAASE"))
      (progn
        (setq filename (concatenate 'string daase "/algebra/" name))
        (format t " Using local database ~a..", filename))
      (setq filename (concatenate 'string $spadroot "/algebra/" name)))
    (when erase? (system::system (concatenate 'string "rm -f " filename))))

Building the interp.daase from hash tables

format of an entry in interp.daase:
(constructor-name
  operationalist
  constructormodemap
  modemaps -- this should not be needed. eliminate it.
  object -- the name of the object file to load for this con.
  constructorcategory -- note that this info is the cadar of the
CHAPTER 66. DATABASES

constructor-modemap for domains and packages so it is stored as NIL for them. It is valid for categories.
niladic -- t or nil directly
unused
cosig -- kept directly
constructor-kind -- kept directly
default-domain -- a short list, for %i
ancestors -- used to compute new category updates

Here I'll try to outline the interp database write procedure

(defun write-interpdb ()
  "build interp.daase from hash tables"
  (declare (special $spadroot *ancestors-hash*))
  (let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
    concategory categorypos kind niladic cosig abbrev default-domain
    ancestors ancestorspos out)
    (declare (special *print-pretty*))
    (print "building interp.daase")
    (setq out (open "interp.build" :direction :output))
    (princ " " out)
    (finish-output out)
    (dolist (constructor (|all-constructors|))
      (let (struct)
        (setq struct (get constructor 'database))
        (setq opalistpos (file-position out)))
      ; 4a. Each constructor has a property list. A property list is a list
      ; of (key . value) pairs. The property we want is called 'database
      ; so there is a ('database . something) in the property list
      ; 5 We write the "operationsalist"
      ; 5a. We remember the current file position before we write
      ; We need this information so we can seek to this position on read
      (setq opalistpos (file-position out)))

; 1. We open the file we're going to create

; 2. We reserve some space at the top of the file for the key-time pair
; We will overwrite these spaces just before we close the file.

; 3. Make sure we write it out

; 4. For every constructor in the system we write the parts:
; 5b. We get the "operationalist" and write it out
     (print (database-operationalist struct) out)
; 5c. We make sure it was written
     (finish-output out)
; 6 We write the "constructormodemap"
; 6a. We remember the current file position before we write
     (setq cmodemappos (file-position out))
; 6b. We get the "constructormodemap" and write it out
     (print (database-constructormodemap struct) out)
; 6c. We make sure it was written
     (finish-output out)
; 7. We write the "modemaps"
; 7a. We remember the current file position before we write
     (setq modemapspos (file-position out))
; 7b. We get the "modemaps" and write it out
     (print (database-modemaps struct) out)
; 7c. We make sure it was written
     (finish-output out)
; 8. We remember source file pathnames in the obj variable
     (if (consp (database-object struct)) ; if asharpc code ...
         (setq obj
             (cons (pathname-name (car (database-object struct)))
                   (cadr (database-object struct))))
         (setq obj
               (pathname-name
                (first (last (pathname-directories (database-object struct)))))))
; 9. We write the "constructorcategory", if it is a category, else nil
; 9a. Get the constructorcategory
     (setq concategory (database-constructorcategory struct))
; 9b. If we have any data we write it out, else we don’t write it
Note that if there is no data then the byte index for the constructorcategory will not be a number but will be nil.

(if concategory ; if category then write data else write nil
 (progn
 (setq categorypos (file-position out))
 (print concategory out)
 (finish-output out))
 (setq categorypos nil))

; 10. We get a set of properties which are kept as "immediate" data
; This means that the key table will hold this data directly
; rather than as a byte index into the file.
; 10a. niladic data

(setq niladic (database-niladic struct))

; 10b. abbreviation data (e.g. POLY for polynomial)

(setq abbrev (database-abbreviation struct))

; 10c. cosig data

(setq cosig (database-cosig struct))

; 10d. kind data

(setq kind (database-constructorkind struct))

; 10e. defaultdomain data

(setq defaultdomain (database-defaultdomain struct))

; 11. The ancestor data might exist. If it does we fetch it
; and write it out. If it does not we place
; and immediate value of nil in the key-value table

(setq ancestors (gethash constructor *ancestors-hash*)) ;cattable.boot
(if ancestors
 (progn
 (setq ancestorspos (file-position out))
 (print ancestors out)
 (finish-output out))
 (setq ancestorspos nil))

; 12. "master" is an alist. Each element of the alist has the name of
; the constructor and all of the above attributes. When the loop
; finishes we will have constructed all of the data for the key-value
; table
(push (list constructor opalistpos cmodemappos modemapspos
   obj categorypos niladic abbrev cosig kind defaultdomain
   ancestorspos) master)))

; 13. The loop is done, we make sure all of the data is written
(finish-output out)

; 14. We remember where the key-value table will be written in the file
(setq masterpos (file-position out))

; 15. We print the key-value table
(print master out)

; 16. We make sure we write the table
(finish-output out)

; 17. We go to the top of the file
(file-position out 0)

; 18. We write out the (master-byte-position . universal-time) pair
; Note that if the universal-time value matches the value of
; *interp-stream-stamp* then there is no reason to read the
; interp database because all of the data is already cached in
; the image. This happens if you build a database and immediately
; save the image. The saved image already has the data since we
; just wrote it out. If the *interp-stream-stamp* and the database
; time stamp differ we "reread" the database on startup. Actually
; we just open the database and fetch as needed. You can see fetches
; by setting the *miss* variable non-nil.
(print (cons masterpos (get-universal-time)) out)

; 19. We make sure we write it.
(finish-output out)

; 20 And we are done
(close out))

defun Write the interp database

[$spadroot p9]
[*ancestors-hash* p??]
(defun write-interpdb ()
  "build interp.daase from hash tables"
  (declare (special $spadroot *ancestors-hash*))
  (let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty* concategory categorypos kind niladic cosig abbrev defaultdomain ancestors ancestorspos out)
    (declare (special *print-pretty*))
    (print "building interp.daase")
    (setq out (open "interp.build" :direction :output))
    (princ " " out)
    (finish-output out)
    (dolist (constructor (|allConstructors|))
      (let (struct)
        (setq struct (get constructor 'database))
        (setq opalistpos (file-position out))
        (print (database-operationalist struct) out)
        (finish-output out)
        (setq cmodemappos (file-position out))
        (print (database-constructormodemap struct) out)
        (finish-output out)
        (setq modemapspos (file-position out))
        (print (database-modemaps struct) out)
        (finish-output out)
        (if (consp (database-object struct)) ; if asharp code ...
            (setq obj
              (cons (pathname-name (car (database-object struct)))
                (cdr (database-object struct))))
            (setq obj
              (pathname-name
                (first (last (pathname-directory (database-object struct))))))
        (setq concategory (database-constructorcategory struct))
        (if concategory ; if category then write data else write nil
            (progn
              (setq categorypos (file-position out))
              (print concategory out)
              (finish-output out))
            (setq categorypos nil))
        (setq niladic (database-niladic struct))
        (setq abbrev (database-abbreviation struct))
        (setq cosig (database-cosig struct))
        (setq kind (database-constructorkind struct))
        (setq defaultdomain (database-defaultdomain struct))
        (setq ancestors (gethash constructor *ancestors-hash*)) ;cattable.boot
        (if ancestors
            (progn
              (setq ancestorspos (file-position out)))
            nil))
    (finish-output out))
Building the browse.daase from hash tables

format of an entry in browse.daase:
( constructorname
  sourcefile
  constructorform
  documentation
  attributes
  predicates
)

This is essentially the same overall process as write-interpdb.

We reserve some space for the (key-table-byte-position, timestamp)

We loop across the list of constructors dumping the data and remembering the byte positions
in a key-value pair table.

We dump the final key-value pair table, write the byte position and time stamp at the top
of the file and close the file.

defun Write the browse database

[allConstructors p1032]
[$spadroot p9]
[*sourcefiles* p??]
[*print-pretty* p??]

— defun write-browsedb —

(defun write-browsedb ()

  (print ancestors out)
  (finish-output out))
  (setq ancestorspos nil))
  (push (list constructor opalistpos cmodemappos modemapspos
       obj categorypos niladic abbrev cosig kind defaultdomain
       ancestorspos) master))
  (finish-output out)
  (setq masterpos (file-position out))
  (print master out)
  (finish-output out)
  (file-position out 0)
  (print (cons masterpos (get-universal-time)) out)
  (finish-output out)
  (close out)))

———

66.1. DATABASE STRUCTURE

1029
"make browse.daase from hash tables"
(declare (special $spadroot *sourcefiles*))
(let (master masterpos src formpos attpos predpos *print-pretty* out)
 (declare (special *print-pretty*))
 (print "building browse.daase")
 (setq out (open "browse.build" :direction :output))
 (princ " " out)
 (finish-output out)
 (dolist (constructor (|allConstructors|))
   (let (struct)
     (setq struct (get constructor 'database))
     ; sourcefile is small. store the string directly
     (setq src (gethash constructor *sourcefiles*))
     (setq formpos (file-position out))
     (print (database-constructorform struct) out)
     (finish-output out)
     (setq docpos (file-position out))
     (print (database-documentation struct) out)
     (finish-output out)
     (setq attpos (file-position out))
     (print (database-attributes struct) out)
     (finish-output out)
     (setq predpos (file-position out))
     (print (database-predicates struct) out)
     (finish-output out)
     (push (list constructor src formpos docpos attpos predpos) master)))
 (finish-output out)
 (setq masterpos (file-position out))
 (print master out)
 (finish-output out)
 (file-position out 0)
 (print (cons masterpos (get-universal-time)) out)
 (finish-output out)
 (close out))

Building the category.daase from hash tables

This is a single table of category hash table information, dumped in the database format.

defun Write the category database

[genCategoryTable p??]
[*print-pretty* p??]
[*hasCategory-hash* p997]
— defun write-categorydb —

(defun write-categorydb ()
"make category.daase from scratch. contains the *hasCategory-hash* table"
(let (out master pos *print-pretty*)
  (declare (special *print-pretty* *hasCategory-hash*))
  (print "building category.daase")
  (genCategoryTable)
  (setq out (open "category.build" :direction :output))
  (princ " " out)
  (finish-output out)
  (maphash #'(lambda (key value)
                (if (or (null value) (eq value t))
                  (setq pos value)
                  (progn
                    (setq pos (file-position out))
                    (print value out)
                    (finish-output out)))
                (push (list key pos) master))
  *hasCategory-hash*)
  (setq pos (file-position out))
  (print master out)
  (finish-output out)
  (file-position out 0)
  (print (cons pos (get-universal-time)) out)
  (finish-output out)
  (close out)))

——

Building the operation.daase from hash tables

This is a single table of operations hash table information, dumped in the database format.

defun Write the operations database

[*operation-hash* p997]

— defun write-operationdb —

(defun write-operationdb ()
  (let (pos master out)
    (declare (special leaves *operation-hash*))
    (setq out (open "operation.build" :direction :output))
    (princ " " out)
    (finish-output out))
Database support operations

defun Data preloaded into the image at build time

[+$topicHash p??]

— defun write-warmdata —

(defun write-warmdata ()
"write out information to be loaded into the image at build time"
(declare (special +$topicHash+))
(with-open-file (out "warm.data" :direction :output)
(format out "(in-package "BOOT")~%")
(format out "((in-package "BOOT")"%")
(maphash #'(lambda (k v)
(format out "((setq +$topicHash+ (make-hash-table))"%")
(maphash #\(lambda (key value)
(setq pos (file-position out))
(print value out)
(finish-output out)
(push (cons key pos) master)
*operation-hash*)
(finish-output out)
(setq pos (file-position out))
(print master out)
(file-position out 0)
(print (cons pos (get-universal-time)) out)
(finish-output out)
(close out))))

—

defun Return all constructors

[*allconstructors* p1000]

— defun allConstructors —

(defun |allConstructors| ()
(declare (special *allconstructors*))
*allconstructors*
defun Return all operations
[*allOperations* p1000]
[*operation-hash* p997]

— defun allOperations —

(defun allOperations ()
  (declare (special *allOperations* *operation-hash*))
  (unless *allOperations*
    (maphash #'(lambda (k v) (declare (ignore v)) (push k *allOperations*))
      *operation-hash*)
    *allOperations*)

———
Chapter 67

System Statistics

defun statisticsInitialization
[gbc-time p??]

— defun statisticsInitialization —

(defun |statisticsInitialization| ()
"initialize the garbage collection timer"
#:acl (system:gbc-time 0)
nil)

67.1 Lisp Library Handling

defun loadLib

[startTimingProcess p??]
[getdatabase p1010]
[isSystemDirectory p1037]
[pathnameDirectory p1041]
[loadLibNoUpdate p1037]
[sayKeyedMsg p329]
[namestring p1040]
[clearConstructorCache p??]
[updateDatabase p1017]
[installConstructor p??]
[updateCategoryTable p??]
defun loadLib (cname)
(let (fullLibName systemdir? update? kind u coSig)
  (declare (special $CategoryFrame $InteractiveMode $printLoadMsgs
               $forceDatabaseUpdate))
  (startTimingProcess 'load)
  (when (setq fullLibName (getdatabase cname 'object))
    (setq systemdir? (isSystemDirectory (pathnameDirectory fullLibName)))
    (setq update? (or $forceDatabaseUpdate (null systemdir?)))
    (cond
      ((null update?) (loadLibNoUpdate cname cname fullLibName))
      (t
       (setq kind (getdatabase cname 'constructorkind))
       (when $printLoadMsgs
         (sayKeyedMsg 'S2IL0002 (list (namestring fullLibName) kind cname))
       (load fullLibName)
       (clearConstructorCache cname)
       (updateDatabase cname cname systemdir?)
       (installConstructor cname kind)
       (setq u (getdatabase cname 'constructormodemap))
       (updateCategoryTable cname kind)
       (setq coSig
         (when u
           (setq sig (cdar u))
           (cons nil (loop for x in (cdr sig) collect ([categoryForm? x]))) )))
     (if (null (cdr (getdatabase cname 'constructorform)))
       (setf (get cname 'niladic) t)
       (remprop cname 'niladic))
     (setf (get cname 'loaded) fullLibName)
     (when $InteractiveMode (setq $CategoryFrame (list (list nil)))
       (stopTimingProcess 'load)
       t)))))
67.1. LISP LIBRARY HANDLING

defun isSystemDirectory

[function p??]
[$spadroot p9]

--- defun isSystemDirectory ---

(defun |isSystemDirectory| (dir)
 (declare (special $spadroot))
 (every (|function| char=) $spadroot dir))

defun loadLibNoUpdate

[getdatabase p1010]
[sayKeyedMsg p329]
toplevel p??
clearConstructorCache p??
installConstructor p??
stopTimingProcess p??
$printLoadMsgs p736
$InteractiveMode p22
$CategoryFrame p??

--- defun loadLibNoUpdate ---

(defun |loadLibNoUpdate| (cname libName fullLibName)
 (declare (ignore libName))
 (let (kind)
  (declare (special |$CategoryFrame| |$InteractiveMode| |$printLoadMsgs|))
  (setq kind (getdatabase cname 'constructorkind))
  (when |$printLoadMsgs|
    (|sayKeyedMsg| 'S2IL0002 (list (|namestring| fullLibName) kind cname)))
  (cond
   ((equal (catch 'versioncheck (load fullLibName)) (- 1))
    (princ "  wrong library version...recompile ")
    (princ fullLibName)
    (terpri)
    (toplevel))
    (t
     (|clearConstructorCache| cname)
     (|installConstructor| cname kind)
     (setf (get cname 'loaded) fullLibName)
     (when |$InteractiveMode| (setq |$CategoryFrame| (list (list nil))))
     (|stopTimingProcess| 'load|))))
defun loadFunctor

(defun |loadFunctor| (u)
  (cond
   ((null (atom u)) (|loadFunctor| (car u)))
   (t
    (|loadLibIfNotLoaded| u)
    u)))
Chapter 68

Special Lisp Functions

68.1 Axiom control structure macros

Axiom used various control structures in the boot code which are not available in Common Lisp. We write some macros here to make the boot to lisp translations easier to read.

defun put

— defun put —

(defun put (sym ind val) (setf (get sym ind) val))

———

defmacro while

While the condition is true, repeat the body. When the condition is false, return t.

— defmacro while —

(defmacro while (condition &rest body) 
  `(loop (if (not ,condition) (return t)) ,@body))

———
defmacro whileWithResult
While the condition is true, repeat the body. When the condition is false, return the result form’s value.

(defmacro whileWithResult
  (condition result &rest body)
  `(loop (if (not ,condition) ,@result) ,@body))

68.2 Filename Handling
This code implements the Common Lisp pathname functions for Lisp/VM. On VM, a filename is 3-list consisting of the filename, filetype and filemode. We also UPCASE everything.

defun namestring

(defun namestring (arg)
  (namestring (pathname arg)))

defun pathnameName

(defun pathnameName (arg)
  (pathname-name (pathname arg)))

defun pathnameType

[pathname p1042]
--- defun pathnameType ---

(defun |pathnameType| (arg)
  (pathname-type (|pathname| arg)))

---

defun pathnameTypeId

[upcase p??]
[object2Identifier p??]
[pathnameType p1040]

--- defun pathnameTypeId ---

(defun |pathnameTypeId| (arg)
  (upcase (|object2Identifier| (|pathnameType| arg))))

---

defun mergePathnames

[pathnameName p1040]
[pathnameType p1040]
[pathnameDirectory p1041]

--- defun mergePathnames ---

(defun |mergePathnames| (a b)
  (let (fn ft fm)
    (cond
      ((string= (setq fn (|pathnameName| a)) "*") b)
      ((not (equal fn (|pathnameName| b))) a)
      ((string= (setq ft (|pathnameType| a)) "*") b)
      ((not (equal ft (|pathnameType| b))) a)
      ((equal (setq fm (|pathnameDirectory| a)) (list "*" ) b)
       (t a)))))

---

defun pathnameDirectory

[pathname p1042]
(defun pathnameDirectory (arg)
  (namestring (make-pathname :directory (pathname-directory (pathname arg)))))

(defun Axiom pathnames
  [pathname p]
  [make-filename p]

defun pathname (p)
  (cond
   ((null p) p)
   ((pathnamep p) p)
   ((null (consp p)) (pathname p))
   (t
    (when (> (length p) 2) (setq p (cons (elt p 0) (cons (elt p 1) nil))))
    (pathname (apply #'make-filename p)))))

defun makePathname
  [pathname p]
  [object2String p]

defun Delete a file
  [erase]
  [pathname p]
68.2. FILENAME HANDLING

---

(defun deleteFile (arg)
  (declare (special $erase))
  ($erase (pathname arg)))

---

(defun wrap (list-of-items wrapper)
  (prog nil
    (cond
      ((or (not (consp list-of-items)) (not wrapper))
       (return list-of-items))
      ((not (consp wrapper))
       (setq wrapper (lotsof wrapper)))
      (return
        (cons
          (if (first wrapper)
            ,(first wrapper) ,(first list-of-items))
            (first list-of-items))
            (wrap (cdr list-of-items) (cdr wrapper)))))))

---

(defun lotsof (&rest items)
  (setq items (copy-list items))
  (nconc items items))

---
defmacro startsId?

— defmacro startsId? —

(defun startsId? (x)
  (or (alpha-char-p x) (member x '(#\? #\% #\!) :test #'char=)))

defun hput

— defun hput —

(defun hput (table key value)
  (setf (gethash key table) value))

defmacro hget

— defmacro hget —

(defun hget (table key &rest default)
  (gethash key table default))

defun hkeys

— defun hkeys —

(defun hkeys (table)
  (let (keys)
    (maphash #'(lambda (key val) (declare (ignore val)) (push key keys)) table)
    keys))
defun digitp

— defun digitp —

(defun digitp (x)
  (or (and (symbolp x) (digitp (symbol-name x)))
      (and (characterp x) (digit-char-p x))
      (and (stringp x) (= (length x) 1) (digit-char-p (char x 0)))))

defun pname

Note it is important that PNAME returns nil not an error for non-symbols
— defun pname 0 —

(defun pname (x)
  (cond ((symbolp x) (symbol-name x))
        ((characterp x) (string x))
        (t nil)))

defun size

— defun size —

(defun size (l)
  (cond ((vectorp l) (length l))
        ((consp l) (list-length l))
        (t 0)))

defun strpos
(defun strpos (what in start dontcare)
  (setq what (string what) in (string in))
  (if dontcare
    (progn
      (setq dontcare (character dontcare))
      (search what in :start2 start
         :test #'(lambda (x y) (or (eql x dontcare) (eql x y))))))))

---

defun strposl

Note that this assumes “table” is a string.

---

defun strposl

 chants strposl (table cvec sint item)
  (setq cvec (string cvec))
  (if (not item)
    (position table cvec :test #'(lambda (x y) (position y x)) :start sint)
    (position table cvec :test-not #'(lambda (x y) (position y x)) :start sint))))

---

defun qenum

---

defun qenum 0

 chants qenum (cvec ind)
  (char-code (char cvec ind)))

---

defmacro identp

---

defmacro identp 0

 chants defmacro identp (x)
  (if (atom x)
'(and ,x (symbolp ,x))
(let ((xx (gensym)))
  '(let ((,xx ,x))
    (and ,xx (symbolp ,xx)))))

---

defun concat

[string-concatenate p??]

--- defun concat 0 ---

(defun concat (a b &rest l)
  (if (bit-vector-p a)
      (if l
          (apply #'concatenate 'bit-vector a b l)
          (concatenate 'bit-vector a b))
      (if l
          (apply #'system:string-concatenate a b l)
          (system:string-concatenate a b))))

---

This function was called |functionp| which is a lower-case version of the common lisp function called functionp. Camm Maguire found a bug related to this ambiguity so this was renamed.

defun canFuncall?

--- defun canFuncall? ---

(defun canFuncall? (fn)
  (if (identp fn)
      (and (fboundp fn) (not (macro-function fn)))
      (functionp fn)))

;; ——— NEW DEFINITION (override in msgdb.boot.pamphlet)
defun brightprint

(defun brightprint (x)
  (messageprint x))

;; ---
;;  NEW DEFINITION (override in msgdb.boot.pamphlet)

defun brightprint-0

(defun brightprint-0 (x)
  (messageprint-1 x))

;; ---

defun member

(defun member (item sequence)
  (cond
    ((symbolp item) (member item sequence :test #'eq))
    ((stringp item) (member item sequence :test #'equal))
    ((and (atom item) (not (arrayp item))) (member item sequence))
    (t (member item sequence :test #'equalp))))

;; ---

defun messageprint

(defun messageprint
  ;; ---

  ;; ---

(defun messageprint-1 (x)
  (cond
    ((or (eq x '|%l|) (equal x "%l") (terpri))
    ((stringp x) (princ x))
    ((identp x) (princ x))
    ((atom x) (princ x))
    ((princ ")")
    (messageprint-1 (car x))
    (messageprint-2 (cdr x))
    (princ "\n"))))

— defun messageprint-2 —

(defun messageprint-2 (x)
  (if (atom x)
    (unless x (progn (princ " . ") (messageprint-1 x)))
    (progn (princ " ") (messageprint-1 (car x)) (messageprint-2 (cdr x))))))

— defun sayBrightly1 —

(deffunction sayBrightly1
  (progn (princ "\n") (dlisp-message "\n")))
--- defun sayBrightly1 ---

(defun sayBrightly1 (x *standard-output*)
  (if (atom x)
      (progn (brightprint-0 x) (terpri) (force-output))
      (progn (brightprint x) (terpri) (force-output)))))

------

defmacro assq

TPDHERE: This could probably be replaced by the default assoc using eql

--- defmacro assq ---

(defmacro assq (a b)
  `(assoc ,a ,b :test #'eq))

------

defun A version of GET that works with lists

--- defun getl 0 ---

(defun getl (op prop)
  (when (and op (symbolp op)) (get op prop)))

------
Chapter 69

Record, Union, Mapping, and Enumeration

— postvars —

(eval-when (eval load)
(mapcar #'(lambda (alist)
    (setf (get (first alist) '|makeFunctionList|) (second alist)))
'((|Record| |mkRecordFunList|)
  (|Union| |mkUnionFunList|)
  (|Mapping| |mkMappingFunList|)
  (|Enumeration| |mkEnumerationFunList|))))
Chapter 70

Common Lisp Algebra Support

These functions are called directly from the algebra source code. They fall into two basic
categories, one are the functions that are raw Comon Lisp calls and the other are Axiom
specific functions or macros.

Raw function calls are used where there is an alignment of the Axiom type and the underlying
representation in Common Lisp. These form the support pillars upon which Axiom rests.
For instance, the 'EQ' function is called to support the Axiom equivalent 'eq?' function.

Macros are used to add type information in order to make low level operations faster. An
example is the use of macros in DoubleFloat to add Common Lisp type information. Since
DoubleFloat is machine arithmetic we give the compiler explicit type information so it can
generate fast code.

Functions are used to do manipulations which are Common Lisp operations but the Axiom
semantics are not the same. Because Axiom was originally written in Maclisp, then VMLisp,
and then Common Lisp some of these old semantics survive.

70.1 ApplicationProgramInterface

defun Report what domains get instantiated

    — defun reportinstantiations —

(defun reportinstantiations (b)
  (setq |$reportInstantiations| b))

    ——
70.2 InputForm

defun unparseInputForm

This fixes bug 7217. The default title generation is bogus. This is called from the unparse function in InputForm, bookvol10.3 Given a form, \( u \), we try to recover the input line that created it. [$InteractiveMode p22]

[formatSigAsTex p??]


defun unparseInputForm (u)
(let (|$formatSigAsTeX| |$InteractiveMode|)
  (declare (special |$formatSigAsTeX| |$InteractiveMode|))
  (setq |$formatSigAsTeX| 1)
  (setq |$InteractiveMode| nil)
  ([form2StringLocal| u])))

70.3 Void

defun voidValue


defun voidValue () "()"

70.4 U8Vector

defmacro qvlenU8


defmacro qvlenU8 (v)
  '((length (the (simple-array (unsigned-byte 8) (*)) ,v))))
70.5. **U16VECTOR**

```lisp
(defmacro eltU8
  (defmacro eltU8 —
  (defmacro eltU8 (v i)
    '(aref (the (simple-array (unsigned-byte 8) (*)) ,v) ,i)))

(defmacro seteltU8
  (defmacro seteltU8 —
  (defmacro seteltU8 (v i s)
    '(setf (aref (the (simple-array (unsigned-byte 8) (*)) ,v) ,i), s)))

(defun getRefvU8
  (defun getRefvU8 —
  (defun getRefvU8 (n x)
    (make-array n :initial-element x :element-type '(unsigned-byte 8))))

70.5 **U16Vector**

(defmacro qvlenU16
  (defmacro qvlenU16 —
  (defmacro qvlenU16 (v)
    (length (the (simple-array (unsigned-byte 16) (*)) ,v))))
```
defmacro eltU16

--- defmacro eltU16 ---

(defmacro eltU16 (v i)
  `(aref (the (simple-array (unsigned-byte 16) (*)) ,v) ,i))

defmacro seteltU16

--- defmacro seteltU16 ---

(defmacro seteltU16 (v i s)
  `(setf (aref (the (simple-array (unsigned-byte 16) (*)) ,v) ,i) ,s))

defun getRefvU16

--- defun getRefvU16 ---

(defun getRefvU16 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 16)))

70.6 U32Vector

defmacro qvlenU32

--- defmacro qvlenU32 ---

(defmacro qvlenU32 (v)
  `(length (the (simple-array (unsigned-byte 32) (*)) ,v)))
defmacro eltU32

— defmacro eltU32 —

(defun eltU32 (v i)
  `(aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i))

———

defmacro seteltU32

— defmacro seteltU32 —

(defun seteltU32 (v i s)
  `(setf (aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i) ,s))

———

defun getRefvU32

— defun getRefvU32 —

(defun getRefvU32 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 32))))

———

70.7  U8Matrix

defmacro aref2U8

— defmacro aref2U8 —

(defun aref2U8 (v i j)
  `(aref (the (simple-array (unsigned-byte 8) (* *)) ,v) ,i ,j))

———
defmacro setAref2U8

<table>
<thead>
<tr>
<th>defmacro setAref2U8</th>
</tr>
</thead>
</table>
| (defmacro setAref2U8 (v i j s)
  '(setf (aref (the (simple-array (unsigned-byte 8) (* *)) ,v) ,i ,j), s)) |

defmacro anrowsU8

<table>
<thead>
<tr>
<th>defmacro anrowsU8</th>
</tr>
</thead>
</table>
| (defmacro anrowsU8 (v)
  '(array-dimension (the (simple-array (unsigned-byte 8) (* *)) ,v) 0)) |

defmacro ancolsU8

<table>
<thead>
<tr>
<th>defmacro ancolsU8</th>
</tr>
</thead>
</table>
| (defmacro ancolsU8 (v)
  '(array-dimension (the (simple-array (unsigned-byte 8) (* *)) ,v) 1)) |

defmacro makeMatrixU8

<table>
<thead>
<tr>
<th>defmacro makeMatrixU8</th>
</tr>
</thead>
</table>
| (defmacro makeMatrixU8 (n m)
  '(make-array (list ,n ,m) :element-type '(unsigned-byte 8)
               :initial-element 0)) |
defmacro makeMatrix1U8

--- defmacro makeMatrix1U8 ---

(defun makeMatrix1U8 (n m s)
  `(make-array (list ,n ,m) :element-type '(unsigned-byte 8)
               :initial-element ,s))

70.8 U16Matrix

defmacro aref2U16

--- defmacro aref2U16 ---

(defun aref2U16 (v i j)
  `(aref (the (simple-array (unsigned-byte 16) (* *)) ,v) ,i ,j))

---

defmacro setAref2U16

--- defmacro setAref2U16 ---

(defun setAref2U16 (v i j s)
  `(setf (aref (the (simple-array (unsigned-byte 16) (* *)) ,v) ,i ,j), s))

---

defmacro anrowsU16

--- defmacro anrowsU16 ---

(defun anrowsU16 (v)
  `(array-dimension (the (simple-array (unsigned-byte 16) (* *)) ,v) 0))

---
defmacro ancolsU16

— defmacro ancolsU16 —

(defmacro ancolsU16 (v)
 `(array-dimension (the (simple-array (unsigned-byte 16) (* *)) ,v) 1))

---

defmacro makeMatrixU16

— defmacro makeMatrixU16 —

(defmacro makeMatrixU16 (n m)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 16)
 :initial-element 0))

---

defmacro makeMatrix1U16

— defmacro makeMatrix1U16 —

(defmacro makeMatrix1U16 (n m s)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 16)
 :initial-element ,s))

---

70.9 U32Matrix

defmacro aref2U32

— defmacro aref2U32 —

(defmacro aref2U32 (v i j)
 `(aref (the (simple-array (unsigned-byte 32) (* *)) ,v) ,i ,j))

---
defmacro setAref2U32

— defmacro setAref2U32 —

(defmacro setAref2U32 (v i j s)
  '(setf (aref (the (simple-array (unsigned-byte 32) (* *)) ,v) ,i ,j), s))

——

defmacro anrowsU32

— defmacro anrowsU32 —

(defmacro anrowsU32 (v)
  '(array-dimension (the (simple-array (unsigned-byte 32) (* *)) ,v) 0))

——

defmacro ancolsU32

— defmacro ancolsU32 —

(defmacro ancolsU32 (v)
  '(array-dimension (the (simple-array (unsigned-byte 32) (* *)) ,v) 1))

——

defmacro makeMatrixU32

— defmacro makeMatrixU32 —

(defmacro makeMatrixU32 (n m)
  '(make-array (list ,n ,m) :element-type '(unsigned-byte 32) :initial-element 0))

——
defmacro makeMatrix1U32

<table>
<thead>
<tr>
<th>defmacro makeMatrix1U32</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defmacro makeMatrix1U32 (n m s)</td>
</tr>
<tr>
<td>`(make-array (list ,n ,m) :element-type '(unsigned-byte 32) :initial-element ,s))</td>
</tr>
</tbody>
</table>

70.10 U32VectorPolynomialOperations

defmacro qsMulAdd6432

<table>
<thead>
<tr>
<th>defmacro qsMulAdd6432</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defmacro qsMulAdd6432 (x y z)</td>
</tr>
<tr>
<td>`(the (unsigned-byte 64)</td>
</tr>
<tr>
<td>(+ (the (unsigned-byte 64)</td>
</tr>
<tr>
<td>(* (the (unsigned-byte 32) ,x)</td>
</tr>
<tr>
<td>(the (unsigned-byte 32) ,y)))</td>
</tr>
<tr>
<td>(the (unsigned-byte 64) ,z))))</td>
</tr>
</tbody>
</table>

defmacro qsMulMod32

<table>
<thead>
<tr>
<th>defmacro qsMulMod32</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defmacro qsMulMod32 (x y)</td>
</tr>
<tr>
<td>`(the (unsigned-byte 64)</td>
</tr>
<tr>
<td>(* (the (unsigned-byte 32) ,x)</td>
</tr>
<tr>
<td>(the (unsigned-byte 32) ,y))))</td>
</tr>
</tbody>
</table>

defmacro qsMod6432

<table>
<thead>
<tr>
<th>defmacro qsMod6432</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defmacro qsMod6432 (x)</td>
</tr>
<tr>
<td>`(the (unsigned-byte 64)</td>
</tr>
<tr>
<td>(* (the (unsigned-byte 32) ,x)</td>
</tr>
<tr>
<td>(the (unsigned-byte 32) ,y))))</td>
</tr>
</tbody>
</table>
(defmacro qsMod6432 (x p)
  `(the (unsigned-byte 32)
    (rem (the (unsigned-byte 64) ,x) (the (unsigned-byte 32) ,p))))

---

defmacro qsMulAddMod6432

— defmacro qsMulAddMod6432 —

(defmacro qsMulAddMod6432 (x y z p)
  `(qsMod6432 (qsMulAdd6432 ,x ,y ,z) ,p))

---

defmacro qsMul6432

— defmacro qsMul6432 —

(defmacro qsMul6432 (x y)
  `(the (unsigned-byte 64)
    (* (the (unsigned-byte 32) ,x)
      (the (unsigned-byte 32) ,y))))

---

defmacro qsDot26432

— defmacro qsDot26432 —

(defmacro qsDot26432 (a1 b1 a2 b2)
  `(qsMulAdd6432 ,a1 ,b1 (qsMul6432 ,a2 ,b2)))

---

defmacro qsDot2Mod6432

— defmacro qsDot2Mod6432 —
(defmacro qsDot2Mod6432 (a1 b1 a2 b2 p) '((qsMod6432 (qsDot26432 ,a1 ,b1 ,a2 ,b2) ,p))

---

70.11 DirectProduct

defun vec2list

— defun vec2list —

(defun vec2list (vec) (coerce vec 'list))

---

70.12 AlgebraicFunction

defun retract

[|objMode| p??]
[|objVal| p??]
[isWrapped p??]
[qcar p??]
[retract1 p??]
[objNew p??]
[$EmptyMode| p??]

— defun retract —

(defun |retract| (object)
  (labels (
    (retract1 (object)
      (let (type val underDomain objectp)
        (declare (special |$SingleInteger| |$Integer| |$NonNegativeInteger|
                      |$PositiveInteger|))
        (setq type (|objMode| object))
        (cond
          ((stringp type) '|failed|)
          (t
           (setq val (|objVal| object))
           (cond
            ((equal type |$PositiveInteger|) (|objNew| val |$NonNegativeInteger|))
...
(equal type |$NonNegativeInteger|) (|objNew| val |$Integer|))
((and (equal type |$Integer|) (typep (|unwrap| val) |fixnum|))
 (|objNew| val |$SingleInteger|)))
(t
 (cond
 ((or (eql 1 (|#| type))
   (and (consp type) (eq (qcar type) '|Union|))
   (and (consp type) (eq (qcar type) '|FunctionCalled|)
     (and (consp (qcdr type)) (eq (qcddr type) nil)))
   (and (consp type) (eq (qcar type) '|OrderedVariableList|
     (and (consp (qcdr type)) (eq (qcddr type) nil)))
   (and (consp type) (eq (qcar type) '|Variable|
     (and (consp (qcdr type)) (eq (qcddr type) nil))))
   (if (setq objectp (|retract2Specialization| object))
     objectp
     '|failed|)
   (null (setq underDomain (|underDomainOf| type)))
     '|failed|)
 ; try to retract the "coefficients", e.g. P RN -> P I or M RN -> M I
 (t
   (setq objectp (|retractUnderDomain| object type underDomain))
   (cond
     ((not (eq objectp '|failed|)) objectp)
     ; see if we can use the retract functions
     ((setq objectp (|coerceRetract| object underDomain)) objectp)
     ; see if we have a special case here
     ((setq objectp (|retract2Specialization| object)) objectp)
     (t '|failed|)))))))))))
(let (type val ans)
 (declare (special |$EmptyMode|))
 (setq type (|objMode| object))
 (cond
   ((stringp type) '|failed|)
   ((equal type |$EmptyMode|) '|failed|)
   (t
     (setq val (|objVal| object))
     (cond
       ((and (null (|isWrapped| val))
         (null (and (consp val) (eq (qcar val) |map|)))
         '|failed|)
       (t
         (cond
           ((eq (setq ans (retract1 (|objNew| val type))) '|failed|)
             ans)
           (t (|objNew| (|objVal| ans) (|objMode| ans))))))))
)
### 70.13 Any

```lisp
defun spad2BootCoerce
  (defun spad2BootCoerce (x source target)
    (let (xp)
      (cond
        ((null (isValidType source)) (throwKeyedMsg 'S2IE0004 (list source)))
        ((null (isValidType target)) (throwKeyedMsg 'S2IE0004 (list target)))
        ((setq xp (coerceInteractive (objNewWrap x source) target))
          (objValUnwrap xp))
        (t
          (throwKeyedMsgCannotCoerceWithValue (wrap x) source target))))
```

### 70.14 ParametricLinearEquations

```lisp
defun algCoerceInteractive
  (defun algCoerceInteractive (p source target)
    (let [(useConvertForCoercions u)
      (declare (special useConvertForCoercions))
      (setq useConvertForCoercions t)
      (setq source (devaluate source))
      (setq target (devaluate target))
      (setq u (coerceInteractive (objNewWrap p source) target))
      (if u
        (objValUnwrap u)
        (error (list "can't convert" p "of mode" source "to mode" target))))
```

70.15  NumberFormats

defun ncParseFromString

— defun ncParseFromString —

(defun ncParseFromString (s)
  (zeroOneTran (catch 'SPAD_READER (parseFromString s))))

70.16  SingleInteger

defun qsquotient

— defun qsquotient 0 —

(defun qsquotient (a b)
  (the fixnum (truncate (the fixnum a) (the fixnum b))))

_____

defun qssremainder

— defun qssremainder 0 —

(defun qssremainder (a b)
  (the fixnum (rem (the fixnum a) (the fixnum b))))

_____

defmacro qsdifference

— defmacro qsdifference 0 —

(defmacro qsdifference (x y)
  '(the fixnum (- (the fixnum ,x) (the fixnum ,y)))))
defmacro qslessp

---
(defmacro qslessp (a b)
  '<(the fixnum ,a) (the fixnum ,b)))

---

defmacro qsadd1

---
(defmacro qsadd1 (x)
  '(the fixnum (1+ (the fixnum ,x))))

---

defmacro qssub1

---
(defmacro qssub1 (x)
  '(the fixnum (1- (the fixnum ,x))))

---

defmacro qsminus

---
(defmacro qsminus (x)
  '(the fixnum (minus (the fixnum ,x))))
defmacro qsplus

    — defmacro qsplus 0 —

(defmacro qsplus (x y)
    '(the fixnum (+ (the fixnum ,x) (the fixnum ,y))))

———

defmacro qstimes

    — defmacro qstimes 0 —

(defmacro qstimes (x y)
    '(the fixnum (* (the fixnum ,x) (the fixnum ,y))))

———

defmacro qsabsval

    — defmacro qsabsval 0 —

(defmacro qsabsval (x)
    '(the fixnum (abs (the fixnum ,x))))

———

defmacro qsoddp

    — defmacro qsoddp 0 —

(defmacro qsoddp (x)
    '(oddp (the fixnum ,x)))

———
defmacro qszerop

   — defmacro qszerop 0 —

(defmacro qszerop (x)
 '(zerop (the fixnum ,x)))

-----

defmacro qsmax

   — defmacro qsmax 0 —

(defmacro qsmax (x y)
 '(the fixnum (max (the fixnum ,x) (the fixnum ,y))))

-----

defmacro qsmin

   — defmacro qsmin 0 —

(defmacro qsmin (x y)
 '(the fixnum (min (the fixnum ,x) (the fixnum ,y))))

-----

70.17 Boolean

defun The Boolean = function support

   — defun BooleanEquality 0 —

(defun |BooleanEquality| (x y) (if x y (null y)))

-----
70.18 IndexedBits

**defmacro truth-to-bit**

IndexedBits new function support

— defmacro truth-to-bit —

(defmacro truth-to-bit (x) `(cond (,x 1) ('else 0)))

---

**defun IndexedBits new function support**

— defun bvec-make-full 0 —

(defun bvec-make-full (n x)
  (make-array (list n) :element-type 'bit :initial-element x))

---

**defmacro bit-to-truth**

IndexedBits elt function support

— defmacro bit-to-truth 0 —

(defmacro bit-to-truth (b) `(eq ,b 1))

---

**defmacro bvec-elt**

IndexedBits elt function support

— defmacro bvec-elt 0 —

(defmacro bvec-elt (bv i) `(sbit ,bv ,i))

---
defmacro bvec-setelt
IndexedBits setelt function support
   — defmacro bvec-setelt —
   (defmacro bvec-setelt (bv i x) '(setf (sbit ,bv ,i) ,x))

defmacro bvec-size
IndexedBits length function support
   — defmacro bvec-size —
   (defmacro bvec-size (bv) '(size ,bv))

defun IndexedBits concat function support
   — defun bvec-concat 0 —
   (defun bvec-concat (bv1 bv2) (concatenate '(vector bit) bv1 bv2))

defun IndexedBits copy function support
   — defun bvec-copy 0 —
   (defun bvec-copy (bv) (copy-seq bv))

defun IndexedBits = function support
   — defun bvec-equal 0 —
(defun bvec-equal (bv1 bv2) (equal bv1 bv2))

defun IndexedBits < function support

— defun bvec-greater 0 —

(defun bvec-greater (bv1 bv2)
  (let ((pos (mismatch bv1 bv2)))
    (cond ((or (null pos) (>= pos (length bv1))) nil)
          ((< pos (length bv2)) (> (bit bv1 pos) (bit bv2 pos)))
          ((find 1 bv1 :start pos) t)
          (t nil))))

defun IndexedBits And function support

— defun bvec-and 0 —

(defun bvec-and (bv1 bv2) (bit-and bv1 bv2))

defun IndexedBits Or function support

— defun bvec-or 0 —

(defun bvec-or (bv1 bv2) (bit-ior bv1 bv2))

defun IndexedBits xor function support

— defun bvec-xor 0 —
(defun bvec-xor (bv1 bv2) (bit-xor bv1 bv2))

(defun IndexedBits nand function support
  — defun bvec-nand 0 —
  (defun bvec-nand (bv1 bv2) (bit-nand bv1 bv2))

(defun IndexedBits nor function support
  — defun bvec-nor 0 —
  (defun bvec-nor (bv1 bv2) (bit-nor bv1 bv2))

(defun IndexedBits not function support
  — defun bvec-not 0 —
  (defun bvec-not (bv) (bit-not bv))

70.19 KeyedAccessFile

defun KeyedAccessFile defstream function support
This is a simpler interface to RDEFIOSTREAM [rdefiostream p??]
  — defun rdefinstream —
(defun rdefinstream (&rest fn)
    ;; following line prevents rdefiostream from adding a default filetype
    (unless (rest fn) (setq fn (list (pathname (car fn))))))
    (rdefiostream (list (cons 'file fn) '(mode . input))))

---

defun KeyedAccessFile defstream function support

---

defun rdefoutstream

(defun rdefoutstream (&rest fn)
    ;; following line prevents rdefiostream from adding a default filetype
    (unless (rest fn) (setq fn (list (pathname (car fn))))))
    (rdefiostream (list (cons 'FILE fn) '(mode . OUTPUT))))

---

70.20 Table

defun Table InnerTable support

We look inside the Key domain given to Table and find if there is an equality predicate associated with the domain. If found then Table will use a HashTable representation, otherwise it will use an AssociationList representation

[compiledLookup p??]
[Boolean p??]
[bpiname p??]
[knownEqualPred p??]

---

defun hashable

(defun |hashable| (dom)
    (labels (
        (|knownEqualPred| (dom)
            (let ((fun (|compiledLookup| '= '((|Boolean|) $ $) dom)))
                (if fun
                    (get (bpiname (car fun)) '|SPADreplace|)
                    nil)))
        (member (|knownEqualPred| dom) '(eq eql equal))))

---
defun compiledLookup

(isDomain p??)
[NRTEvalDomain p1079]

— defun compiledLookup —

(defun compiledLookup (op sig dollar)
  (unless (isDomain dollar) (setq dollar (NRTEvalDomain dollar)))
  (basicLookup op sig dollar dollar))

— defun basicLookup —

(defun basicLookup

(spadcall p??)
[hashCode? p??]
[opIsHasCat p??]
[HasCategory p??]
[hashType p??]
[hashString p??]
[error p??]
[vecp p??]
[isNewWorldDomain p??]
[oldCompLookup p1079]
[lookupInDomainVector p1078]
[$hashSeg p??]
[$hashOpSet p??]
[$hashOpApply p??]
[$hashOp0 p??]
[$hashOp1 p??]

(defun basicLookup (op sig domain dollar)
  (let (hashPercent box dispatch lookupFun hashSig val boxval)
    (declare (special $hashSeg $hashOpSet $hashOpApply $hashOp0 $hashOp1))
    (cond
      ((vecp domain)
       (if (isNewWorldDomain domain)
           (oldCompLookup op sig domain dollar)
           (lookupInDomainVector op sig domain dollar)))
      (t
       (setq hashPercent
            (if (vecp dollar)
(hashType\ (elt dollar 0) 0)
(hashType\ dollar 0))
(setq box (cons nil nil))
(cond
  ((null (vecp (setq dispatch (car domain))))
   (|error| 'bad domain format))
  (t
   (setq lookupFun (elt dispatch 3))
   (cond
    ((eql (elt dispatch 0) 0)
     (setq hashSig
       (cond
        ((hashCode? sig) sig)
        ((|opIsHasCat| op) (|hashType| sig hashPercent))
        (t (|hashType| (cons '|Mapping| sig) hashPercent))))
      (when (symbolp op)
        (cond
         ((eq op '|Zero|) (setq op |$hashOp0|))
         ((eq op '|One|) (setq op |$hashOp1|))
         ((eq op '|elt|) (setq op |$hashOpApply|))
         ((eq op '|setelt|) (setq op |$hashOpSet|))
         (t (setq op (|hashString| (symbol-name op)))))
      (cond
       ((setq val
         (car
          (spadcall (cdr domain) dollar op hashSig box nil lookupFun)))
        val)
       ((hashCode? sig) nil)
       ((or (> (#\ sig) 1) (|opIsHasCat| op)) nil)
       (setq boxval
        (spadcall (cdr dollar) dollar op
          (|hashType| (car sig) hashPercent)
          box nil lookupFun))
       (cons #'identity (car boxval)))
       (t nil))
    ((|opIsHasCat| op) (|HasCategory| domain sig))
    (t
     (when (hashCode? op)
      (cond
       ((eql op |$hashOp1|) (setq op '|One|))
       ((eql op |$hashOp0|) (setq op '|Zero|))
       ((eql op |$hashOpApply|) (setq op '|elt|))
       ((eql op |$hashOpSet|) (setq op '|setelt|))
       ((eql op |$hashSeg|) (setq op 'segment)))
      (cond
       ((and (hashCode? sig) (eql sig hashPercent))
        (spadcall
         (car (spadcall (cdr dollar) dollar op '($) box nil lookupFun)))
        t)
       (car
        (spadcall (cdr dollar) dollar op '($) box nil lookupFun)))
     t))
defun lookupInDomainVector  
|defun lookupInDomainVector| (op sig domain dollar)  
(if (consp domain)  
  (|basicLookupCheckDefaults| op sig domain domain)  
  (spadcall op sig dollar (elt domain 1))))

defun basicLookupCheckDefaults  
|defun basicLookupCheckDefaults| (op sig domain dollar)  
(declare (ignore domain))  
(let (box dispatch lookupFun hashPercent hashSig)  
  (declare (special |$lookupDefaults|))  
  (setq box (cons nil nil))  
  (cond  
    ((null (vecp (setq dispatch (car dollar))))  
      (|error| '|bad domain format|))  
    (t  
      (setq lookupFun (elt dispatch 3))  
      (cond  
        ((eql (elt dispatch 0) 0)  
          (setq hashPercent  
            (if (vecp dollar)  
              (setq hashPercent (elt dispatch 1))  
              (if (vecp domain)  
                (setq hashPercent (elt dispatch 1))  
                (setq hashPercent 0))))  
        (t  
          (setq hashPercent (elt dispatch 1))))  
      (setq hashSig (elt dispatch 2)))  
  (setq dispatch (car dollar)  
    (if (vecp dispatch)  
      (setq dispatch (cadr dispatch))  
      (setq dispatch (caddr dispatch))))  
  (setq hashType (elt dispatch 0))  
  (hashType p??)  
  (hashCode? p??)  
  (hashCode? p??)  
  (hashString p??)  
  (spadcall p??)  
  ($lookupDefaults p??))
((|hashType| (elt dollar 0) 0)
 (|hashType| dollar 0)))
(setq hashSig
 (if (|hashCode?| sig)
  sig
  (|hashType| (cons '|Mapping| sig) hashPercent)))
(when (symbolp op) (setq op (|hashString| (symbol-name op)))
 (car (spadcall (cdr dollar) dollar op hashSig
  box (null |$lookupDefaults|) lookupFun)))
(t
 (car (spadcall (cdr dollar) dollar op sig box
  (null |$lookupDefaults|) lookupFun))))))

defun oldCompLookup

[lookupInDomainVector p1078]
[$lookupDefaults p??]

— defun oldCompLookup —

(defun |oldCompLookup| (op sig domvec dollar)
 (let ((|$lookupDefaults| u)
  (declare (special |$lookupDefaults|)))
  (setq |$lookupDefaults| nil)
  (cond
   ((setq u (|lookupInDomainVector| op sig domvec dollar))
    u)
   (t
    (setq |$lookupDefaults| t)
    (|lookupInDomainVector| op sig domvec dollar)))))

— defun NRTevalDomain —

defun NRTevalDomain

[qcar p??]
[eval p??]
[evalDomain p913]

— defun NRTevalDomain —

(defun |NRTevalDomain| (form)
 (if (and (consp form) (eq (qcar form) 'setelt))
(|eval| form)
(|evalDomain| form))

---

### 70.21 Plot3d

We catch numeric errors and throw a different failure than normal. The trapNumericErrors macro will return a pair of the form `Union(type-of-form, "failed")`. This pair is tested for eq-ness so it has to be unique. It lives in the defvar `$numericFailure`. The old value of the `$BreakMode` variable is saved in a defvar named `$oldBreakMode`.

```lisp
(defvar $numericFailure
  This is a failed union branch which is the value returned for numeric failure.
  — initvars —
  (defvar $numericFailure (cons 1 "failed"))

---

defvar $oldBreakMode
  — initvars —
  (defvar $oldBreakMode nil "the old value of the $BreakMode variable")

---

defmacro trapNumericErrors
  The following macro evaluates form returning `Union(type-of form, "failed")`. It is used in the myTrap local function in Plot3d.
  — defmacro trapNumericErrors —
  (defmacro |trapNumericErrors| (form)
    `(let ((|$oldBreakMode| |$BreakMode|) (|$BreakMode| '|trapNumerics|) (val))
      (declare (special |$BreakMode| |$numericFailure| |$oldBreakMode|))
      (setq val (catch '|trapNumerics| ,form))
      (if (eq val |$numericFailure|) val (cons 0 val))))
```
70.22  **DoubleFloatVector**

Double Float Vectors are simple arrays of lisp double-floats made available at the Spad language level. Note that these vectors are 0 based whereas other Spad language vectors are 1-based.

**defmacro dlen**

DoubleFloatVector Qsize function support

— **defmacro dlen** —

(defmacro dlen (v)
  `(length (the (simple-array double-float (*)) ,v)))

**defmacro make-double-vector**

DoubleFloatVector Qnew function support

— **defmacro make-double-vector** —

(defmacro make-double-vector (n)
  `(make-array (list ,n) :element-type 'double-float))

**defmacro make-double-vector1**

DoubleFloatVector Qnew1 function support

— **defmacro make-double-vector1** —

(defmacro make-double-vector1 (n s)
  `(make-array (list ,n) :element-type 'double-float :initial-element ,s))
CHAPTER 70. COMMON LISP ALGEBRA SUPPORT

defmacro delt

DoubleFloatVector Qelt1 function support
—— defmacro delt ——

(defmacro delt (v i)
  `(aref (the (simple-array double-float (*)) ,v) ,i))

———

defmacro dsetelt

DoubleFloatVector Qsetelt1 function support
—— defmacro dsetelt ——

(defmacro dsetelt (v i s)
  '(setf (aref (the (simple-array double-float (*)) ,v) ,i) ,s))

———

70.23 ComplexDoubleFloatVector

Complex Double Float Vectors are simple arrays of lisp double-floats made available at the Spad language level. Note that these vectors are 0 based whereas other Spad language vectors are 1-based. Complex array is implemented as an array of doubles. Each complex number occupies two positions in the real array.

defmacro make-cdouble-vector

ComplexDoubleFloatVector Qnew function support
—— defmacro make-cdouble-vector ——

(defmacro make-cdouble-vector (n)
  `(make-array (list (* 2 ,n)) :element-type 'double-float))

———

defmacro cdelt

ComplexDoubleFloatVector Qelt1 function support
—— defmacro cdelt ——
(defmacro CDELT(ov oi)
   (let ((v (gensym))
      (i (gensym)))
   '(let ((,v ,ov)
      (,i ,oi))
   (cons
    (aref (the (simple-array double-float (*)) ,v) (* 2 ,i))
    (aref (the (simple-array double-float (*)) ,v) (+ (* 2 ,i) 1))))))

defmacro cdsetelt
ComplexDoubleFloatVector Qsetelt1 function support
   — defmacro cdsetelt —

(defmacro cdsetelt(ov oi os)
   (let ((v (gensym))
      (i (gensym))
      (s (gensym)))
   '(let ((,v ,ov)
      (,i ,oi)
      (,s ,os))
   (setf (aref (the (simple-array double-float (*)) ,v) (* 2 ,i))
      (car ,s))
   (setf (aref (the (simple-array double-float (*)) ,v) (+ (* 2 ,i) 1))
      (cdr ,s))
   ,s)))

defmacro cdlen
ComplexDoubleFloatVector Qsize function support
   — defmacro cdlen —

(defmacro cdlen(v)
   '(truncate (length (the (simple-array double-float (*)) ,v)) 2))
CHAPTER 70. COMMON LISP ALGEBRA SUPPORT

70.24 DoubleFloatMatrix

defmacro make-double-matrix

DoubleFloatMatrix qnew function support
— defmacro make-double-matrix —

(defun make-double-matrix (n m)
  '(make-array (list ,n ,m) :element-type 'double-float))

—

defmacro make-double-matrix1

DoubleFloatMatrix new function support
— defmacro make-double-matrix1 —

(defun make-double-matrix1 (n m s)
  '(make-array (list ,n ,m) :element-type 'double-float
    :initial-element ,s))

—

defmacro daref2

DoubleFloatMatrix qelt function support
— defmacro daref2 —

(defun daref2 (v i j)
  '(aref (the (simple-array double-float (* *)) ,v) ,i ,j))

—

defmacro dsetaref2

DoubleFloatMatrix qsetelt! function support
— defmacro dsetaref2 —

(defun dsetaref2 (v i j s)
  '(setf (aref (the (simple-array double-float (* *)) ,v) ,i ,j)
    ,s))

—
defmacro danrows
DoubleFloatMatrix nrows function support
— defmacro danrows —

(defun danrows (v)
  (array-dimension (the (simple-array double-float (* *)) ,v) 0))
—

defmacro dancols
DoubleFloatMatrix ncols function support
— defmacro dancols —

(defun dancols (v)
  (array-dimension (the (simple-array double-float (* *)) ,v) 1))
—

70.25 ComplexDoubleFloatMatrix
defmacro make-cdouble-matrix
ComplexDoubleFloatMatrix function support
— defmacro make-cdouble-matrix —

(defun make-cdouble-matrix (n m)
  (make-array (list ,n (* 2 ,m)) :element-type 'double-float))
—

defmacro cdaref2
ComplexDoubleFloatMatrix function support
— defmacro cdaref2 —

(defun cdaref2 (ov oi oj)
  (let ((v (gensym))
        (i (gensym))
        (j (gensym))
        (,v ,ov)
        (,i i)
        (,j j))
    (let ((,v ,ov)
          (,i i)
          (,j j))))
(,i ,oi)
(,j ,oj))
(cons
  (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
  (aref (the (simple-array double-float (* *)) ,v)
    ,i (+ (* 2 ,j) 1)))))

---

defmacro cdsetaref2

ComplexDoubleFloatMatrix function support
  — defmacro cdsetaref2 —

(defun cdsetaref2 (ov oi oj os)
  (let ((v (gensym))
      (i (gensym))
      (j (gensym))
      (s (gensym)))
    (let ((,v ,ov)
      (,i ,oi)
      (,j ,oj)
      (,s ,os))
      (setf (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
        (car ,s))
      (setf (aref (the (simple-array double-float (* *)) ,v)
        ,i (+ (* 2 ,j) 1))
        (cdr ,s))
      ,s))))

---

defmacro cdanrows

ComplexDoubleFloatMatrix function support
  — defmacro cdanrows —

(defun cdanrows (v)
  '(array-dimension (the (simple-array double-float (* *)) ,v) 0))
defmacro cdancols

ComplexDoubleFloatMatrix function support

— defmacro cdancols —

```
(defun cdancols (v)
  (truncate
   (array-dimension (the (simple-array double-float (* *)) ,v) 1) 2))
```

70.26 Integer

defun Integer divide function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses
the Integer divide function actually inlines a call to this code. The Integer domain contains
the line:

```
(PUT (QUOTE |INT;divide;2$R;44|) (QUOTE |SPADreplace|) (QUOTE DIVIDE2))
```

— defun divide2 0 —

```
(defun divide2 (x y)
  (multiple-value-call #'cons (truncate x y)))
```

defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses
the Integer quo function actually inlines a call to this code. The Integer domain contains
the line:

```
(PUT (QUOTE |INT;rem;3$;46|) (QUOTE |SPADreplace|) (QUOTE REMAINDER2))
```

Because these are identical except for name we make the symbol-functions equivalent. This
was done in the original code for efficiency.

— defun remainder2 0 —

```
(setf (symbol-function 'remainder2) #'rem)
```
defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses
the Integer quo function actually inlines a call to this code. The Integer domain contains
the line:

```lisp
(PUT (QUOTE |INT;quo;3$;45|) (QUOTE |SPADreplace|) (QUOTE QUOTIENT2))
```

— defun quotient2 0 —

```lisp
(defun quotient2 (x y)
  (values (truncate x y)))
```

defun Integer random function support

This is used for calls to random with no arguments. If an argument is supplied to random
then the common lisp random function is called directly. This could be lifted up into the
spad code.

— defun random 0 —

```lisp
(defun random () (random (expt 2 26)))
```

70.27 IndexCard

defun IndexCard origin function support

```lisp
(defun alqlGetOrigin (x)
  (let (field k)
    (setq field (\|dbPart\| x 5 1))
    (setq k (\|charPosition\| \#\|\( field 2\)
      (substring field 1 (1- k)))))
```

———
defun IndexCard origin function support
[dbPart p??]
[charPosition p??]
[substring p??]

— defun alqlGetParams —

(defun |alqlGetParams| (x)
  (let (field k)
    (setq field (|dbPart| x 5 1))
    (setq k (|charPosition| #\( field 2))
    (substring field k nil)))

defun IndexCard elt function support
[dbPart p??]
[substring p??]

— defun alqlGetKindString —

(defun |alqlGetKindString| (x)
  (if (or (char= (elt x 0) #\a) (char= (elt x 0) #\o))
    (substring (|dbPart| x 5 1) 0 1)
    (substring x 0 1)))

70.28  OperationsQuery

defun OperationQuery getDatabase function support

This function, called as |getBrowseDatabase| (arg) returns a list of appropriate entries in the browser database. The legal values for arg are

- “o” (operations)
- “k” (constructors)
- “d” (domains)
- “c” (categories)
• “p” (packages)

---

70.29 Database

defun Database elt function support

---

70.30 FileName

defun FileName filename function implementation

---
defun FileName filename support function

[lastc p??]

--- defun StringToDir ---

(defun StringToDir (s)
  (cond
    ((string= s "/") '(:root))
    ((string= s "/") nil)
    (t
      (let ((lastc (aref s (- (length s) 1))))
        (if (char= lastc #\/) (pathname-directory (concat s "name.type"))
            (pathname-directory (concat s "/name.type")))))))

---

defun FileName directory function implementation

[DirToString p1091]

--- defun fnameDirectory ---

(defun fnameDirectory (f)
  (dirToString (pathname-directory f)))

---

defun FileName directory function support

For example, "/" "/u/swatt" ".src"

--- defun DirToString 0 ---

(defun DirToString (d)
  (cond
    ((equal d '(:root) "/")
    (null d "")
    (t (string-right-trim "/" (namestring (make-pathname :directory d)))))))

---
defun FileName name function implementation

— defun fnameName 0 —

(defun |fnameName| (f)
  (let ((s (pathname-name f)))
    (if s s "") ))

——

defun FileName extension function implementation

— defun fnameType 0 —

(defun |fnameType| (f)
  (let ((s (pathname-type f)))
    (if s s "") ))

——

defun FileName exists? function implementation

— defun fnameExists? 0 —

(defun |fnameExists?| (f)
  (if (probe-file (namestring f)) 't nil))

——

defun FileName readable? function implementation

— defun fnameReadable? 0 —

(defun |fnameReadable?| (f)
  (let ((s (open f :direction :input :if-does-not-exist nil)))
    (cond (s (close s) t) (t nil)) ))

——
defun FileName writeable? function implementation

[myWriteable? p??]
    — defun fnameWritable? —

(defun |fnameWritable?| (f)
    (|myWriteable?| (namestring f)) )

defun FileName writeable? function support

[error p??]
[fnameExists? p1092]
[fnameDirectory p1091]
[writeablep p??]
    — defun myWritable? —

(defun |myWritable?| (s)
    (if (not (stringp s)) (|error| "'myWritable?' requires a string arg.")
    (if (string= s "") (setq s ".")
    (if (not (|fnameExists?| s)) (setq s (|fnameDirectory| s)))
    (if (string= s "") (setq s ".")
    (if (> (|writeablep| s) 0) 't nil) )

defun FileName new function implementation

[fnameMake p1090]
    — defun fnameNew —

(defun |fnameNew| (d n e)
    (if (not (|myWritable?| d))
        nil
    (do ((fn))
        (nil)
        (setq fn (|fnameMake| d (string (gensym n)) e))
    (if (not (probe-file (namestring fn)))
        (return-from |fnameNew| fn)))))
70.31 DoubleFloat

These macros wrap their arguments with strong type information in order to optimize double-float computations. They are used directly in the DoubleFloat domain (see Volume 10.3).

**defmacro DFLessThan**

Compute a strongly typed double-float comparison See Steele Common Lisp 1990 p293

```
(defmacro DFLessThan (x y)
  '<(the double-float ,x) (the double-float ,y)))
```

**defmacro DFUnaryMinus**

Compute a strongly typed unary double-float minus See Steele Common Lisp 1990 p295

```
(defmacro DFUnaryMinus (x)
  '(the double-float (~ (the double-float ,x))))
```

**defmacro DFMinusp**

Compute a strongly typed unary double-float test for negative See Steele Common Lisp 1990 p292

```
(defmacro DFMinusp (x)
  '(minusp (the double-float ,x)))
```

**defmacro DFZerop**

Compute a strongly typed unary double-float test for zero See Steele Common Lisp 1990 p292

```
(defmacro DFZerop (x)
  '(zerop (the double-float ,x)))
```
(defmacro DFZerop (x)
  '(zerop (the double-float ,x)))

---

defmacro DFAdd

Compute a strongly typed doublefloat addition See Steele Common Lisp 1990 p295

---

(defmacro DFAdd (x y)
  '(the double-float (+ (the double-float ,x) (the double-float ,y))))

---

defmacro DFSubtract

Compute a strongly typed doublefloat subtraction See Steele Common Lisp 1990 p295

---

(defmacro DFSubtract (x y)
  '(the double-float (- (the double-float ,x) (the double-float ,y))))

---

defmacro DFMultiply

Compute a strongly typed doublefloat multiplication See Steele Common Lisp 1990 p296

---

(defmacro DFMultiply (x y)
  '(the double-float (* (the double-float ,x) (the double-float ,y))))

---

defmacro DFIntegerMultiply

Compute a strongly typed doublefloat multiplication by an integer. See Steele Common Lisp 1990 p296
(defmacro DFIntegerMultiply (i y)
  '(the double-float (* (the integer ,i) (the double-float ,y))))

----------

defmacro DFMax

Choose the maximum of two doublefats. See Steele Common Lisp 1990 p294
— defmacro DFMax —

(defmacro DFMax (x y)
  '(the double-float (max (the double-float ,x) (the double-float ,y))))

----------

defmacro DFMin

Choose the minimum of two doublefats. See Steele Common Lisp 1990 p294
— defmacro DFMin —

(defmacro DFMin (x y)
  '(the double-float (min (the double-float ,x) (the double-float ,y)))))

----------

defmacro DFEql

Compare two doublefats for equality, where equality is eq, or numbers of the same type
with the same value. See Steele Common Lisp 1990 p105
— defmacro DFEql —

(defmacro DFEql (x y)
  '(eql (the double-float ,x) (the double-float ,y)))))

----------

defmacro DFDivide

Divide a doublefloat by a a doublefloat See Steele Common Lisp 1990 p296
— defmacro DFDivide —
(defmacro DFDivide (x y)
 'the double-float (/ (the double-float ,x) (the double-float ,y))))

defmacro DFIntegerDivide

Divide a doublefloat by an integer See Steele Common Lisp 1990 p296
— defmacro DFIntegerDivide —

(defmacro DFIntegerDivide (x i)
 'the double-float (/ (the double-float ,x) (the integer ,i))))

defmacro DFSqrt

Compute the doublefloat square root of x. The result will be complex if the argument is negative. See Steele Common Lisp 1990 p302
— defmacro DFSqrt —

(defmacro DFSqrt (x)
 'sqrt (the double-float ,x)))

defmacro DFLogE

Compute the doublefloat log of x with the base e. The result will be complex if the argument is negative. See Steele Common Lisp 1990 p301
— defmacro DFLogE —

(defmacro DFLogE (x)
 'log (the double-float ,x)))

defmacro DFLog

Compute the doublefloat log of x with a given base b. The result will be complex if x is negative. See Steele Common Lisp 1990 p301
— defmacro DFLog —
(defmacro DFLog (x b)
  '(log (the double-float ,x) (the fixnum ,b)))

---

**defmacro DFIntegerExpt**

Compute the double float expt of \textit{x} with a given integer power \textit{i} See Steele Common Lisp 1990 p300

---

(defmacro DFIntegerExpt (x i)
  '(the double-float (expt (the double-float ,x) (the integer ,i)))))

---

**defmacro DFExpt**

Compute the double float expt of \textit{x} with a given power \textit{p}. The result could be complex if the base is negative and the power is not an integer. See Steele Common Lisp 1990 p300

---

(defmacro DFExpt (x p)
  '(expt (the double-float ,x) (the double-float ,p)))

---

**defmacro DFExp**

Compute the double float exp with power \textit{e} See Steele Common Lisp 1990 p300

---

(defmacro DFExp (x)
  '(the double-float (exp (the double-float ,x)))))

---

**defmacro DFSin**

Compute a strongly typed double float sin See Steele Common Lisp 1990 p304

---
(defmacro DFSin (x)
  `(the double-float (sin (the double-float ,x)))))

------

defmacro DFCos
Compute a strongly typed double float cos See Steele Common Lisp 1990 p304
   — defmacro DFCos —

(defmacro DFCos (x)
  `(the double-float (cos (the double-float ,x)))))

------

defmacro DFTan
Compute a strongly typed double float tan See Steele Common Lisp 1990 p304
   — defmacro DFTan —

(defmacro DFTan (x)
  `(the double-float (tan (the double-float ,x)))))

------

defmacro DFAsin
Compute a strongly typed double float asin. The result is complex if the absolute value of
the argument is greater than 1. See Steele Common Lisp 1990 p305
   — defmacro DFAsin —

(defmacro DFAsin (x)
  `(asin (the double-float ,x)))))

------

defmacro DFAcos
Compute a strongly typed double float acos. The result is complex if the absolute value of
the argument is greater than 1. See Steele Common Lisp 1990 p305
   — defmacro DFAcos —

(defmacro DFAcos (x)
  '(acos (the double-float ,x)))

---

**defmacro DFAtan**

Compute a strongly typed doublefloat atan See Steele Common Lisp 1990 p305

---

(defmacro DFAtan (x)
  '(the double-float (atan (the double-float ,x))))

---

**defmacro DFAtan2**

Compute a strongly typed doublefloat atan with 2 arguments

\[
\begin{align*}
  y &= 0 \quad x > 0 \quad \text{Positive x-axis} \quad 0 \\
  y > 0 \quad x > 0 \quad \text{Quadrant I} \quad 0 < \text{result} < \pi/2 \\
  y > 0 \quad x = 0 \quad \text{Positive y-axis} \quad \pi/2 \\
  y > 0 \quad x < 0 \quad \text{Quadrant II} \quad \pi/2 < \text{result} < \pi \\
  y = 0 \quad x < 0 \quad \text{Negative x-axis} \quad \pi \\
  y < 0 \quad x < 0 \quad \text{Quadrant III} \quad -\pi < \text{result} < -\pi/2 \\
  y < 0 \quad x = 0 \quad \text{Negative y-axis} \quad -\pi/2 \\
  y < 0 \quad x > 0 \quad \text{Quadrant IV} \quad -\pi/2 < \text{result} < 0 \\
  y = 0 \quad x = 0 \quad \text{Origin} \quad \text{error}
\end{align*}
\]
See Steele Common Lisp 1990 p306

---

(defmacro DFAtan2 (y x)
  '(the double-float (atan (the double-float ,x) (the double-float ,y))))

---

**defmacro DFSinh**

Compute a strongly typed doublefloat sinh

\[
\frac{e^z - e^{-z}}{2}
\]
See Steele Common Lisp 1990 p308
(defmacro DFSinh (x)
  '(the double-float (sinh (the double-float ,x)))))

----

defmacro DFCosh

Compute a strongly typed doublefloat cosh

\[(e^z + e^{-z})/2\]

See Steele Common Lisp 1990 p308
  — defmacro DFCosh —

(defmacro DFCosh (x)
  '(the double-float (cosh (the double-float ,x)))))

----

defmacro DFTanh

Compute a strongly typed doublefloat tanh

\[(e^z - e^{-z})/(e^z + e^{-z})\]

See Steele Common Lisp 1990 p308
  — defmacro DFTanh —

(defmacro DFTanh (x)
  '(the double-float (tanh (the double-float ,x)))))

----

defmacro DFAsinh

Compute the inverse hyperbolic sin.

\[\log \left(z + \sqrt{1 + z^2}\right)\]

See Steele Common Lisp 1990 p308
  — defmacro DFAsinh —

(defmacro DFAsinh (x)
  '(the double-float (asinh (the double-float ,x)))))

----
defmacro DFAcosh

Compute the inverse hyperbolic cos. Note that the acosh function will return a complex result if the argument is less than 1.

$$\log \left( z + (z + 1)\sqrt{(z - 1)/(z + 1)} \right)$$

See Steele Common Lisp 1990 p308

— defmacro DFAcosh —

(defmacro DFAcosh (x)
  `(acosh (the double-float ,x)))

defmacro DFAtanh

Compute the inverse hyperbolic tan. Note that the acosh function will return a complex result if the argument is greater than 1.

$$\log \left( (1 + z)\sqrt{1/(1 - z^2)} \right)$$

See Steele Common Lisp 1990 p308

— defmacro DFAtanh —

(defmacro DFAtanh (x)
  `(atanh (the double-float ,x)))

defun Machine specific float numerator

This is used in the DoubleFloat integerDecode function

— defun integer-decode-float-numerator 0 —

(defun integer-decode-float-numerator (x)
  (integer-decode-float x))
defun Machine specific float denominator
This is used in the DoubleFloat integerDecode function
   — defun integer-decode-float-denominator 0 —

(defun integer-decode-float-denominator (x)
  (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
    (declare (ignore mantissa sign)) (expt 2 (abs exponent))))

defun Machine specific float sign
This is used in the DoubleFloat integerDecode function
   — defun integer-decode-float-sign 0 —

(defun integer-decode-float-sign (x)
  (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
    (declare (ignore mantissa exponent) sign))

defun Machine specific float bit length
This is used in the DoubleFloat integerDecode function
   — defun integer-decode-float-exponent 0 —

(defun integer-decode-float-exponent (x)
  (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
    (declare (ignore mantissa sign)) exponent))

defun Decode floating-point values
This function is used by DoubleFloat to implement the “mantissa” and “exponent” functions.
   — defun manexp 0 —

(defun manexp (u)
  (multiple-value-bind (f e s)
      (decode-float u)
    (cons (* s f) e)))


defun The cotangent routine

The cotangent function is defined as
\[ \cot(z) = \frac{1}{\tan(z)} \]

---

(defun cot (a)
  (if (or (> a 1000.0) (< a -1000.0))
      (/ (cos a) (sin a))
      (/ 1.0 (tan a))))

---

defun The inverse cotangent function

The inverse cotangent (arc-cotangent) function is defined as
\[ \text{acot}(z) = \cot^{-1}(z) = \tan^{-1}\left(\frac{1}{z}\right) \]

See Steele Common Lisp 1990 pp305-307

---

(defun acot (a)
  (if (> a 0.0)
      (if (> a 1.0)
          (atan (/ 1.0 a))
          (- (/ pi 2.0) (atan a)))
      (if (< a -1.0)
          (- pi (atan (/ -1.0 a)))
          (+ (/ pi 2.0) (atan (- a))))))

---

defun The secant function

\[ \sec(x) = \frac{1}{\cos(x)} \]

---

(defun sec (a)
  (if (> a 0.0)
      (if (> a 1.0)
          (atan (/ 1.0 a))
          (- (/ pi 2.0) (atan a)))
      (if (< a -1.0)
          (- pi (atan (/ -1.0 a)))
          (+ (/ pi 2.0) (atan (- a)))))

---
(defun sec (x) (/ 1 (cos x)))

defun The inverse secant function

\[ asec(x) = \cos^{-1} \left( \frac{1}{x} \right) \]

— defun asec 0 —

(defun asec (x) (acos (/ 1 x)))

defun The cosecant function

\[ csc(x) = \frac{1}{\sin(x)} \]

— defun csc 0 —

(defun csc (x) (/ 1 (sin x)))

defun The inverse cosecant function

\[ acsc(x) = \frac{1}{\sin^{-1}(x)} \]

— defun acsc 0 —

(defun acsc (x) (asin (/ 1 x)))
defun The hyperbolic cosecant function

$$csch(x) = \frac{1}{\sinh(x)}$$

— defun csch 0 —

(defun csch x (/ 1 (sinh x)))

——

defun The hyperbolic cotangent function

$$coth(x) = \cosh(x)csch(x)$$

— defun coth 0 —

(defun coth x (* (cosh x) (csch x)))

——

defun The hyperbolic secant function

$$sech(x) = \frac{1}{\cosh(x)}$$

— defun sech 0 —

(defun sech x (/ 1 (cosh x)))

——

defun The inverse hyperbolic cosecant function

$$acsch(x) = \text{asinh}\left(\frac{1}{x}\right)$$

— defun acsch 0 —
(defun acsch (x) (asinh (/ 1 x)))

defun The inverse hyperbolic cotangent function

\[ acoth(x) = \text{atanh} \left( \frac{1}{x} \right) \]

— defun acoth 0 —

(defun acoth (x) (atanh (/ 1 x)))

defun The inverse hyperbolic secant function

\[ asech(x) = \text{acosh} \left( \frac{1}{x} \right) \]

— defun asech 0 —

(defun asech (x) (acosh (/ 1 x)))
Chapter 71

OpenMath

71.1 A Technical Overview

OpenMath is a standard for representing mathematical data in as unambiguous a way as possible. It can be used to exchange mathematical objects between software packages or via email, or as a persistent data format in a database. It is tightly focussed on representing semantic information and is not intended to be used directly for presentation, although tools exist to facilitate this.

The original motivation for OpenMath came from the Computer Algebra community. Computer Algebra packages were getting bigger and more unwieldy, and it seemed reasonable to adopt a generic "plug and play" architecture to allow specialised programs to be used from general purpose environments. There were plenty of mechanisms for connecting software components together, but no common format for representing the underlying data objects. It quickly became clear that any standard had to be vendor-neutral and that objects encoded in OpenMath should not be too verbose. This has led to the design outlined below.

In 1998, the Worldwide Web Consortium (W3C) produced its first recommendation for the Extensible Markup Language (XML), intended to be a universal format for representing structured information on the worldwide web. It was swiftly followed by the first MathML recommendation which is an XML application oriented mainly towards the presentation (i.e. the rendering) of mathematical expressions.

The formal definition of OpenMath is contained within The OpenMath Standard and its accompanying documents, and the reader is referred there for more details.

The OpenMath Architecture

The OpenMath representation of a mathematical structure is referred to as an OpenMath object. This is an abstract structure which is represented concretely via an OpenMath encoding. These encoded objects are what an OpenMath application would read and write,
and in practice the OpenMath objects themselves almost never exist, except on paper. The advantage of this is that OpenMath is not tied to any one underlying mechanism: in the past we have used functional, SGML and binary encodings. The current favourite is XML, as described below, and we will tend to use XML notation when describing OpenMath objects (even though strictly speaking the XML representation is an encoding). OpenMath Objects

Formally, an OpenMath object is a labelled tree whose leaves are the basic OpenMath objects integers, IEEE double precision floats, unicode strings, byte arrays, variables or symbols. Of these, symbols are the most interesting since they consist of a name and a reference to a definition in an external document called a content dictionary (or CD). Using XML notation where the element name OMS indicates an OpenMath symbol, the following:

\[
\text{<OMS name="sin" cd="transc1"/>}
\]

represents the usual sine function, as defined in the CD "transc1". A basic OpenMath object is an OpenMath object, although its XML representation will be:

\[
\text{<OMOBJ>}
\text{  <OMS name="sin" cd="transc1"/>}
\text{</OMOBJ>}
\]

OpenMath objects can be built up recursively in a number of ways. The simplest is function application, for example the expression \(\sin(x)\) can be represented by the XML:

\[
\text{<OMOBJ>}
\text{  <OMA>}
\text{    <OMS name="sin" cd="transc1"/>}
\text{    <OMV name="x"/>}
\text{  </OMA>}
\text{</OMOBJ>}
\]

where OMV introduces a variable and OMA is the application element. Another straightforward method is attribution which as the name suggests can be used to add additional information (for example "the AXIOM command which generated me was ...") to an object without altering its fundamental meaning. More interesting are binding objects which are used to represent an expression containing bound variables, for example:

\[
\text{<OMOBJ>}
\text{  <OMA>}
\text{    <OMS cd="calculus1" name="int"/>}
\text{    <OMS cd="transc1" name="sin"/>}
\text{  </OMA>}
\text{</OMOBJ>}
\]

represents the integral of the sin function, but the encoding:

\[
\text{<OMOBJ>}
\text{  <OMA>}
\text{    <OMS cd="calculus1" name="int"/>}
\text{    <OMS cd="transc1" name="sin"/>}
\text{  </OMA>}
\text{</OMOBJ>}
\]
represents $\sin(x)dx$. This may appear overly complicated but it is useful, for example when searching in a database for expressions which match $\sin(y)dy$. The definition of a symbol in the CD specifies whether or not it may be used to bind variables, which is why

\[
\text{\textit{OMBIND}}
\]

\[
\text{\textit{OMA}} \quad \text{\textit{OMS name="sin" cd="transc1"/}}
\]

\[
\text{\textit{OMV name="x"/}}
\]

\[
\text{\textit{OMOBJ}}
\]

cannot be used as a binding symbol.

The final kind of OpenMath object is an error which is built up from a symbol describing the error and a sequence of OpenMath objects. For example:

\[
\text{\textit{OMOBJ}}
\]

\[
\text{\textit{OME}} \\
\text{\textit{OMS name="unexpected_symbol" cd="error1"}}
\]

\[
\text{\textit{OMS name="sine" cd="transc1"}}
\]

\[
\text{\textit{OMOBJ}}
\]

represents the error which might be generated when an application sees a symbol it doesn’t recognise from a CD it thought it knew about.

**OpenMath Encodings**

We have already seen some examples of the XML encoding, but it is by no means the only encoding. In the past there was a functional encoding (which looked like Lisp) and an SGML encoding which evolved into the current XML. Both of these are now obsolete, but there is still a binary encoding described in the standard, which is much more compact than the XML one.

In fact the XML encoding is not 100% XML. When XML was in its infancy the developers of OpenMath realised that it might become significant and decided to add some XML-like features to the SGML encoding so that an an OpenMath object could be encoded as valid XML. Thus it is currently the case that any well-formed OpenMath object encoded using the XML encoding as described in the standard is a valid XML document. However, if one uses standard XML tools to generate an OpenMath object in the XML encoding from the DTD given in chapter 4 of the standard, it is possible that the result will not be valid OpenMath,
although in practice this is highly unlikely. To cover all the possibilities allowed by XML would make it much more complicated to write an application to read any OpenMath object from scratch. Whether to adopt XML completely remains a hot topic of debate within the OpenMath community!

Generally speaking, it is not intended that the existing encodings should be readable by a human user or writable by hand. It is desirable that they be compact and it is also desirable that they be linear, but neither of these is a requirement. It is a property of encodings that it is possible to convert between them with no loss of information.

### Content Dictionaries

Content Dictionaries (or CDs for short) are the most important, and the most interesting, aspect of OpenMath because they define the meaning of the objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in an XML format. Defining the meaning of a symbol is not a trivial task, and even referring to well-known references can be fraught with pitfalls. Formal definitions and properties can be very useful but time-consuming to produce and verbose, not to mention difficult to get right. A symbol definition in an OpenMath CD consists of the following pieces of information:

- the symbol name;
- a description in plain text;
- optionally, a set of this symbol’s properties in plain text (Commented Mathematical Properties, or CMPs);
- optionally, a set of this symbol’s properties encoded in OpenMath (Formal Mathematical Properties, or FMPs);
- optionally, one or more examples of its use (encoded in OpenMath).

In practice the CMPs and FMPs can come as pairs, and often serve in the place of examples. A very simple instance of a CD definition is:

```xml
<CDDefinition>
  <Name> log </Name>
  <Description>
    This symbol represents a binary log function; the first argument is the base, to which the second argument is log’ed. It is defined in Abramowitz and Stegun, Handbook of Mathematical Functions, section 4.1
  </Description>
  <CMP>
    a^b = c implies log_a c = b
  </CMP>
  <FMP>
    <OMOBJ>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMA>
          <OMS cd="logic1" name="implies"/>
        </OMA>
      </OMA>
    </OMOBJ>
  </FMP>
</CDDefinition>
```
Another example would be to print the list

\[ [1, \frac{1}{2}] \]

as

<OMOBJ>
  <OMA>
    <OMS cd="list1" name="list"/>
    <OMI>1</OMI></OMA>
  <OMA>
    <OMS cd="nums1" name="rational"/>
    <OMI>1</OMI></OMA>
</OMOBJ>
This provides a symbol to represent the log function by giving a pointer to a standard reference book. It provides the property that:

\[ a^b = c \rightarrow \log_a(c) = b \]

both as plain text and as OpenMath, and also gives an example of how the symbol is used.

CDs usually consist of related symbols and collections of related CDs can be grouped together, for convenience, as CD Groups. One very important CD Group is that corresponding to the content part of MathML.

It is possible to associate extra information with CDs, in particular type information. Since there are many type systems available, each of which has its own strengths and advocates, the OpenMath community does not mandate any single system. Simple signatures can be encoded using the Simple Type System, while more formal definitions are possible using the Extended Calculus of Constructorss. Other associated information can include style sheets for rendering OpenMath symbols in MathML, and mathematical definitions to be used by formal logic systems.

Given the evolutionary nature of mathematics, it is clear that the set of CDs should be forever growing and never complete. Currently there are CDs for high-school mathematics, linear algebra, polynomials and group theory to name a few, and new contributions are always welcome. There is no requirement that applications use the standard set of CDs and it is often very useful to design a "private" CD for a specific purpose.

**OpenMath in Action**

There is no definitive way in which OpenMath should be used, as the protocol has been designed to be as flexible as possible. Nevertheless many OpenMath applications share common characteristics which we shall discuss here.

Suppose that we wish to have two applications communicating by sending OpenMath objects to each other, e.g. a client program and a computational server. It is unlikely that the internal data structures used by the applications will be OpenMath, and so translation between the internal representations and OpenMath (almost certainly OpenMath encodings rather than objects) will have to take place. The piece of software which does this is usually referred to as a phrase-book.

It is possible to write a generic phrase-book which can handle any piece of OpenMath, but applications where this makes sense are few and far between. In practice an OpenMath phrase book will usually only handle a fixed set of CDs (and hence a fixed set of symbols). What “handle” means will vary from case to case: a computer algebra system will usually try and evaluate its input and return a result or an error, while a typesetter will print its input according to some rendering rules and not return anything. OpenMath carefully avoids
defining what the “right” behaviour is in a given circumstance, and leaves that up to the phrase-book writer. Indeed it is quite possible that a piece of software could have multiple phrase-books associated with it for different purposes. OpenMath symbols should not be regarded as verbs since they are used to construct objects rather than to send commands, and the presence of both nouns and verbs in a CD (e.g. “integral” and “integrate”) is strongly discouraged.

Writing a phrase-book may be non-trivial, and requires an understanding of the semantics of the underlying software. An OpenMath object may not map directly into a private object and vice-versa, for example in some systems a rational number might have to be represented by a float, or a sparse matrix by a dense one.

The OpenMath standard includes a section on compliance, which describes the behaviour of an OpenMath application when certain errors occur. It also insists that all compliant software has the capability to use the XML encoding, to guarantee a degree of interoperability. This is an area where the standard is expected to evolve as more OpenMath applications become available.

71.2 Technical Details[3]

This chapter describes the Axiom implementation of the OpenMath project at INRIA [3]. The code enables the exchange of OpenMath objects between two processes and more generally the input and output of OpenMath objects. First we describe the library API and then we implement the functions used by Axiom.

71.3 The Structure of the API

The library and its API are logically structured in four parts:

- Functions that deal with devices, the abstraction from which OpenMath objects are read and written to.
- Functions that read from and write to OpenMath devices. These functions use a simple model that read and write tokens.
- Functions that create I/O structures to be used by devices, so that, for example, an OpenMath object can be read from a file or a socket. This part is extensible by the user.
- Functions that deal with interprocess communication.
71.4 OpenMath Expressions

Expressions

The library understands the following kinds of basic OpenMath expressions:

- integers
- double precision floating-point numbers (64 bits, following IEEE 754)
- byte arrays
- character strings
- symbols
- variables

and the four kinds of constructions:

- applications $e_0(e_1, \ldots e_n)$
- errors $s(e_1, \ldots e_n)$
- binders $e_1, (v_1, \ldots v_n), e_2$
- attributed expressions $[s_1e_1, \ldots s_ne_n]e$

where $e_i$ are OpenMath expressions, $v_i$ are OpenMath variables and $s$ and $s_i$ are OpenMath symbols.

Symbols

Symbols are constructed from a content dictionary (abbreviated as CD in the sequel) and a name. A content dictionary is identified by its name. The API permits the creation of any symbol in any content dictionary: there is nothing that prevents creating symbols that do not belong to a known CD.

Encoding and Decoding OpenMath Expressions

An OpenMath object is encoded as a sequence of bytes that is read and written sequentially. The library views this sequence as a stream of tokens. Expressions are linearized in a way that looks like Lisp with typed parenthesis. For example, the linearization of the application of $S$ to $E_1 \ldots E_n$ is:

- indicating that this is an application (a “begin application” token)
- linearizing $S$
linearizing $E_1, \ldots, E_n$,

- indicating that all arguments have been given (an “end application” token)

The other constructions are linearized the same way (each one with its own begin and end tokens). Note that there is no explicit arity indication so that we don’t have to introduce a special mechanism when we don’t know beforehand how many arguments there are.

To give attributes to an expression, the attributes and their associated values are put before the expression. To give the attributes $a_i$ with values $v_i$ (where $a_i$ are symbols and $v_i$ are OpenMath expressions) to an expression $E$ the process is:

- put a “begin attributed expression” token
- put a “begin attribute pairs” token
- put the symbol $a_1$ followed by the linearization of $v_1$ etc
- put an “end attribute pairs” token
- linearize $E$
- put an “end attributed expression” token

Decoding is done by first querying the type of the next OpenMath token and then invoking the right function to get this particular kind of token.

### 71.5 Big Integers

The library supports big integers that can potentially be given in various formats. The \texttt{OMBigIntType} describes the different possible formats.

```c
typedef enum OMbigIntType {
    OMBIunknown = 0, /* this is base 10, digits in normal order */
    OMBIbase10 /* this is base 16, digits in normal order (MSB) */
} OMbigIntType;
```

### 71.6 Functions Dealing with OpenMath Devices

OpenMath expressions are read and written through devices. Basically, an OpenMath device has an associated encoding and an I/O method. There are basically two encodings defined and implemented. The first one is a human readable and writable one that can be used for example as the encoding for sending OpenMath objects via e-mail or storing OpenMath objects to files. This encoding is SGML compatible in the sense that it can be used to represent OpenMath objects in SGML texts. It has an XML variant. The second encoding
is a binary one that can be used when compactness and speed of encoding and decoding is important. The encodings are defined by the `OMencodingType` type which is an enumerated type defined as

```c
typedef enum OMencodingType
{
  OMencodingUnknown,
  OMencodingBinary,
  OMencodingSGML,
  OMencodingXML
} OMencodingType;
```

`OMencodingUnknown` is to be used when creating a device that does not know which kind of encoding will be used. It must be used only for input devices.

A device is created with the following function, given an encoding and an appropriate I/O method:

- `OMdev OMmakeDevice(OMencodingType encoding, OMIO IO)`

Devices are closed with the following function

- `void OMcloseDevice(OMdev dev)`

Whether a device could be used both for reading and writing is entirely dependent on its I/O method.

The user can define its own I/O method as a function returning an `OMIO` object. This could enable him, for example, to use an existing transport protocol to exchange OpenMath expressions or to implement cut-and-paste of OpenMath expression by writing I/O structures that input and output to strings. The I/O section describes the available I/O structures in the library.

An `OMdev` object is a pointer to a structure that contains a lot of state. Almost all functions taking an `OMdev` object modify it. Likewise, an `OMIO` object carries a lot of state.

### 71.7 Functions to Write OpenMath Expressions to Devices

#### Beginning and Ending Objects

The following two functions mark the beginning and end of an OpenMath object.

- `OMstatus OMputObject(OMdev dev)`
- `OMstatus OMputEndObject(OMdev dev)`

These functions should be called before and after an OpenMath object in constructed in a device. In particular, the `OMputEndObject` function insures that the object has been completely written if any buffering was used.
71.7. FUNCTIONS TO WRITE OPENMATH EXPRESSIONS TO DEVICES

Writing Basic Objects

Basic OpenMath objects are written using these functions:

- OMstatus OMputInt32(OMdev dev, int n)
- OMstatus OMputBigInt(OMdev dev, const char *data, int len, int sign, OMbigIntType format)
- OMstatus OMputFloat64(OMdev dev, double *f)
- OMstatus OMputByteArray(OMdev dev, const char *data, int len)
- OMstatus OMputString(OMdev dev, const char *s)
- OMstatus OMputVar(OMdev dev, const char *name)
- OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name)

The char * arguments of OMputString, OMputVar and OMputSymbol are null-terminated strings. There are other functions that accept non null-terminated arrays of characters with their length. These are

- OMstatus OMputStringN(OMdev dev, const char *str, int len)
- OMstatus OMputVarN(OMdev dev, const char *var, int len)
- OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen, const char *name, int nlen)

The format for the data argument of the OMputBigInt function is given by format. When format is OMBIbase10, it is the sequence of character of its base 10 representation without sign (most significant digit first). The sign of the big integer is given by the sign argument that should be an integer greater or equal to zero for a positive integer and less than zero for a negative one. For example, the following line outputs the value of 20! to dev:

OMputBigInt(dev, "265252859812191058636308480000000", 33, 1, OMBIbase10);

Writing Structured Objects

The following functions are used to mark the beginning and end of the structured objects. They should be called in nested pairs, correctly bracketed:

- OMstatus OMputApp(OMdev dev)
- OMstatus OMputEndApp(OMdev dev)
- OMstatus OMputAttr(OMdev dev)
- OMstatus OMputEndAttr(OMdev dev)
OMstatus OMputBind(OMdev dev)
OMstatus OMputEndBind(OMdev dev)
OMstatus OMputBVar(OMdev dev)
OMstatus OMputEndBVar(OMdev dev)
OMstatus OMputAtp(OMdev dev)
OMstatus OMputEndAtp(OMdev dev)
OMstatus OMputError(OMdev dev)
OMstatus OMputEndError(OMdev dev)

Here is an example showing how to use these functions to output \( \sin x + y \), where \( x \) and \( y \) are represented as variables and \( \sin \) is the symbol whose name is \texttt{sin} in the \texttt{Basic} content dictionary. This can be done using the following sequence:

OMputObject(dev);
OMputApp(dev);
    OMputSymbol(dev, "Basic", "sin");
    OMputApp(dev)
        OMputSymbol(dev, "Basic", "+");
        OMputVar(dev, "x");
        OMputVar(dev, "y");
    OMputEndApp(dev);
OMputEndApp(dev);
OMputEndObject(dev);

71.8 Functions to Extract OpenMath Expressions from Devices

Testing the type of the current token

The first step in decoding an expression from a device is to call the \texttt{OMgetType} function

OMstatus OMgetType(OMdev dev, OMtokenType *type)

so that the correct function can be called to recover the current token. \texttt{OMgetType} returns via its \texttt{type} argument an \texttt{OMtokenType} object indicating the type of the next object to be read from the device. \texttt{OMtokenType} is an enumerated type defined as

typedef enum OMtokenType {
    OMtokenUnknown, /* error catching trick */
    OMtokenInt32,
71.8. FUNCTIONS TO EXTRACT OPENMATH EXPRESSIONS FROM DEVICES

OMtokenBigInt,
OMtokenFloat64,
OMtokenByteArray,
OMtokenVar,
OMtokenString,
OMtokenSymbol,
OMtokenComment,
OMtokenApp,  OMtokenEndApp,
OMtokenAttr,  OMtokenEndAttr,
OMtokenAtp,  OMtokenEndAtp,
OMtokenError, OMtokenEndError,
OMtokenObject, OMtokenEndObject,
OMtokenBind,  OMtokenEndBind,
OMtokenBVar,  OMtokenEndBVar,
} OMtokenType;

Note that the type of the current token can be tested multiple times. Two successive calls to OMgetType will always return the same result if no other OMget... function was called in between.

Extracting the current token

The following functions are used to read the basic OpenMath objects from devices:

- OMstatus OMgetInt32(OMdev dev, int *i)
- OMstatus OMgetFloat64(OMdev dev, double *d)
- OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)
- OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)
- OMstatus OMgetByteArray(OMdev dev, char **data, int *len)
- OMstatus OMgetByteArrayN(OMdev dev, char *data, int len)
- OMstatus OMgetString(OMdev dev, char **str)
- OMstatus OMgetStringN(OMdev dev, char *str, int len)
- OMstatus OMgetVar(OMdev dev, char **var)
- OMstatus OMgetVarN(OMdev dev, char *var, int len)
- OMstatus OMgetSymbol(OMdev dev, char **cd, char **name)
- OMstatus OMgetSymbolN(OMdev, char *cd, int clen, char *name, int nlen)
The functions that return variable size data exist in two versions. A simple version that does the necessary memory allocation itself (using `OMmalloc`) and a version (suffixied with `N`) that lets the user do the allocation itself. The size of the needed area can be determined with the following function:

- `int OMgetLength(OMdev dev)` returns the length of the next object.

that works for big integers, byte arrays, strings and variables. For symbols, the following function returns both the length of the content dictionary name and the length of the symbol name:

- `OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen)`

When the current token does not carry any data i.e. when `OMgetType` returns a marker, i.e. one of:

- `OMtokenApp`,
- `OMtokenEndApp`,
- `OMtokenAttr`,
- `OM tokenEndAttr`,
- `OMtokenAtp`,
- `OMtokenEndAtp`,
- `OMtokenError`,
- `OMtokenEndError`,
- `OMtokenObject`,
- `OMtokenEndObject`,
- `OMtokenBind`,
- `OMtokenEndBind`,
- `OMtokenBVar`
- `OMtokenEndBVar`

it is necessary to call the correct function to remove the marker. The available functions are

- `OMstatus OMgetObject(OMdev dev)`
- `OMstatus OMgetEndObject(OMdev dev)`
- `OMstatus OMgetApp(OMdev dev)`
71.8. FUNCTIONS TO EXTRACT OPENMATH EXPRESSIONS FROM DEVICES

- OMstatus OMgetEndApp(OMdev dev)
- OMstatus OMgetAttr(OMdev dev)
- OMstatus OMgetEndAttr(OMdev dev)
- OMstatus OMgetAtp(OMdev dev)
- OMstatus OMgetEndAtp(OMdev dev)
- OMstatus OMgetBind(OMdev dev)
- OMstatus OMgetEndBind(OMdev dev)
- OMstatus OMgetBVar(OMdev dev)
- OMstatus OMgetEndBVar(OMdev dev)
- OMstatus OMgetError(OMdev dev)
- OMstatus OMgetEndError(OMdev dev)

All the previous functions return OMsuccess when they succeed. When they return something else, there has been a problem such as calling the wrong function (OMgetApp when there is not a “beginning of application” mark) or a system error.

The sequence of calls to read an expression is thus completely similar (if we omit the calls to OMgetType) to the sequence of calls to write the expression. For example, the previous expression \((\sin x + y)\) can be recovered via the sequence:

```c
OMgetObject(dev);
OMgetApp(dev);
    OMgetSymbol(dev, ...);
    OMgetApp(dev);
    OMgetSymbol(dev, ...);
    OMgetVar(dev, ...);
    OMgetVar(dev, ...);
    OMgetEndApp(dev);
    OMgetEndApp(dev);
    OMgetEndObject(dev);
OMgetInt32(OMdev dev, int *i) returns the integer through its i argument.

OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)
returns the data corresponding to the big integer in data, its length in len, its sign in sign and its format in fmt.

OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)
copies the data corresponding to the big integer in `data` buffer that should be (at least) `len` characters long. The sign and format are returned in the `sign` and `fmt` arguments.

`OMgetByteArray(OMdev dev, char **data, int *len)` returns the byte array through its `data` argument. Its length is returned via the `len` argument.

`OMgetByteArrayN(OMdev dev, char *data, int len)` copies the byte array in the `data` buffer that should be (at least) `len` characters long.

`OMgetString(OMdev dev, char **str)` returns the string trough its `str` argument.

`OMgetStringN(OMdev dev, char *str, int len)` copies the string in the `str` buffer whose length should be (at least) `len`. If `len` is greater than the actual length of the string, a null character is added at the end of `str`.

`OMgetVar(OMdev dev, char **var)` returns the name of the variable (as a null-terminated string) in its `var` argument.

`OMgetVarN(OMdev dev, char *var, int len)` copies the name of the variable in the `var` buffer, whose length should be (at least) `len`. If `len` is greater than the actual length of the variable name, a null character is added at the end of `var`.

`OMgetSymbol(OMdev dev, char **cd, char **name)` returns the content dictionary and the name of the symbol through the `cd` and `name` arguments.

`OMgetSymbolN(OMdev dev, char *cd, int clen, char *name, int nlen)` copies the content dictionary and the name of the symbols in the `cd` and `name` buffers. `cd` should be at least `clen` character long and `name` should be at least `nlen` long. When there is enough room (based on `clen` or `nlen`) a null character is added after the last character of the name (`cd` or `name`).

### 71.9 Comments in the SGML/XML Encodings

The library can also output and read comments (SGML/XML comments) with the following functions:

- `OMstatus OMputComment(OMdev dev, char *comment)`
- `OMstatus OMputCommentN(OMdev dev, char *comment, int len)`
- `OMstatus OMgetComment(OMdev dev, char **comment)`
- `OMstatus OMgetCommentN(OMdev dev, char *comment, int len)`

By default, comments are silently ignored by the library when reading OpenMath objects (and writing them using the binary encoding). The function

- `OMbool OMignoreComment(OMdev dev, OMbool set)`

changes this behaviour. When called with `OMfalse`, comments are passed to the application: the `OMgetType` function will return `OMtokenComment` when the current token is a comment and the `OMgetComment` or `OMgetCommentN` functions should be used to get the comments. When `OMignoreComment` is called with `OMtrue`, comments are ignored.
71.10 I/O Functions for Devices

We provide four functions that produce OMIO objects for devices. These functions provide I/O through the stdio library (on FILE object), file descriptors and character strings.

- OMIO OMmakeIOFile(FILE *f) associates the device with the file pointer f.
- OMIO OMmakeIOfd(int fd) associates the device with the file descriptor fd.
- OMIO OMmakeIOHandle(HANDLE handle) associates the device with a file handle *Windows specific version of OMmakeIOfd().fd.
- OMIO OMmakeIOString(char **s) associates the device with a string.

For example, the following code opens a device that reads from standard input:

```c
dev = OMmakeDevice(OMencodingSGML, OMmakeIOFile(stdin));
```

The OMmakeI0String builds an input device that reads from a string or an output device that writes to a string. For input, s must point to a character string (null terminated). For output, s will point to a string allocated by the library (note that the string s points to can be reallocated by the library).

71.11 Communications

A communication layer can be put above the device layer. In fact, the I/O structure in a device provides all the necessary support to use any transmission or communication means. This library directly provides some connection-oriented, client-server facilities (based on TCP).

A set of functions are used to set up connections. Connections are described by the OMconn type. An OMconn is a (pointer to a) structure with two user-accessible fields in and out. in is a pointer to a device to be used for input. out is pointer to a device to be used for output. These devices use the binary encoding.

An OMconn object is made with the following function:

- OMconn OMmakeConn(int timeout)

where timeout is a timeout for the connection, expressed in milliseconds.

- OMdev OMconnIn(OMconn conn) returns the input device associated with the connection.
- OMdev OMconnOut(OMconn conn) returns the output device associated with the connection.
Functions to Initiate an OMconn

The functions we provide can be divided in two classes. The first one simply establishes an interprocess communication using IP addresses. The second one provides functions that can be used to launch a server. The addresses used are then generated by the library.

Simple Connections Functions

The following functions allow a client OpenMath application to contact an OpenMath server at a specified address:

- OMstatus OMconnTCP(OMconn conn, char *machine, int port)
- OMstatus OMconnUnix(OMconn conn, char *file)

These functions first physically establish the connection. Then, they enter negotiation with the server (they send the first message). When they return, the negotiation is finished and the devices in the conn argument are ready.

On the server side, the following functions provide bindings at specified addresses and take care of the negotiation:

- OMstatus OMbindTCP(OMconn conn, int port)
- OMstatus OMbindUnix(OMconn conn, char *file)

All four the previous functions block until the connection is established (and negotiation is over) or the timeout of the conn argument is reached.

The following function returns the file descriptor associated with a device. This is intended to be used when there is a need to poll the device (through the select or poll system calls).

- OMdeviceFd(OMdev dev)

Functions that Launch Servers

These functions provide the same functionalities for launching a server that were provided in the ASAP library.

In this model, the client calls OMlaunch with a machine name mach and a string cmd that is executed via rsh on machine mach as a shell command line. This command is supposed to launch the server program. The command is executed in an environment (in the UNIX sense) where some variables are associated with an address on the machine that runs the client. The server can then connect to the client with the OMserveClient function.

If the machine name is localhost, the command is started on the same machine (without calling rsh).

- OMstatus OMlaunchEnv(OMconn conn, char *machine, char *command, char *env)
71.12 Parameters

The library internally uses three functions that can be supplied by the user.

- `extern void *(*OMmalloc)(size_t size)`
  - `OMmalloc` is used for all memory allocations in the library. The default value is the `malloc` function.
- `extern void *(*OMrealloc)(void *ptr, size_t size)`
  - `OMrealloc` is used for reallocations. The default value is the `free` function.
- `extern void (*OMfree)(void *ptr)`
  - `OMfree` is used for deallocations. The default value is the `free` function.
- `OMfatal` is invoked when a fatal error is detected in the library (for example when memory allocation failed or when an inconsistency is detected in the library code data structures). The default value just does an `exit`.

`OMfatal` is declared as `extern void (*OMfatal)(OMstatus status)`. All memory allocations and deallocations in the library are done through the `OMmalloc` and `OMfree` functions.

71.13 Miscellaneous Functions and Variables

- `char *OMstatusToString(OMstatus status)` make a status into a human readable string.
• char *OMtokenTypeToString(OMtokenType ttype) makes a tokenType into a human readable string.

• OMencodingType OMgetDeviceEncoding(OMdev dev) returns the encoding actually used by the device.

• char *OMlibDynamicInfo(void)

• extern const char *OMlibVersion is the version of the library.

• extern const char *OMlibInfo contains some textual information about the library.

### 71.14 The OM.h header file

```c
#ifndef __OM_h__
#define __OM_h__

/*
 * All types used through API.
 */

/* These types are anonymized by the mean of a generic pointer.
 * You should not allocate or dereference objects of these types.
 * API (hopefully) provides you with all needed methods.
 * If you find any that are not included, please refer to
 * us rather than using private structures.
 * ie: If you need to do something like
 *     malloc(sizeof(OMdevStruct));
 * or
 *     OMdevStruct * pDev;
 *     pDev->anyField = something;
 * this probably means we need to discuss your problem.
 */

/* A device is an abstraction for put/get of OpenMath tokens */
typedef struct OMdevStruct *OMdev;

/* IO is a device field, (the physical IO channel) */
typedef struct OMIOStruct *OMIO;

/* Error status that may be returned */
typedef enum OMstatus {
    /* Last call was successful. */
    OMsuccess = 0,
    /* Last call failed for some undetermined reason. */
    OMfailed = 1,
    /* Last call failed for memory reasons. */
};
```
OMnoMem,
    /* Last call failed during some system call. */
OMerrorSys,
    /* Last call to some OMget* function failed due to an unexpected EOF
     on input IO. */
OMemptyIO,
    /* Last call to some OMget* function failed because there is no more
     token on device. */
OMnoMoreToken,
    /* Last call to some OMget* function timeouted. */
OMtimeoutedRead,
    /* Last call to some OMget* function failed due to malformed input.
     (this error covers all low level lexical or syntactic problems). */
OMmalformedInput,
    /* Last call to OMbindTCP failed because address is already in use
     (EADDRINUSE). */
OMaddrInUse,
    /* Last call to OMconnTCP failed to set connection. */
OMconnectFailed,
    /* Last call trigered some not (yet) implemented code in this lib. */
OMnotImplemented,
    /* Last call caused some internal trouble. */
OMinternalError
} OMstatus;

/* All OpenMath token kinds are identified by one of these types.*/
/* Values given in this enum have been chosen to:
 * - avoid conflicts with specific XML characters
 * - to help automatic detection of encoding type.
 *   (no: \t(9) \r(13) \n(10) <(60) or ' (32))
 * - keep some bits (3) available for special encodings purpose
 *   (eg: sharing or big len flags in binary encoding)
 */
typedef enum OMtokenType {
    OMtokenUnknown = 0, /* error catching trick */
    OMtokenInt32 = 1,
    OMtokenBigInt = 2,
    OMtokenFloat64 = 3,
    OMtokenByteArray = 4,
    OMtokenVar = 5,
    OMtokenString = 6,
    OMtokenWCString = 7,
    OMtokenSymbol = 8,
    OMtokenComment = 15,
    OMtokenApp = 16, OMtokenEndApp = 17,
    OMtokenAttr = 18, OMtokenEndAttr = 19,
    OMtokenAtp = 20, OMtokenEndAtp = 21,
    OMtokenError = 22, OMtokenEndError = 23,
    OMtokenObject = 24, OMtokenEndObject = 25,
    OMtokenBind = 26, OMtokenEndBind = 27,
typedef enum OMbigIntType {
    OMbigIntUnknown = 0,
    /* this is base 10, digits in normal order (MSB) */
    OMbigIntBase10,
    /* this is base 16, digits in normal order (MSB) */
    OMbigIntBase16
} OMbigIntType;

/* Encodings should not be "user visible"
 * We thus refer to encoding as "symbolic constants" from this enum type. */
typedef enum OMencodingType {
    /* You may set an input stream to "unknown encoding".
     * By doing this, you let library auto detect the
     * encoding type of the device during first token input. */
    OMencodingUnknown = 0,
    /* Binary encoding, more compact than XML one. */
    OMencodingBinary,
    /* XML-like encoding, human readable. */
    OMencodingXML
} OMencodingType;

/* This is a portable equivalent to wchar_t for unicode strings */
typedef unsigned short OMUCS2;

/* Replacement for lacking C bools */
typedef unsigned char OMbool;
#define OMfalse (0)
#define OMtrue (1)

/* Some global variables */

/* Version of this lib (eg: "1.0") */
extern const char *OMlibVersion;

/* Some textual information about this lib (eg: "debug is on") */
extern const char *OMlibInfo;

/* These pointers allow you to redefine memory management functions
   used in lib. */
extern void *(OMmalloc) (size_t size);
extern void *(OMrealloc) (void *ptr, size_t size);
extern void *(OMfree) (void *ptr);

/* If set, this function will be called by OMfatal, thus you may use it for
   error handling (by default it is set to exit()) */
extern void (*OMfatal) (OMstatus status);

/* Prototypes of OpenMath API */

extern OMstatus OMputInt32(OMdev dev, int n);
extern OMstatus OMputFloat64(OMdev dev, double *d);
extern OMstatus OMputBigInt(OMdev dev, const char *data, int len,
    int sign, OMbigIntType format);
extern OMstatus OMputByteArray(OMdev dev, const char *data, int len);
extern OMstatus OMputString(OMdev dev, const char *str);
extern OMstatus OMputStringN(OMdev dev, const char *str, int len);
/* OMputWCString*/
/* If you want to output plain 8bits C like strings there is no need*/
/* to use the OMputWCString* functions. This one is more efficient*/
/* (faster and more compact output for some encodings) */
extern OMstatus OMputString(OMdev dev, const char *str);
extern OMstatus OMputStringN(OMdev dev, const char *str, int len);
/* OMputWCString*/
/* If you are using wide char strings you need to output them*/
/* with that function rather than with OMputString.*/
/* (It takes endianess into account)*/
extern OMstatus OMputWCString(OMdev dev, const OMUCS2 * wcstr);
extern OMstatus OMputVar(OMdev dev, const char *var);
extern OMstatus OMputVarN(OMdev dev, const char *var, int len);
extern OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name);
extern OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen,
                           const char *name, int nlen);
extern OMstatus OMputApp(OMdev dev);
extern OMstatus OMputEndApp(OMdev dev);
extern OMstatus OMputAttr(OMdev dev);
extern OMstatus OMputEndAttr(OMdev dev);
extern OMstatus OMputAtp(OMdev dev);
extern OMstatus OMputEndAtp(OMdev dev);
extern OMstatus OMputBind(OMdev dev);
extern OMstatus OMputEndBind(OMdev dev);
extern OMstatus OMputBVar(OMdev dev);
extern OMstatus OMputEndBVar(OMdev dev);
extern OMstatus OMputObject(OMdev dev);
extern OMstatus OMputEndObject(OMdev dev);
extern OMstatus OMputError(OMdev dev);
extern OMstatus OMputEndError(OMdev dev);
extern OMstatus OMputComment(OMdev dev, const char *comment);
extern OMstatus OMputCommentN(OMdev dev, const char *comment, int len);

extern OMstatus OMgetType(OMdev dev, OMtokenType * type);
/* Get the type of the current token on device <dev>/
* dev: device to look at.
* type: where to store returned type.
* return: 0 or some error code */

extern OMstatus OMgetLength(OMdev dev, int *len);
/* Get the current token length.
* dev: device to read from
* len: where to put the token length
* the last ‘\0’ for string like tokens is not counted
* (rem: for WCString it is the number of bytes not the number of
* wide chars)
* return: 0 or some error code */

extern OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen);
/* Get the current token (wich is assumed to be a symbol) lengths.
* dev: device to read from
* clen: where to put the cd length (not counting the last ‘\0’)
* nlen: where to put the name length (not counting the last ‘\0’)
* return: 0 or some error code */

/* OMget* functions.
* They all take a device <dev> to get token from.
Some of them need more parameters to fill with the token content.
They are thoroughly documented in OpenMath Specification shipped with
the library.

return: a status that reflect the operation success.

/*
 * OMstatus OMgetInt32(OMdev dev, int *i);
 * OMstatus OMgetFloat64(OMdev dev, double *d);
 * OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign,
 * OMbigIntType * format);
 * OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign,
 * OMbigIntType * format);
 * OMstatus OMgetByteArrayList(OMdev dev, char **data, int *len);
 * OMstatus OMgetByteArrayListN(OMdev dev, char *data, int len);
 * OMstatus OMgetString(OMdev dev, char **str);
 * OMstatus OMgetStringN(OMdev dev, char *str, int len);
 * OMstatus OMgetWCString(OMdev dev, OMUCS2 ** wcstr);
 * OMstatus OMgetWCStringN(OMdev dev, OMUCS2 * wcstr, int len);
 * OMstatus OMgetVar(OMdev dev, char **var);
 * OMstatus OMgetVarN(OMdev dev, char *var, int len);
 * OMstatus OMgetSymbol(OMdev dev, char **cd, char **name);
 * OMstatus OMgetSymbolN(OMdev dev, char *cd, int clen, char *name,
 * int nlen);
 * OMstatus OMgetApp(OMdev dev);
 * OMstatus OMgetEndApp(OMdev dev);
 * OMstatus OMgetAttr(OMdev dev);
 * OMstatus OMgetEndAttr(OMdev dev);
 * OMstatus OMgetAtp(OMdev dev);
 * OMstatus OMgetEndAtp(OMdev dev);
 * OMstatus OMgetBind(OMdev dev);
 * OMstatus OMgetEndBind(OMdev dev);
 * OMstatus OMgetBVar(OMdev dev);
 * OMstatus OMgetEndBVar(OMdev dev);
 */
extern OMstatus OMgetObject(OMdev dev);
extern OMstatus OMgetEndObject(OMdev dev);
extern OMstatus OMgetError(OMdev dev);
extern OMstatus OMgetEndError(OMdev dev);
extern OMstatus OMgetComment(OMdev dev, char **comment);
extern OMstatus OMgetCommentN(OMdev dev, char *comment, int len);

/ * OMbeginObject 
 * Must be called before every new OpenMath object put.
 * (Not before every token!) 
 * dev: device where new object is to be put.
 * return: status describing operation success
 */
extern OMstatus OMbeginObject(OMdev dev);

/ * OMendObject 
 * Must be called after every OpenMath object put.
 * (Not after every token!) 
 * dev: device where object has been put.
 * return: status describing operation success
 */
extern OMstatus OMendObject(OMdev dev);

/ * OMignoreComment 
 * Set behavior of a device concerning comments.
 * (Comments on an input device may safely be ignored.)
 * dev: device to modify
 * set: If set == OMtrue then device will ignore incoming comments
 * If set == OMfalse then device will process incoming comments
 * like other tokens.
 * By default comments are ignored.
 * Whatever is <set> value, output of comments is always done.
 * return: previous value
 */
extern OMbool OMignoreComment(OMdev dev, OMbool set);

/ * OMtokenCount 
 * Reports the number of tokens that have been in/output on a device
 * dev: device to examine
 * inTokenNb: where to store number of input tokens (if not NULL)
 * outTokenNb: where to store number of output tokens (if not NULL)
 */
extern void OMtokenCount(OMdev dev, int *inTokenNb, int *outTokenNb);

/ * OMgetDeviceEncoding 
 * Get the current encoding used by a device
 * dev: device to examine
 * return: current encoding
 */
extern OMencodingType OMgetDeviceEncoding(OMdev dev);

/ * OMsetDeviceEncoding 
 * Set the encoding that will be used on a device
 * BEWARE: changing encoding on a device that has already been used
 * for IO is unsafe.
 * but setting encoding on a new device is safe.
71.14. THE OM.H HEADER FILE

* (in some occasions, it is not easy to know which encoding to
* use at device creation)
* dev: device to modify
* encoding: encoding to use
*/
extern void OMsetDeviceEncoding(OMdev dev, OMencodingType encoding);

/* OMmakeDevice
* Create a device from a low level IO
* Warning: "IO" should be a "generated" (new) structure as it contains some
* state that is private to the device. It is very dangerous for two devices
* to share the same "IO" structure.
* encoding: encoding scheme used by device
* IO: low level I/O support for device
* return: a newly allocated device
*/
extern OMdev OMmakeDevice(OMencodingType encoding, OMIO IO);

/* OMcloseDevice
* Close a device previously created with OMmakeDevice
* (embedded IO is closed too)
* dev: device to close
*/
extern void OMcloseDevice(OMdev dev);

/* OMmakeIOFd
* Create a low level IO object from a file descriptor.
* (May be used on socket for instance.)
* fd: file descriptor to wrap into the OpenMath IO object.
* return: a newly allocated IO object.
*/
extern OMIO OMmakeIOFd(int fd);

/* OMmakeIOFile
* Create a low level IO object from a FILE*.
* (May be used on stdin for instance.)
* fd: FILE* to wrap into the OpenMath IO object.
* return: a newly allocated IO object.
*/
extern OMIO OMmakeIOFile(FILE * f);

/* OMmakeIOString
* Create a low level IO object from a string (NUL terminator is not needed).
* (May be used for copy/paste for instance.)
* s: pointer to string to use into the OpenMath IO object.
* - In case of input device the string must be NUL terminated.
* - In case of output device string may be reallocated
* to fit size of outcoming objects.
* return: a newly allocated IO object.
*/
extern OMIO OMmakeIOString(char **s);

/* OMstatusToString
* Convert a status to a human readable string that explain its meaning
* status: status to explain
* return: corresponding string
extern char *OMstatusToString(OMstatus status);
/* OMtokenTypeToString 
 * Convert a tokenType to a human readable string 
 * ttype: type to convert
 * return: corresponding string 
 */
extern char *OMtokenTypeToString(OMtokenType ttype);
/* OMsetVerbosityLevel 
 * When using API some infos may be loged. 
 * This set the required verbosity level. 
 * level: level of verbosity. 
 * 0 means nothing is neither printed 
 * 1 everything is printed (default) 
 * 2,... less verbose 
 * return: last verbosity level 
 */
extern int OMsetVerbosityLevel(int level);
/* OMsetVerbosityOutput 
 * When using API some infos may be loged. 
 * This set the destination for logs. 
 * logFile: where to output logs (default is stderr) 
 * return: last output 
 */
extern FILE *OMsetVerbosityOutput(FILE * logFile);
/* OMlibDynamicInfo 
 * Gather some informations about lib that can’t be statically determined. 
 * Complete them with some relevant static information too. 
 * return: a newly allocated string 
 */
extern char *OMlibDynamicInfo(void);
/*** End Prototypes */
/* end of automaticaly updated part */
 ifdef WIN32
#include "windows.h"

/* OMmakeIOHandle 
 * Create a low level IO object from a widows handle. 
 * handle: windows handle to wrap into the OpenMath IO object. 
 * return: a newly allocated IO object. 
 */
extern OMIO OMmakeIOHandle(HANDLE handle);
extern void OMfreeIOHandle(OMIO io);
#endif
#endif /* OM_DEV */
/* The prototypes above are in fact collected from all these .h files */
#include "OMbase.h"
#include "OMdev.h"
71.15  Axiom OpenMath stub functions

These stub functions will eventually be expanded to handle OpenMath. See the OpenMath-Device domain in Volume 10.3. Note that the argument list for the Spad functions does not always match the argument list specified in the OpenMath specification.

There are 4 known OpenMath encodings which are set up in the OpenMathEncoding domain in Volume 10.3.

- Unknown
- Binary
- XML
- SGML

Axiom specific functions

This is used in OpenMathPackage in Volume 10.4.

(read OMdev)  ->  LispObject
(listCDs)     ->  List(String)
(listSymbols) ->  List(String)
(supportsCD cd)  ->  Boolean
(supportsSymbol cd name)  ->  Boolean

(defun om-Read

Read an OpenMath object from dev.

— defun om-Read —
(defun om-Read (dev)
  (declare (ignore dev)))

---

defun om-listCDs
Lists all of the CDs supported by Axiom.
  — defun om-listCDs —
(defun om-listCDs ())

---

defun om-listSymbols
Lists all the symbols in CD
  — defun om-listSymbols —
(defun om-listSymbols ())

---

defun om-supportsCD
Return true if Axiom supports this CD.
  — defun om-supportsCD —
(defun om-supportsCD (cd)
  (declare (ignore cd)))

---

defun om-supportsSymbol
  — defun om-supportsSymbol —
(defun om-supportsSymbol (cd name)
  (declare (ignore cd name)))

---
Lisp conversion functions

The Lisp conversion functions are:

- `(toDev LispObject) -> OMdev`
- `(fromDev OMdev) -> LispObject`
- `(toStatus LispObject) -> LispObject`
- `(fromStatus OMstatus) -> LispObject`
- `(toEncodingType LispObject) -> OMencodingType`
- `(fromEncodingType OMencodingType) -> LispObject`
- `(toBigNumStr LispObject) -> char *`
- `(fromBigNumStr char *, int, int, OMbigIntType) -> LispObject`
- `(toConn LispObject) -> OMconn`
- `(fromConn OMconn) -> LispObject`
- `(toCString LispObject) -> char **`
- `(fromCString char **) -> LispObject`
- `(lispStringFromCString LispObject) -> LispObject`
- `(cStringFromLispString LispObject) -> LispObject`

(defun om-setDevEncoding

This sets the encoding used for reading or writing OpenMath objects to or from dev to enc.

---

(defun om-setDevEncoding

(defun om-setDevEncoding (dev enc)
  (declare (ignore dev enc))
)

---

Device manipulation functions

- `(openFileDev LispObject, int, ...) -> LispObject`
- `(openStrDev LispObject, LispObject, LispObject) -> LispObject`
- `(closeDev LispObject, LispObject) -> LispObject`

(defun om-openFileDev

This opens file fname for reading or writing OpenMath objects. The mode can be "r" for read, "w" for write, or "a" for append.

---

(defun om-openFileDev (fname fmode enc)
  (declare (ignore fname fmode enc))
)
defun om-openStringDev
This opens the string str for reading and writing OpenMath objects in encoding enc.

(defun om-openStringDev (str enc)
  (declare (ignore str enc)))

——

defun om-closeDev
This closes dev, flushing output if necessary.

(defun om-closeDev (dev)
  (declare (ignore dev)))

——

Connection manipulation functions
These are covered in the OpenMathConnection domain in Volume 10.3.

(makeConn LispObject LispObject) -> LispObject
(closeConn LispObject) -> LispObject
(getConnInDev LispObject) -> LispObject
(getConnOutDev LispObject) -> LispObject

defun om-makeConn

(defun om-makeConn (conn)
  (declare (ignore conn)))

——

defun om-closeConn

(defun om-closeConn (conn)
  (declare (ignore conn)))

——
(defun om-closeConn (conn)
  (declare (ignore conn)))

---

defun om-getConnInDev

  — defun om-getConnInDev —

  (defun om-getConnInDev (conn)
    (declare (ignore conn)))

---

defun om-getConnOutDev

  — defun om-getConnOutDev —

  (defun om-getConnOutDev (conn)
    (declare (ignore conn)))

---

Client/Server functions

These are covered in the OpenMathConnection domain in Volume 10.3. See OMconn.h

(bindTCP  LispObject, LispObject, LispObject)  \rightarrow  LispObject
(connectTCP LispObject, int, ...)  \rightarrow  LispObject

defun om-bindTCP

  — defun om-bindTCP —

  (defun om-bindTCP (conn port)
    (declare (ignore conn port)))

---
defun om-connectTCP

— defun om-connectTCP —

(defun om-connectTCP (conn host port)
  (declare (ignore conn host port)))

Device input/output functions

Most of these functions are in the OpenMathDevice domain in Volume 10.3. The only exception seems to be the om-stringPtrToString and om-stringToStringPtr functions which are called in the domains that export primitives. Currently these are:

- Complex (10.3)
- DoubleFloat (10.3)
- Float (10.3)
- Fraction (10.3)
- Integer (10.3)
- List (10.3)
- SingleInteger (10.3)
- String (10.3)
- Symbol (10.3)
- ExpressionToOpenMath (10.4)
- OpenMathPackage (10.4)

Note that putSymbol2 is not implemented.

(getApp LispObject, LispObject) -> LispObject
(getAtp LispObject, LispObject) -> LispObject
(getAttr LispObject, LispObject) -> LispObject
(getBind LispObject, LispObject) -> LispObject
(getBVar LispObject, LispObject) -> LispObject
(getByteArray LispObject, LispObject) -> LispObject
(getEndApp LispObject, LispObject) -> LispObject
(getEndAtp LispObject, LispObject) -> LispObject
(getEndAttr LispObject, LispObject) -> LispObject
71.15. AXIOM OPENMATH STUB FUNCTIONS

(defun om-getApp
  (dev)
  (declare (ignore dev)))
defun om-getAtp
Reads a begin attribute pair token from dev.
— defun om-getAtp —

(defun om-getAtp (dev)
  (declare (ignore dev)))

———

defun om-getAttr
Reads a begin attribute token from dev
— defun om-getAttr —

(defun om-getAttr (dev)
  (declare (ignore dev)))

———

defun om-getBind
Reads a begin binder token from dev.
— defun om-getBind —

(defun om-getBind (dev)
  (declare (ignore dev)))

———

defun om-getBVar
Reads a begin bound variable list token from dev.
— defun om-getBVar —

(defun om-getBVar (dev)
  (declare (ignore dev)))

———
defun om-getByteArray
Reads a byte array from dev.

(defun om-getByteArray (dev))

defun om-getEndApp
Reads an end application token from dev.

(defun om-getEndApp (dev)
  (declare (ignore dev)))

defun om-getEndAtp
Reads an end attribute pair token from dev.

(defun om-getEndAtp (dev)
  (declare (ignore dev)))

defun om-getEndAttr
Reads an end attribute token from dev.

(defun om-getEndAttr (dev)
  (declare (ignore dev)))
defun om-getEndBind
Reads an end binder token from dev.
— defun om-getEndBind —
(defun om-getEndBind (dev)
(declare (ignore dev)))

defun om-getEndBVar
Reads an end bound variable list token from dev.
— defun om-getEndBVar —
(defun om-getEndBVar (dev)
(declare (ignore dev)))

defun om-getEndError
Reads an end error token from dev.
— defun om-getEndError —
(defun om-getEndError (dev)
(declare (ignore dev)))

defun om-getEndObject
Reads an end object token from dev.
— defun om-getEndObject —
(defun om-getEndObject (dev)
(declare (ignore dev)))
defun om-getError
Reads a begin error token from dev.
  — defun om-getError —
(defun om-getError (dev)
  (declare (ignore dev)))

defun om-getFloat
Reads a float from dev.
  — defun om-getFloat —
(defun om-getFloat (dev)
  (declare (ignore dev)))

defun om-getInt
Reads an integer from dev.
  — defun om-getInt —
(defun om-getInt (dev)
  (declare (ignore dev)))

defun om-getObject
Reads a begin object token from dev.
  — defun om-getObject —
(defun om-getObject (dev)
  (declare (ignore dev)))
defun om-getString
Reads a string from dev.

(defun om-getString (dev)
  (declare (ignore dev)))

defun om-getSymbol
Reads a symbol from dev.

(defun om-getSymbol (dev)
  (declare (ignore dev)))

defun om-getType
Returns the type of the next object on dev.

(defun om-getType (dev)
  (declare (ignore dev)))

defun om-getVar
Reads a variable from dev.

(defun om-getVar (dev)
  (declare (ignore dev)))
defun om-putApp

Writes a begin application token to dev.

---

(defun om-putApp (dev)
  (declare (ignore dev)))

---

defun om-putAtp

This writes a begin application pair token to dev.

---

(defun om-putAtp (dev)
  (declare (ignore dev)))

---

defun om-putAttr

This writes a begin attribute token to dev.

---

(defun om-putAttr (dev)
  (declare (ignore dev)))

---

defun om-putBind

This writes a begin binder token to dev.

---

(defun om-putBind (dev)
  (declare (ignore dev)))

---
defun om-putBVar
This writes a begin bound variable list token to dev.
— defun om-putBVar —
(defun om-putBVar (dev)
  (declare (ignore dev)))

defun om-putByteArray
This writes a byte array to dev.
— defun om-putByteArray —
(defun om-putByteArray (dev b)
  (declare (ignore dev b)))

defun om-putEndApp
This writes an end application token to dev.
— defun om-putEndApp —
(defun om-putEndApp (dev)
  (declare (ignore dev)))

defun om-putEndAtp
This writes an end attribute pair to dev.
— defun om-putEndAtp —
(defun om-putEndAtp (dev)
  (declare (ignore dev)))
defun om-putEndAttr
This writes an end attribute token to dev.

(defun om-putEndAttr (dev)
  (declare (ignore dev)))

---

defun om-putEndBind
This writes an end binder token to dev.

(defun om-putEndBind (dev)
  (declare (ignore dev)))

---

defun om-putEndBVar
This writes and end bound variable list token to dev

(defun om-putEndBVar (dev)
  (declare (ignore dev)))

---

defun om-putEndError
This writes an end error token to dev

(defun om-putEndError (dev)
  (declare (ignore dev)))

---
defun om-putEndObject
This writes an end object token to dev.
   — defun om-putEndObject —

(defun om-putEndObject (dev)
  (declare (ignore dev)))

---

defun om-putError
This writes a begin error token to dev.
   — defun om-putError —

(defun om-putError (dev)
  (declare (ignore dev)))

---

defun om-putFloat
This writes the float f to dev.
   — defun om-putFloat —

(defun om-putFloat (dev f)
  (declare (ignore dev f)))

---

defun om-putInt
This writes the integer i to dev
   — defun om-putInt —

(defun om-putInt (dev i)
  (declare (ignore dev i)))

---
defun om-putObject
This writes a begin object token to dev.
   — defun om-putObject —

(defun om-putObject (dev)
  (declare (ignore dev)))

\[\]

defun om-putString
This writes the string s to dev.
   — defun om-putString —

(defun om-putString (dev s)
  (declare (ignore dev s)))

\[\]

defun om-putSymbol
This writes the symbol nm using semantics from cd to dev.
   — defun om-putSymbol —

(defun om-putSymbol (dev cd nm)
  (declare (ignore dev cd nm)))

\[\]

defun om-putVar
This writes the variable v to dev.
   — defun om-putVar —

(defun om-putVar (dev v)
  (declare (ignore dev v)))

\[\]
defun om-stringToStringPtr

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the string from its address? It would appear to be a nop in lisp.

— defun om-stringToStringPtr —

(defun om-stringToStringPtr (str)
  (declare (ignore str)))

——

defun om-stringPtrToString

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the string address from a string? It would appear to be a nop in lisp.

— defun om-stringPtrToString —

(defun om-stringPtrToString (str)
  (declare (ignore str)))

——
Chapter 72

NRLIB code.lisp support code

defun makeByteWordVec2

    — defun makeByteWordVec2 0 —

(defun makeByteWordVec2 (maxelement initialvalue)
  (let ((n (cond ((null initialvalue) 7) ('t maxelement))))
    (make-array (length initialvalue)
      :element-type (list 'mod (1+ n))
      :initial-contents initialvalue)))

---

defmacro spadConstant

    — defmacro spadConstant 0 —

(defmacro spadConstant (dollar n)
  '(spadcall (svref ,dollar (the fixnum ,n))))

---

1155
Chapter 73

Monitoring execution

This file contains a set of function for monitoring the execution of the functions in a file. It constructs a hash table that contains the function name as the key and monitor-data structures as the value.

The technique is to use a :cond parameter on trace to call the monitor-incr function to incr the count every time a function is called.

*monitor-table* HASH TABLE
     is the monitor table containing the hash entries
*monitor-nrlibs* LIST of STRING
     list of nrlib filenames that are monitored
*monitor-domains* LIST of STRING
     list of domains to monitor-report (default is all exposed domains)
monitor-data STRUCTURE
     is the defstruct name of records in the table
     name is the first field and is the name of the monitored function
     count contains a count of times the function was called
     monitorp is a flag that skips counting if nil, counts otherwise
     sourcefile is the name of the file that contains the source code

***** SETUP, SHUTDOWN ****

monitor-inittable () FUNCTION
     creates the hashtable and sets *monitor-table*
     note that it is called every time this file is loaded
monitor-end () FUNCTION
     unhooks all of the trace hooks

***** TRACE, UNTRACE *****
monitor-add (name &optional sourcefile) FUNCTION
  sets up the trace and adds the function to the table
monitor-delete (fn) FUNCTION
  untraces a function and removes it from the table
monitor-enable (&optional fn) FUNCTION
  starts tracing for all (or optionally one) functions that
  are in the table
monitor-disable (&optional fn) FUNCTION
  stops tracing for all (or optionally one) functions that
  are in the table

***** COUNTING, RECORDING *****

monitor-reset (&optional fn) FUNCTION
  reset the table count for the table (or optionally, for a function)
monitor-incr (fn) FUNCTION
  increments the count information for a function
  it is called by trace to increment the count
monitor-decr (fn) FUNCTION
  decrements the count information for a function
monitor-info (fn) FUNCTION
  returns the monitor-data structure for a function

***** FILE IO *****

monitor-write (items file) FUNCTION
  writes a list of symbols or structures to a file
monitor-file (file) FUNCTION
  will read a file, scan for defuns, monitor each defun
  NOTE: monitor-file assumes that the file has been loaded

***** RESULTS *****

monitor-results () FUNCTION
  returns a list of the monitor-data structures
monitor-untested () FUNCTION
  returns a list of files that have zero counts
monitor-tested (&optional delete) FUNCTION
  returns a list of files that have nonzero counts
  optionally calling monitor-delete on those functions

***** CHECKPOINT/RESTORE *****

monitor-checkpoint (file) FUNCTION
  save the *monitor-table* in a loadable form
monitor-restore (file) FUNCTION
  restore a checkpointed file so that everything is monitored

***** ALGEBRA *****

monitor-autoload () FUNCTION
  traces autoload of algebra to monitor corresponding source files
NOTE: this requires the /spad/int/algebra directory

monitor-dirname (args) {FUNCTION}
  expects a list of 1 libstream (loadvol's arglist) and monitors the source
  this is a function called by monitor-autoload

monitor-nrlib (nrlib) {FUNCTION}
  takes an nrlib name as a string (eg POLY) and returns a list of
  monitor-data structures from that source file

monitor-report () {FUNCTION}
  generate a report of the monitored activity for domains in
  *monitor-domains*

monitor-spadfile (name) {FUNCTION}
  given a spad file, report all nrlibs it creates
  this adds each nrlib name to *monitor-domains* but does not
  trace the functions from those domains

monitor-percent () {FUNCTION}
  ratio of (functions executed)/(functions traced)

monitor-apropos (str) {FUNCTION}
  given a string, find all monitored symbols containing the string
  the search is case-insensitive. returns a list of monitor-data items

for example:
suppose we have a file "/u/daly/testmon.lisp" that contains:

(defun foo1 () (print 'foo1))
(defun foo2 () (print 'foo2))
(defun foo3 () (foo1) (foo2) (print 'foo3))
(defun foo4 () (print 'foo4))

an example session is:

; FIRST WE LOAD THE FILE (WHICH INITs *monitor-table*)

>(load "/u/daly/monitor.lisp")
Loading /u/daly/monitor.lisp
Finished loading /u/daly/monitor.lisp
T

; SECOND WE LOAD THE TESTMON FILE
>(load "/u/daly/testmon.lisp")
T

; THIRD WE MONITOR THE FILE
>(monitor-file "/u/daly/testmon.lisp")
monitoring "/u/daly/testmon.lisp"
NIL

; FOURTH WE CALL A FUNCTION FROM THE FILE (BUMP ITS COUNT)
>(foo1)

FOO1

FOO1
; AND ANOTHER FUNCTION (BUMP ITS COUNT)
>(foo2)

FOO2

; AND A THIRD FUNCTION THAT CALLS THE OTHER TWO (BUMP ALL THREE)
>(foo3)

FOO1
FOO2
FOO3
FOO3

; CHECK THAT THE RESULTS ARE CORRECT

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 2 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 1 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")

; STOP COUNTING CALLS TO FOO2

>(monitor-disable 'foo2)
NIL

; INVOKE FOO2 THRU FOO3

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NOTICE THAT FOO1 AND FOO3 WERE BUMPED BUT NOT FOO2

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
"/u/daly/testmon.lisp")

; TEMPORARILY STOP ALL MONITORING

>(monitor-disable)
NIL

; CHECK THAT NOTHING CHANGES

>(foo3)
FOO1
FOO2
FOO3
FOO3

; NO COUNT HAS CHANGED

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
   "/u/daly/testmon.lisp")

; MONITOR ONLY CALLS TO FOO1

>(monitor-enable 'foo1)
T

; FOO3 CALLS FOO1

>(foo3)
FOO1
FOO2
FOO3
FOO3

; FOO1 HAS CHANGED BUT NOT FOO2 OR FOO3

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 4 MONITORP T SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
CHAPTER 73. MONITORING EXECUTION

"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
   "/u/daly/testmon.lisp")

; MONITOR EVERYBODY

>(monitor-enable)
NIL

; CHECK THAT EVERYBODY CHANGES

>(foo3)
FOO1
FOO2
FOO3
FOO3

; EVERYBODY WAS BUMPED

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
  "/u/daly/testmon.lisp")

; WHAT FUNCTIONS WERE TESTED?

>(monitor-tested)
(FOO1 FOO2 FOO3)

; WHAT FUNCTIONS WERE NOT TESTED?

>(monitor-untested)
(FOO4)

; UNTRACE THE WHOLE WORLD, MONITORING CANNOT RESTART

>(monitor-end)
NIL

; CHECK THE RESULTS

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")

; CHECK THAT THE FUNCTIONS STILL WORK

> (foo3)
FOO1
FOO2
FOO3
FOO3

; CHECK THAT MONITORING IS NOT OCCURRING

> (monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")

defvar $*monitor-domains*

— initvars —
(defvar *monitor-domains* nil "a list of domains to report")

-----

defvar $*monitor-nrlibs*

— initvars —
(defvar *monitor-nrlibs* nil "a list of nrlibs that have been traced")

-----
defvar $*monitor-table*

— initvars —

(defvar *monitor-table* nil "a table of all of the monitored data")

———

— postvars —

(eval-when (eval load)
  (unless *monitor-table* (monitor-inittable)))

———

defstruct $monitor-data

— initvars —

(defstruct monitor-data name count monitorp sourcefile)

———

defstruct $libstream

— initvars —

(defstruct libstream mode dirname (indextable nil) (indexstream nil))

———

defun Initialize the monitor statistics hashtable

[*monitor-table* p1164]

— defun monitor-inittable 0 —
(defun monitor-init-table ()
 "initialize the monitor statistics hashtable"
 (declare (special *monitor-table*))
 (setq *monitor-table* (make-hash-table)))

(defun End the monitoring process, we cannot restart
 [*monitor-table* p1164]
 — defun monitor-end 0 —

(defun monitor-end ()
 "End the monitoring process, we cannot restart"
 (declare (special *monitor-table*))
 (maphash
   #'(lambda (key value)
       (declare (ignore value))
       (eval `(untrace ,key)))
   *monitor-table*))

(defun Return a list of the monitor-data structures
 [*monitor-table* p1164]
 — defun monitor-results 0 —

(defun monitor-results ()
 "return a list of the monitor-data structures"
 (let (result)
   (declare (special *monitor-table*))
   (maphash
     #'(lambda (key value)
         (declare (ignore key))
         (push value result))
     *monitor-table*)
   (mapcar #'(lambda (x) (pprint x))
     (sort result #'string-lessp :key #'monitor-data-name))))
defun Add a function to be monitored

(defun monitor-add (name &optional sourcefile)
  "add a function to be monitored"
  (declare (special *monitor-table*))
  (unless (fboundp name) (load sourcefile))
  (when (gethash name *monitor-table*)
    (monitor-delete name))
  (eval `(trace (,name :cond (progn (monitor-incr ',name) nil)))))
  (setf (gethash name *monitor-table*)
    (make-monitor-data
     :name name :count 0 :monitorp t :sourcefile sourcefile)))

---

defun Remove a function being monitored

(defun monitor-delete (fn)
  "Remove a function being monitored"
  (declare (special *monitor-table*))
  (eval `(untrace ,fn))
  (remhash fn *monitor-table*))

---

defun Enable all (or optionally one) function for monitoring

(defun monitor-enable (&optional fn)
  "enable all (or optionally one) function for monitoring"
  (declare (special *monitor-table*))
  (if fn
(progn
  (eval '(trace ,('fn :cond (progn (monitor-incr ',fn) nil))))
  (setf (monitor-data-monitorp (gethash fn *monitor-table*)) t))
(maphash
  #'(lambda (key value)
      (declare (ignore value))
      (eval `(trace ,key :cond (progn (monitor-incr ',key) nil))))
      (setf (monitor-data-monitorp (gethash key *monitor-table*)) t))
    *monitor-table*)))

—
defun Disable all (optionally one) function for monitoring
[*monitor-table* p1164]

  — defun monitor-disable 0 —

(defun monitor-disable (&optional fn)
  "disable all (optionally one) function for monitoring"
  (declare (special *monitor-table*))
  (if fn
   (progn
     (eval '(untrace ,fn))
     (setf (monitor-data-monitorp (gethash fn *monitor-table*)) nil))
     (maphash
       #'(lambda (key value)
           (declare (ignore value))
           (eval '(untrace ,key))
           (setf (monitor-data-monitorp (gethash key *monitor-table*)) nil))
     *monitor-table*)))

—
defun Reset the table count for the table (or a function)
[*monitor-table* p1164]

  — defun monitor-reset 0 —

(defun monitor-reset (&optional fn)
  "reset the table count for the table (or a function)"
  (declare (special *monitor-table*))
  (if fn
   (setf (monitor-data-count (gethash fn *monitor-table*)) 0)
(maphash
  #'(lambda (key value)
     (declare (ignore value))
     (setf (monitor-data-count (gethash key *monitor-table*)) 0)
    *monitor-table*)))

defun Incr the count of fn by 1
[*monitor-table* p1164]

— defun monitor-incr 0 —

(defun monitor-incr (fn)
"incr the count of fn by 1"
(let (data)
  (declare (special *monitor-table*))
  (setq data (gethash fn *monitor-table*))
  (if data
     (incf (monitor-data-count data)) ;; change table entry by side-effect
     (warn "~s is monitored but not in table..do (untrace "~s" fn fn)))
)

defun Decr the count of fn by 1
[*monitor-table* p1164]

— defun monitor-decr 0 —

(defun monitor-decr (fn)
"decr the count of fn by 1"
(let (data)
  (declare (special *monitor-table*))
  (setq data (gethash fn *monitor-table*))
  (if data
     (decf (monitor-data-count data)) ;; change table entry by side-effect
     (warn "~s is monitored but not in table..do (untrace "~s" fn fn)))
)
defun Return the monitor information for a function

[*monitor-table* p1164]

— defun monitor-info 0 —

(defun monitor-info (fn)
 "return the monitor information for a function"
 (declare (special *monitor-table*))
 (gethash fn *monitor-table*))

defun Hang a monitor call on all of the defuns in a file

[done p??]
[done p??]
[monitor-add p1166]

— defun monitor-file 0 —

(defun monitor-file (file)
 "hang a monitor call on all of the defuns in a file"
 (let (expr (package "BOOT"))
 (format t "monitoring ~s~%" file)
 (with-open-file (in file)
 (catch 'done
 (loop
 (setq expr (read in nil 'done))
 (when (eq expr 'done) (throw 'done nil))
 (if (and (consp expr) (eq (car expr) 'in-package))
 (if (and (consp (second expr)) (eq (first (second expr)) 'quote))
 (setq package (string (second (second expr))))
 (setq package (second expr)))
 (when (and (consp expr) (eq (car expr) 'defun))
 (monitor-add (intern (string (second expr)) package) file)))))))

defun Return a list of the functions with zero count fields

[*monitor-table* p1164]

— defun monitor-untested 0 —
(defun monitor-untested ()
  "return a list of the functions with zero count fields"
  (let (result)
    (declare (special *monitor-table*))
    (maphash
     #'(lambda (key value)
         (if (and (monitor-data-monitorp value) (= (monitor-data-count value) 0))
           (push key result)))
     *monitor-table*)
    (sort result #'string-lessp )))

defun Return a list of functions with non-zero counts

[monitor-delete p1166]
[*monitor-table*) p??]

— defun monitor-tested 0 —

(defun monitor-tested (&optional delete)
  "return a list of functions with non-zero counts, optionally deleting them"
  (let (result)
    (declare (special *monitor-table*))
    (maphash
     #'(lambda (key value)
         (when (and (monitor-data-monitorp value)
                   (> (monitor-data-count value) 0))
           (when delete (monitor-delete key))
           (push key result)))
     *monitor-table*)
    (sort result #'string-lessp)))

— defun monitor-write 0 —

(defun monitor-write (items file)
  "write out a list of symbols or structures to a file"
  (with-open-file (out file :direction :output)
    (dolist (item items)
      (if (symbolp item)

    (dolist (item items)
      (if (symbolp item)

    (dolist (item items)
      (if (symbolp item)

    (dolist (item items)
      (if (symbolp item)

    (dolist (item items)
      (if (symbolp item)
(format out "'-s'-%' item)
(format out "'-50t'-s'-100t'-s'-%'"  
 (monitor-data-sourcefile item)
 (monitor-data-name item)
 (monitor-data-count item)))))

defun Save the *monitor-table* in loadable form

[*monitor-table* p1164]
[*print-package* p??]

— defun monitor-checkpoint 0 —

(defun monitor-checkpoint (file)
 "save the *monitor-table* in loadable form"
(let ((*print-package* t))
 (declare (special *print-package* *monitor-table*))
 (with-open-file (out file :direction :output)
 (format out "(in-package "BOOT")\n")
 (format out "(monitor-inittable)\n")
 (dolist (data (monitor-results))
 (format out "(monitor-add 's 's 's 's)\n"
 (monitor-data-name data)
 (monitor-data-sourcefile data))
 (format out "(setf (gethash 's *monitor-table*)
 "make-monitor-data :name 's :count 's :monitorp 's
 :sourcefile 's)\n")
 (monitor-data-name data)
 (monitor-data-name data)
 (monitor-data-count data)
 (monitor-data-monitorp data)
 (monitor-data-sourcefile data)))))

— defun restore a checkpointed file —

(defun monitor-restore (file)
 "restore a checkpointed file"
(load file))
defun Printing help documentation

— defun monitor-help 0 —

(defun monitor-help ()
  (format t ";; MONITOR
  ;; This file contains a set of function for monitoring the execution
  ;; of the functions in a file. It constructs a hash table that contains
  ;; the function name as the key and monitor-data structures as the value
  ;; The technique is to use a :cond parameter on trace to call the
  ;; monitor-incr function to incr the count every time a function is called
  ;; *monitor-table* HASH TABLE
  ;; is the monitor table containing the hash entries
  ;; *monitor-nrlibs* LIST of STRING
  ;; list of nrlib filenames that are monitored
  ;; *monitor-domains* LIST of STRING
  ;; list of domains to monitor-report (default is all exposed domains)
  ;; monitor-data STRUCTURE
  ;; is the defstruct name of records in the table
  ;; name is the first field and is the name of the monitored function
  ;; count contains a count of times the function was called
  ;; monitorp is a flag that skips counting if nil, counts otherwise
  ;; sourcefile is the name of the file that contains the source code
  ;; ***** SETUP, SHUTDOWN *****
  ;; monitor-inittable () FUNCTION
  ;; creates the hashtable and sets *monitor-table*
  ;; note that it is called every time this file is loaded
  ;; monitor-end () FUNCTION
  ;; unhooks all of the trace hooks
  ;; ***** TRACE, UNTRACE *****
  ;; monitor-add (name &optional sourcefile) FUNCTION
  ;; sets up the trace and adds the function to the table
  ;; monitor-delete (fn) FUNCTION
  ;; untraces a function and removes it from the table
  ;; monitor-enable (&optional fn) FUNCTION
  ;; starts tracing for all (or optionally one) functions that
  ;; are in the table
  ;; monitor-disable (&optional fn) FUNCTION

  ;; ;;; monitor-help 0
  ;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ..
; ; stops tracing for all (or optionally one) functions that
; ; are in the table
; ;
; ;; ***** COUNTING, RECORDING *****
; ;
; ;; monitor-reset (&optional fn) FUNCTION
; ;; reset the table count for the table (or optionally, for a function)
; ;; monitor-incr (fn) FUNCTION
; ;; increments the count information for a function
; ;; it is called by trace to increment the count
; ;; monitor-decr (fn) FUNCTION
; ;; decrements the count information for a function
; ;; monitor-info (fn) FUNCTION
; ;; returns the monitor-data structure for a function
; ;
; ;; ***** FILE IO *****
; ;
; ;; monitor-write (items file) FUNCTION
; ;; writes a list of symbols or structures to a file
; ;; monitor-file (file) FUNCTION
; ;; will read a file, scan for defuns, monitor each defun
; ;; NOTE: monitor-file assumes that the file has been loaded
; ;
; ;; ***** RESULTS *****
; ;
; ;; monitor-results () FUNCTION
; ;; returns a list of the monitor-data structures
; ;; monitor-untested () FUNCTION
; ;; returns a list of files that have zero counts
; ;; monitor-tested (&optional delete) FUNCTION
; ;; returns a list of files that have nonzero counts
; ;; optionally calling monitor-delete on those functions
; ;
; ;; ***** CHECKPOINT/RESTORE *****
; ;
; ;; monitor-checkpoint (file) FUNCTION
; ;; save the *monitor-table* in a loadable form
; ;; monitor-restore (file) FUNCTION
; ;; restore a checkpointed file so that everything is monitored
; ;
; ;; ***** ALGEBRA *****
; ;
; ;; monitor-autoload () FUNCTION
; ;; traces autoload of algebra to monitor corresponding source files
; ;; NOTE: this requires the /spad/int/algebra directory
; ;; monitor-dirname (args) FUNCTION
; ;; expects a list of 1 libstream (loadvol's argc) and monitors the source
; ;; this is a function called by monitor-autoload
; ;; monitor-nrlib (nrlib) FUNCTION
; ;; takes an nrlib name as a string (eg POLY) and returns a list of
Monitor algebra files

defun Monitoring algebra code.lsp files

[*monitor-nrlibs* p1163]

— defun monitor-dirname 0 —

(defun monitor-dirname (args)
  "expects a list of 1 libstream (loadvol’s arglist) and monitors the source"
  (let (name)
    (declare (special *monitor-nrlibs*))
    (setq name (libstream-dirname (car args)))
    (setq name (file-namestring name))
    (setq name (concatenate 'string "/spad/int/algebra/" name "/code.lsp"))
    (when (probe-file name)
      (push name *monitor-nrlibs*)
      (monitor-file name))))

[— — —]

defun Monitor autoloaded files

— defun monitor-autoload 0 —

(defun monitor-autoload ()
  "given a spad file, report all nrlibs it creates"
  "this adds each nrlib name to *monitor-domains* but does not trace the functions from those domains"
  "ratio of (functions executed)/(functions traced)"
  "given a string, find all monitored symbols containing the string"
  "the search is case-insensitive. returns a list of monitor-data items"
) nil)
"traces autoload of algebra to monitor corresponding source files"
(trace (vmlisp::loadvol
 :entrycond nil
 :exitcond (progn (monitor-dirname system::arglist) nil)))

---

defun Monitor an nrlib

[*monitor-table* p1164]

— defun monitor-nrlib 0 —

(defun monitor-nrlib (nrlib)
 "takes an nrlib name as a string (eg POLY) and returns a list of
 monitor-data structures from that source file"
 (let (result)
   (declare (special *monitor-table*))
   (maphash
    #'(lambda (k v)
       (declare (ignore k))
       (when (string= nrlib
          (pathname-name (car (last
            (pathname-directory (monitor-data-sourcefile v)))))))
       (push v result)))
   *monitor-table*)
   result))

---

defun Given a monitor-data item, extract the nrlib name

— defun monitor-libname 0 —

(defun monitor-libname (item)
 "given a monitor-data item, extract the nrlib name"
 (pathname-name (car (last
   (pathname-directory (monitor-data-sourcefile item))))))

---
defun Is this an exposed algebra function?

— defun monitor-exposedp 0 —

(defun monitor-exposedp (fn)
"exposed functions have more than 1 semicolon. given a symbol, count them"
(> (count #\; (symbol-name fn)) 1))

———

defun Monitor exposed domains

TPDHERE: note that the file interp.exposed no longer exists. The exposure information is now in bookvol5. This needs to work off the internal exposure list, not the file.
[done p??]
[done p??]
[*monitor-domains* p1163]

— defun monitor-readinterp 0 —

(defun monitor-readinterp ()
"read interp.exposed to initialize *monitor-domains* to exposed domains. this is the default action. adding or deleting domains from the list will change the report results"
(let (skip expr name)
  (declare (special *monitor-domains*))
  (setq *monitor-domains* nil)
  (with-open-file (in "/spad/src/algebra/interp.exposed")
    (read-line in)
    (read-line in)
    (read-line in)
    (read-line in)
    (catch 'done
      (loop
        (setq expr (read-line in nil "done"))
        (when (string= expr "done") (throw 'done nil))
        (cond
          ((string= expr "basic") (setq skip nil))
          ((string= expr "categories") (setq skip t))
          ((string= expr "hidden") (setq skip t))
          ((string= expr "defaults") (setq skip nil)))
        (when (and (not skip) (> (length expr) 58))
          (setq name (subseq expr 58 (length expr)))
          (setq name (string-right-trim ’(#\space) name))
          (when (> (length name) 0)
            (push name *monitor-domains*))))))))
defun Generate a report of the monitored domains

[monitor-readinterp p1176]
[*monitor-domains* p1163]

---

(defun monitor-report 0 )
"generate a report of the monitored activity for domains in *monitor-domains*"
(let (nrlibs nonzero total)
  (declare (special *monitor-domains*))
  (unless *monitor-domains* (monitor-readinterp))
  (setq nonzero 0)
  (setq total 0)
  (maphash
    #'(lambda (k v)
      (declare (ignore k))
      (let (nextlib point)
        (when (> (monitor-data-count v) 0) (incf nonzero))
        (incf total)
        (setq nextlib (monitor-libname v))
        (setq point (member nextlib nrlibs :test #'string= :key #'car))
        (if point
          (setf (cdr (first point)) (cons v (cdr (first point))))
          (push (cons nextlib (list v)) nrlibs)))
    *monitor-table*)
  (format t "~d of ~d (~d percent) tested~%" nonzero total
    (round (/ (* 100.0 nonzero) total)))
  (setq nrlibs (sort nrlibs #'string< :key #'car))
  (dolist (pair nrlibs)
    (let ((exposedcount 0) (testcount 0))
      (when (member (car pair) *monitor-domains* :test #'string=)
        (format t "for library "s" %" (car pair))
        (dolist (item (sort (cdr pair) #'> :key #'monitor-data-count))
          (when (monitor-exposedp (monitor-data-name item))
            (incf exposedcount)
            (when (> (monitor-data-count item) 0) (incf testcount))
            (format t "~d ~s" (monitor-data-count item)
              (monitor-data-name item)))))
      (if (= exposedcount testcount)
        (format t "a has all exposed functions tested"%" (car pair))
        (format t "Daly bug: a has untested exposed functions"%" (car pair)))))))
nil)
defun Parse an )abbrev expression for the domain name

— defun monitor-parse 0 —

(defun monitor-parse (expr)
  (let (point1 point2)
    (setq point1 (position #\space expr :test #'char=))
    (setq point1 (position #\space expr :start point1 :test-not #'char=))
    (setq point1 (position #\space expr :start point1 :test #'char=))
    (setq point1 (position #\space expr :start point1 :test-not #'char=))
    (setq point2 (position #\space expr :start point1 :test #'char=))
    (subseq expr point1 point2)))

defun Given a spad file, report all nrlibs it creates

[done p??]
[done p??]
[monitor-parse p1178]
[*monitor-domains* p1163]

— defun monitor-spadfile 0 —

(defun monitor-spadfile (name)
  "given a spad file, report all nrlibs it creates"
  (let (expr)
    (declare (special *monitor-domains*))
    (with-open-file (in name)
      (catch 'done
        (loop
          (setq expr (read-line in nil 'done))
          (when (eq expr 'done) (throw 'done nil))
          (when (and (> (length expr) 4) (string= (subseq expr 0 4) ")abb"))
            (setq *monitor-domains* (adjoin (monitor-parse expr) *monitor-domains* :test #'string=))))))))
defun Print percent of functions tested
[*monitor-table* p1164]

— defun monitor-percent 0 —

(defun monitor-percent ()
  "Print percent of functions tested"
  (let (nonzero total)
    (declare (special *monitor-table*))
    (setq nonzero 0)
    (setq total 0)
    (maphash
     #'(lambda (k v)
         (declare (ignore k))
         (when (> (monitor-data-count v) 0) (incf nonzero))
         (incf total))
     *monitor-table*)
    (format t "~d of ~d (~d percent) tested~%
             (round (/ (* 100.0 nonzero) total))))

———

defun Find all monitored symbols containing the string
[*monitor-table* p1164]

— defun monitor-apropos 0 —

(defun monitor-apropos (str)
  "given a string, find all monitored symbols containing the string
   the search is case-insensitive. returns a list of monitor-data items"
  (let (result)
    (maphash
     #'(lambda (k v)
         (when
          (search (string-upcase str)
                   (string-upcase (symbol-name k))
                   :test #'string=)
          (push v result)))
     *monitor-table*)
    result))

———
Chapter 74

HyperDoc Basic Command support

Most of the functions create a new page with a call to the function `htMakePage`. This function takes an association list which has several possible keys.

- **domainConditions** with tests such as `(--isDomain-- S (--String--))` constraining the domains. The possible tests are
  - `isDomain`
- **text** which takes a string argument which may contain latex-like format strings.
  - a plain string
  - `beginmenu`
  - `blankline`
  - `centerline`
  - `em` with an argument to be emphasized
  - `indent` sets the column
  - `indentrel` does a relative indent by a positive or negative amount
  - `inputStrings`
  - `item` occurs between a `beginmenu` and `endmenu` text
  - `lispdwnlink` takes a string and a function to call
  - `lisplinks`
  - `menuitemstyle` takes a set of characters as an argument
  - `newline`
  - `space` with a numeric argument of the number of spaces
— **tab** with a numeric argument indicating the tab column
— **vspace** with the number of blank lines needed

- **bcStrings** which takes a list. The first element is the width of the input box, the second is the default contents, the third is the name of the variable to hold the contents, and the fourth is the domains allowed as input (see **domainConditions** above).

- **bcLinks** which takes a list containing strings and function calls. It will link to another page by calling the page generation function for that page.

- **doneButton** which takes 2 arguments, a label and a function to call.

- **radioButtons** takes a button name and set of lists, each one creating a new radio button

- **inputStrings**

- **bcHt**

The **htMakeDoneButton** will put a button on the page with the given title and a function to call when pressed.

**defun Basic Command matrix entry**

[bcReadMatrix p1182]

| defun bcMatrix |

```
(defun |bcMatrix| () (|bcReadMatrix| nil))
```

**defun Read Matrix**

[htInitPage p1262]
[htpSetProperty p1254]
[htMakePage p1263]
[htShowPage p1263]
This routine is called from several places to enter a matrix. The argument bcReadMatrix is the name of a function to call when the matrix has been entered. This value is set as an exit-Function in the page's association table.

---
defun bcReadMatrix ---

(defun bcReadMatrix (exitFunctionOrNil)
  (let (page)
    (setq page (htInitPage "Matrix Basic Command" nil))
    (htpSetProperty page 'exitFunction exitFunctionOrNil)
    (htMakePage
      '((domainConditions (isDomain PI (PositiveInteger))))
      (|text| . "Enter the size of the matrix:"))
    (inputStrings|
      ("Number of \em rows\: \space\[3\] " " 5 2 |rows| PI)
      ("Number of \em columns\: " " 5 2 |cols| PI))
    (|text| . "\blankline ")
    (|text| . "How would you like to enter the matrix?"
      (|text| . "\item")
      (|bcLinks|
        ("\menuitemstyle{By entering individual entries}" "
        |bcInputExplicitMatrix| |explicit|))
    (|text| . "\item")
    (|bcLinks|
      ("\menuitemstyle{By formula}" 
       |bcInputMatrixByFormula| |formula|)
    (|text| . "\endmenu")))
    (htShowPage)))

---

defun Input Matrix By Formula

[htInitPage p1262]
[htMakePage p1263]
[htMakeDoneButton p1284]
[objValUnwrap p128]
[htpLabelSpadValue p1256]
[parse-integer p127]
[htpLabelInputString p1254]
[htpSetProperty p1254]
[htShowPage p1263]
[$bcParseOnly p1249]
Pressing the **Continue** button will call the function `bcInputMatrixByFormulaGen` due to this line:

```lisp
(|htMakeDoneButton| "Continue" '|bcInputMatrixByFormulaGen|)
```

---

```lisp
(defun bcInputMatrixByFormula |
  (defun |bcInputMatrixByFormula| (htPage junk)
    (declare (ignore junk))
    (let (page nrows ncols)
      (declare (special |$bcParseOnly|))
      (setq page (|htInitPage| "Basic Matrix Command" (|htpPropertyList| htPage)))
      (|htMakePage|)

      (|(domainConditions| |{isDomain| S |{Symbol|})
        (|domainConditions| FE |{Expression| (|Integer|))}))
      (|text| "\menuitemstyle{}\tab{2}"
        |{text| "Enter the \{em row variable\}: ")
      (|text| "\tab{36}"
        |{bcStrings| (6 |i| |rowVar| S))
      (|text| "\blankline \newline"
        |{text| "\menuitemstyle{}\tab{2}"
        |{text| "Enter the \{em column variable\}: ")
      (|text| "\tab{36}"
        |{bcStrings| (6 |j| |colVar| S))
      (|text| "\blankline \newline"
        |{text| "\menuitemstyle{}\tab{2}"}
      (|text| "Enter the general \{em formula\} for the entries:\n"
        |{text| "\newline\tab{2} "
        |{bcStrings| (40 "1/(x - i - j - 1)" |formula| FE)))
      (|htMakeDoneButton| "Continue" '|bcInputMatrixByFormulaGen|)

      (setq nrows
        (if (null |$bcParseOnly|)
          (|objValUnwrap| (|htpLabelSpadValue| htPage '|rows|)))
        (parse-integer (|htpLabelInputString| htPage '|rows|))))
    (setq ncols
      (if (null |$bcParseOnly|)
        (|objValUnwrap| (|htpLabelSpadValue| htPage '|cols|)))
        (parse-integer (|htpLabelInputString| htPage '|cols|))))
```
defun Basic Command Matrix by Formula generate

(defun bcInputMatrixByFormulaGen (htPage)
  (let (fun formula rowVar colVar nrows ncols)
    (cond
      ((setq fun (|htpProperty| htPage '|exitFunction|))
        (funcall fun htPage))
      (t
        (setq formula (|htpLabelInputString| htPage '|formula|))
        (setq rowVar (|htpLabelInputString| htPage '|rowVar|))
        (setq colVar (|htpLabelInputString| htPage '|colVar|))
        (setq nrows (|htpProperty| htPage '|nrows|))
        (setq ncols (|htpProperty| htPage '|ncols|))
        (bcGen (strconc "matrix([[" formula
                     " for " colVar
                     " in 1.." (stringimage ncols)
                     "] for " rowVar
                     " in 1.." (stringimage nrows)
                     "]])"))))

defun Input Explicit Matrix

(defun InputExplicitMatrix
  (objValUnwrap)
  (htpLabelSpadValue)
  (parse-integer)
  (htpLabelInputString)
  (objValUnwrap)
  (htpLabelSpadValue)
  (strconc)
  (strconc)
  (parse-integer)
  (length)
  (stringimage)
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

Pressing the Continue button will call the function bcGenExplicitMatrix due to this line:

```lisp
(|htMakeDoneButton| "Continue" '|bcGenExplicitMatrix|)
```

---

**defun bcInputExplicitMatrix**

```lisp
(defun |bcInputExplicitMatrix| (htPage junk)
  (declare (ignore junk))
  (let (nrows ncols cond wrows wcols rowpart colpart prefix k name
        labelList page t1 t2)
    (declare (special |$EmptyMode| |$bcParseOnly|)))
    (setq nrows
      (if (null |$bcParseOnly|)
          (|objValUnwrap| (|htpLabelSpadValue| htPage '|rows|))
          (parse-integer (|htpLabelInputString| htPage '|rows|))))
    (setq ncols
      (if (null |$bcParseOnly|)
          (|objValUnwrap| (|htpLabelSpadValue| htPage '|cols|))
          (parse-integer (|htpLabelInputString| htPage '|cols|))))
    (setq k 0)
    (setq wrows (|stringimage nrows|))
    (setq wcols (|stringimage ncols|))
    (setq labelList
      (do ((i 1 (1+ i))) ((> i nrows) t1)
          (setq t2 nil)
            (setq t2 (cons (list (list rowpart i) (list colpart k)) t2)))
    (setq cond nil)
    (setq t2 (cons labelList t2))
    (setq rowpart (stringimage 'row))
    (setq colpart (stringimage 'col))
    (setq prefix (stringimage 'prefix))
    stronc p??]
[nreverse0 p??]
[htpPropertyList p1253]
[htShowPage p1263]
[$EmptyMode p??]
[$bcParseOnly p1249]
(setq t1
  (append t1
    (do ((j 1 (1+ j))) ((> j ncols) (nreverse0 t2))
      (setq t2
        (cons
          (progn
            (setq rowpart (strconc "{\em Row" (htStringPad| i wrows))
            (setq colpart (strconc ", Column" (htStringPad| j wcols) ":}\space{2}"))
            (setq prefix (strconc rowpart colpart))
            (setq name (intern (stringimage (setq k (1+ k))))
                (list prefix "" 30 0 name 'P)
                t2)))))))
  (setq labelList
    (list
      (list '|domainConditions| |
        (isDomain| P (|Polynomial| $\text{EmptyMode}$))
        cond
      )
    (cons '|inputStrings| labelList)))
  (setq page (|htInitPage| "Solve Basic Command" (|htpPropertyList| htPage)))
  (bcHt| "Enter the entries of the matrix:")
  (|htMakePage| labelList)
  (|htMakeDoneButton| "Continue" 'bcGenExplicitMatrix)
  (|htpSetProperty| page '|nrows| nrows)
  (|htpSetProperty| page '|ncols| ncols)
  (|htShowPage|)))

---

defun Basic Command generate explicit matrix

[htpSetProperty p1254]
[htpInputAreaAlist p1253]
[htpProperty p1254]
[bcGen p1245]
[bcMatrixGen p1188]

— defun bcGenExplicitMatrix —

(defun bcGenExplicitMatrix |htPage|
  (let (fun)
    (|htpSetProperty| htPage 'matrix (|htpInputAreaAlist| htPage))
    (if (setq fun (|htpProperty| htPage 'exitFunction))
      (funcall fun htPage)
      (|bcGen| (|bcMatrixGen| htPage)))))

---
defun Basic Command generate matrix

(defun bcMatrixGen (htPage)
  (let (nrows ncols formula rowVar colVar mat k matform matstring)
    (setq nrows (htpProperty htPage '|nrows|))
    (setq ncols (htpProperty htPage '|ncols|))
    (setq mat (htpProperty htPage '|matrix|))
    (cond
      ((setq formula (lassoc '|formula| mat))
       (setq formula (elt formula 0))
       (setq rowVar (elt (lassoc '|rowVar| mat) 0))
       (setq colVar (elt (lassoc '|colVar| mat) 0))
       (strconc "matrix([[
         " formula
         " for " colVar
         " in 1.." (stringimage ncols)
         "] for " rowVar
         " in 1.." (stringimage nrows)
         ")])")
      ((setq mat (htpProperty htPage '|matrix|))
       (setq mat (reverse mat))
       (setq k (- 1))
       (setq matform
         (loop for i from 0 to (1- nrows)
           collect (loop for j from 0 to (1- ncols)
                       collect (elt (elt mat (incf k)) 1))))
       (setq matstring
         (bcwords2liststring|
         (loop for t1 in matform collect (bcwords2liststring| t1)))))
      (strconc "matrix(" matstring ")")
      (t (systemError nil))))

; Hypertex commands other than solve and matrix
1189

defun Basic Command iteration
[bcMatrixGen p1188]
— defun bcDrawIt2 —
(defun |bcDrawIt2| (ind a b)
(strconc "{}" ind "=" a "{}.." b "{}"))

———-

defun Indefinite Integration Basic Command
[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]

Pressing
the Continue button will call the function bcIndefiniteIntegrateGen due to this line:
(|doneButton| "Continue" |bcIndefiniteIntegrateGen|)))

— defun bcIndefiniteIntegrate —
(defun |bcIndefiniteIntegrate| ()
(declare (special |$EmptyMode|))
(|htInitPage| ’|Indefinite Integration Basic Command| nil)
(|htMakePage|
’((|domainConditions| (|isDomain| EM |$EmptyMode|)
(|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
(|text| . "\\newline ")
(|text| . "\\menuitemstyle{}\\tab{2}")
(|text| . "Enter the {\\em function} you would like to integrate:")
(|text| . "\\newline\\tab{2} ")
(|bcStrings| (45 "1/(x**2 + 6)" |integrand| EM))
(|text| . "\\blankline") (|text| . "\\newline ")
(|text| . "\\menuitemstyle{}\\tab{2}")


defun bcIndefiniteIntegrateGen

(htpLabelInputString p1254)
(strconc p??)
[bcGen p1245]

— defun bcIndefiniteIntegrateGen —

(defun bcIndefiniteIntegrateGen (htPage)
(let (integrand var)
 (setq integrand (htpLabelInputString htPage '|integrand|))
 (setq var (htpLabelInputString htPage '|symbol|))
 (bcGen (strconc "integrate(" integrand "," var ")"))))

defun Definite Integration Basic Command

(htInitPage p1262)
(htMakePage p1263)
(htShowPage p1263)
[$EmptyMode p??]
Pressing the **Continue** button will call the function `bcDefiniteIntegrateGen` due to this line:

```
(defun bcDefiniteIntegrate ()
  (declare (special $EmptyMode))
  (htInitPage|'|Definite Integration Basic Command| NIL)
  (htMakePage|
    '((domainConditions| (isDomain EM $EmptyMode)
        (isDomain S (String)) (isDomain SY (Symbol)))
      (text . "\newline")
      (text . "\menuitemstyle{}\tab{2}"
        Enter the {\em function} you would like to integrate:"
      )
      (bcStrings| (45 "1/(x**2 + 6)" |integrand| EM))
      (text . "\blankline")
      (text . "\newline")
      (text . "\menuitemstyle{}\tab{2}"
        Enter the {\em variable of integration}:"
      )
      (text . "\tab{37}
        |x| |symbol| SY)
      (text . "\blankline")
      (text . "\newline")
      (text . "\menuitemstyle{}\tab{2}"
        Enter {\em lower limit}:"
      )
      (radioButtons| |fromButton|
        "" "Minus infinity" |minusInfinity|)
      ""
      ((|text| . "A finite point:\tab{15}"
        (bcStrings| (10 0 |from| EM . |bcOptional|))))
```

---

1/(x**2 + 6)
defun bcDefiniteIntegrateGen

(defun bcDefiniteIntegrateGen (htPage)
  (let (integrand var lowerLimit upperLimit varpart)
    (setq integrand (htpLabelInputString htPage '|integrand|))
    (setq var (htpLabelInputString htPage '|symbol|))
    (setq lowerLimit
      (if (eq (htpButtonValue htPage '|fromButton|) '|fromPoint|)
        (htpLabelInputString htPage '|from|) "%minusInfinity")
      (htpLabelInputString htPage '|from|))
    (setq upperLimit
      (if (eq (htpButtonValue htPage '|toButton|) '|toPoint|)
        (htpLabelInputString htPage '|to|) "%plusInfinity")
      (htpLabelInputString htPage '|to|))
    (setq varpart (strconc var " = " lowerLimit .. upperLimit))
    (bcGen (strconc "integrate(" integrand "," varpart ")"))))

—— defun bcDefiniteIntegrateGen ——

defun Sum Basic Command

(defun Sum Basic Command (htInitPage htMakePage)
  (let (htPage)
    (htInitPage htPage)
    (htMakePage htPage)
    (htShowPage htPage))
Pressing the Continue button will call the function \texttt{bcSumGen} due to this line:

\begin{verbatim}
(|doneButton| "Continue" |bcSumGen|)
\end{verbatim}

--- defun bcSum ---

\begin{verbatim}
(defun |bcSum| ()
  (declare (special |$EmptyMode|))
  (|htInitPage| '|Sum Basic Command| NIL)
  (|htMakePage|
   '(((|domainConditions| (|isDomain| EM |$EmptyMode|))
      (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
    (|text| . "\newline ")
    (|text| . "\menuitemstyle{}\tab{2}"
    (|text| . "Enter the \{\em function\} you would like to sum:"))
    (|text| . "\newline\tab{2}"
    (|bcStrings| (44 "i**3" |summand| EM))
    (|text| . "\newline\tab{2}"
    (|text| . "\menuitemstyle{}\tab{2}"
    (|text| . "Enter the \{\em summation index\}:")
    (|text| . "\newline\tab{2}"
    (|bcStrings| (10 |i| |index| SY))
    (|text| . "\newline\tab{2}"
    (|text| . "\menuitemstyle{}\tab{2}"
    (|text| . "Enter the limits of the sum:"))
    (|text| . "\newline\tab{10}{\em From:}"
    (|bcStrings| (10 1 |first| S))
    (|text| . "\newline\tab{10}{\em To:}"))
  (|doneButton| "Continue" |bcSumGen|)))
  (|htShowPage|)))
\end{verbatim}
defun bcSumGen

(defun bcSumGen (htPage)
  (let (mand index car last)
    (setq mand (htpLabelInputString htPage '|summand|))
    (setq index (htpLabelInputString htPage '|index|))
    (setq car (htpLabelInputString htPage '|first|))
    (setq last (htpLabelInputString htPage '|last|))
    (bcGen (strconc "sum(" mand "," index " = " car "." last ")"))))

—— defun bcSumGen ——

defun Sum Basic Command

(defun bcProduct ()
  (declare (special $EmptyMode$))
  (htInitPage '|Product Basic Command| NIL)
  (htMakePage
   '((domainConditions (isDomain EM $EmptyMode$) (isDomain S (String)) (isDomain SY (Symbol)))
     (text . "Enter the \{em function\} you would like to compute the product of:"
     (inputStrings| "" "" 45 "i**2" |mand| EM))
     (text . "\vspace{1}\newline Enter the \{em index of the product\}:
     (inputStrings| "" "" 5 |i| |index| SY)
     (text . "\vspace{1}\newline Enter the limits of the index:"
     (inputStrings| "\newline\{em From:\}" "" 10 "1" |first| EM)
     ("\{em To:\}\space{2}" "" 10 "n" |last| EM))
     (doneButton |Continue| bcProductGen)))
  (htShowPage)))

——
(defun bcProductGen

(htpLabelInputString p1254)
[strconc p??]
[bcGen p1245]

defun bcProductGen

(defun |bcProductGen| (htPage)
(let (mand index car last)
(setq mand (|htpLabelInputString| htPage '|mand|))
(setq index (|htpLabelInputString| htPage '|index|))
(setq car (|htpLabelInputString| htPage '|first|))
(setq last (|htpLabelInputString| htPage '|last|))
(|bcGen| (strconc "product(" mand "," index "," car "," last ")"))))

---

defun Differentiate Basic Command

[htInitPage p1262]
[htMakePage p1263]
[htMakeDoneButton p1284]
[htShowPage p1263]
[$EmptyMode p??]

---

Pressing the Continue button will call the function bcDifferentiateGen due to this line:

(|htMakeDoneButton| "Continue" '|bcDifferentiateGen|)

---

defun bcDifferentiate ---
(defun |bcDifferentiate| ()
  (declare (special |$EmptyMode|))
  (|htInitPage| ' differentiated| Basic Command| nil)
  (|htMakePage| ' (|domainConditions| (|isDomain| EM |$EmptyMode|)
          (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))))
  (|htMakeDoneButton| "Continue" '|bcDifferentiateGen|)
  (|htShowPage|))

---

defun bcDifferentiateGen

[htpLabelInputString p1254]
[bcString2WordList p1246]
[bcwords2liststring p1246]
[length p??]
[bcError p1247]
[strconc p??]
[bcGen p1245]

— defun bcDifferentiateGen —

(defun |bcDifferentiateGen| (htPage)
  (let (mand varlist indexList varpart indexpart lastPart)
    (setq mand (|htpLabelInputString| htPage '|diffand|))
    (setq varlist
      (|bcString2WordList| (|htpLabelInputString| htPage '|variables|)))
    (setq indexList
      (|bcString2WordList| (|htpLabelInputString| htPage '|times|))))
(setq varpart
  (if (> (length varlist) 1)
    (|bcwords2liststring| varlist)
    (car varlist)))
(setq indexpart
  (cond
    ((null indexList) nil)
    ((null (cdr indexList)) (car indexList))
    ((= (length indexList) (length varlist)) (|bcwords2liststring| indexList))
    (t (|bcError| "You must say how many times you want to differentiate with respect to each variable---or leave that entry blank.\n")
      (setq lastPart (if indexpart (strconcat ", " indexpart ") "")
          (|bcGen| (strconcat "differentiate(" mand ", " varpart lastPart")))\n))

defun Draw Basic Command

[htInitPage p1262]
[bcHt p1260]
[htShowPage p1263]

| defun bcDraw |

--- defun bcDraw ---

(defun |bcDraw| ()
  (|htInitPage| "Draw Basic Command" NIL)
  (|bcHt| "What would you like to draw?\n")
  (|bcHt| "\ncenterline{\em Two Dimensional Plots}\n")
  (|bcHt| "\lispdownlink{A function of one variable}{(|bcDraw2Dfun|)}")
  (|bcHt| "\space{2}y = f(x)\n")
  (|bcHt| "\lispdownlink{A parametrically defined curve}{(|bcDraw2Dpar|)}")
  (|bcHt| "\space{2}(x(t), y(t))\n")
  (|bcHt| "\lispdownlink{A solution to a polynomial equation}{(|bcDraw2DSolve|)}")
  (|bcHt| "\space{2} p(x,y) = 0\n")
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

---

```lisp
defun Draw Basic Command by Function
```

Pressing the **Continue** button will call the function `bcDraw2DfunGen` due to this line:

```lisp
  (doneButton "Continue" |bcDraw2DfunGen|)
```

```lisp
— defun bcDraw2Dfun —
```

```lisp
  (defun |bcDraw2Dfun| ()
```
(declare (special $EmptyMode$))
(\htInitPage\ "Draw Basic Command" NIL)
(\htMakePage|
'((\domainConditions\ (\isDomain\ EM $EmptyMode$)
  (\isDomain\ F (Float)) (\isDomain\ SY (Symbol)))
  (\text\ "\centerline{Drawing (\em y = f(x)})\newline "
   "\centerline{where (\em y) is the dependent variable and (\em x) is the independent variable}\vspace{1}\newline "
   (\bcStrings\ (55 "x*cos(x)" |function| EM))
  (\text\ .)
  "\vspace{1}\newline\menuitemstyle{}\tab{2}What (\em function) f would you like to draw?\newline\tab{2}"
  (\bcStrings\ (6 |y| |dependent| SY))
  (\text\ .)"
  "\vspace{1}\newline\menuitemstyle{}\tab{2}Enter (\em dependent) variable:"
  (\bcStrings\ (6 |y| |dependent| SY))
  (\text\ .)"
  "\vspace{1}\newline\menuitemstyle{}\tab{2}Enter (\em independent) variable and (\em range):\newline\tab{2}"
  (\text\ .)"
  "\vspace{1}\newline\menuitemstyle{}\tab{2}Optionally enter a (\em title) for your curve:"
  (\bcStrings\ (15 "y = x*cos(x)" |title| S))
  (\text\ .)"
  (\doneButton\ "Continue" |bcDraw2DfunGen|) (\text\ . "{"})
)
(\htShowPage|)

defun bcDraw2DfunGen

[htpLabelInputString p1254]
[strconc p??]
[bcFinish p1243]
[bcDrawIt2 p1189]

— defun bcDraw2DfunGen —
defun Draw Basic Command by Parameters

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]

Pressing the Continue button will call the function bcDraw2DparGen due to this line:

(\doneButton\ "Continue" bcDraw2DparGen))

— defun bcDraw2Dpar —

(defun \bcDraw2Dpar\ ()
 (declare (special \$EmptyMode|)))
 (\htInitPage\ "Draw Basic Command" NIL)
 (\htMakePage\
   '((\domainConditions| (\isDomain| EM \$EmptyMode)\))
Drawing a parametrically defined curve: \( \text{(f1(t), f2(t))} \) in terms of two functions \( \text{f1} \) and \( \text{f2} \) and an independent variable \( \text{t} \). Enter the two \( \text{functions:} \)

- \( \text{Function 1:} \) \(-9\sin(4t/5)\)
- \( \text{Function 2:} \) \(8\sin(t)\)

Enter \( \text{independent} \) variable and range:
- \( \text{Variable:} \) \( t \) ranges from \(-5\pi\) to \(5\pi\)

Optionally enter a \( \text{title} \) for your curve:
- \( \text{Title:} \) \( \text{Lissajous} \)

---

defun bcDraw2DparGen

[htpLabelInputString p1254]
[strconc p??]
[bcFinish p1243]
[bcDrawIt2 p1189]

---

defun bcDraw2DparGen

(let (fun1 fun2 ind from1 to1 title curvePart titlePart)  
  (setq fun1 (htpLabelInputString htPage '|function1|)) 
  (setq fun2 (htpLabelInputString htPage '|function2|)) 
  (setq ind (htpLabelInputString htPage '|ind|)) 
  (setq from1 (htpLabelInputString htPage '|from1|)) 
  (setq to1 (htpLabelInputString htPage '|to1|)) 
  (setq title (htpLabelInputString htPage '|title|)) 
  (setq curvePart (strconc "curve( "{}" fun1 ",{}" fun2 ")")) 
  (cond 
    ((not (string-equal title "")) 
     (setq titlePart (strconc "{}" "title ==" title "")) 
     (bcFinish "draw" curvePart (bcDrawIt2 ind from1 to1 titlePart)
   )
defun Draw Basic Command by Equation Solution

(htInitPage p1262)
(htMakePage p1263)
(htMakeDoneButton p1284)
(htShowPage p1263)
($EmptyMode p??)

Pressing

the Continue button will call the function bcDraw2DSolveGen due to this line:

(|htMakeDoneButton| "Continue" '|bcDraw2DSolveGen|)

— defun bcDraw2DSolve —

(defun |bcDraw2DSolve| ()
(declare (special |$EmptyMode|))
(|htInitPage| "Draw Basic Command" nil)
(|htMakePage|
  '(((|domainConditions| (|isDomain| EM |$EmptyMode|))
      (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
   (|text| "Plotting the solution to \em \(p(x,y) = 0\), where \)
            "\em \(p\) is a polynomial in two variables \(x\) and \(y\)\)"
            "\vspace{1}\newline\menuitemstyle{}\tab{2}Enter the \em polynomial \: p:"

(defun |bcDraw2DSolve| ()
(declare (special |$EmptyMode|))
(|htInitPage| "Draw Basic Command" nil)
(|htMakePage|
  '(((|domainConditions| (|isDomain| EM |$EmptyMode|))
      (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
   (|text| "Plotting the solution to \em \(p(x,y) = 0\), where \)
            "\em \(p\) is a polynomial in two variables \(x\) and \(y\)\)"
            "\vspace{1}\newline\menuitemstyle{}\tab{2}Enter the \em polynomial \: p:"

(t
  (|bcFinish| "draw" curvePart (|bcDrawIt2| ind from1 to1)))))

———
Enter the \emph{variables}:

\begin{itemize}
\item \texttt{Variable 1:} \texttt{x} \texttt{ranges from:} -15 \texttt{to:} 10
\item \texttt{Variable 2:} \texttt{y} \texttt{ranges from:} -10 \texttt{to:} 50
\end{itemize}

Optionally enter a \emph{title} for your curve:

```
(defun bcDraw2DSolveGen (htPage)
  (let (func ind1 from1 to1 ind2 from2 to2 title clipPart titlePart)
    (setq func (htpLabelInputString htPage \texttt{|function|}))
    (setq ind1 (htpLabelInputString htPage \texttt{|independent1|}))
    (setq from1 (htpLabelInputString htPage \texttt{|from1|}))
    (setq to1 (htpLabelInputString htPage \texttt{|to1|}))
    (setq ind2 (htpLabelInputString htPage \texttt{|independent2|}))
    (setq from2 (htpLabelInputString htPage \texttt{|from2|}))
    (setq to2 (htpLabelInputString htPage \texttt{|to2|}))
    (setq title (strconc \texttt{"{}"} \texttt{|title|}))
    (setq clipPart (strconc \texttt{"{}"} "range==\{\}"
      from1 ".." to1 \texttt{[{},]}"
      from2 ".." to2 \texttt{[{}])
    ))
    (cond
      ((not (string-equal title \texttt{""}))
      (setq titlePart (strconc \texttt{"{}"} "title==\"" title \"\")
      )
    )
    (bcFinish "draw" (strconc func \texttt{" = 0"}) ind1 ind2 clipPart titlePart))
```
defun Draw Basic Command by 3D function

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]

Pressing the Continue button will call the function bcDraw3DfunGen due to this line:

(|doneButton| "Continue" |bcDraw3DfunGen|)

— defun bcDraw3Dfun —

(defun |bcDraw3Dfun| ()
  (declare (special |$EmptyMode|))
  (|htInitPage| "Three Dimensional Draw Basic Command" nil)
  (|htMakePage| '(((|domainConditions| (|isDomain| EM |$EmptyMode|))
    (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|))))
  (|text| "\centerline{Drawing {\em z = f(x,y)}}\newline "
    "\centerline{where {{\em z} is the dependent variable and}}\newline "
  )

(t
  (|bcFinish| "draw" (strconc fun " = 0 ") ind1 ind2 clipPart))))

———

defun bcDraw3Dfun

(defun bcDraw3Dfun ()
  (declare (special |$EmptyMode|))
  (|htInitPage| "Three Dimensional Draw Basic Command" nil)
  (|htMakePage| '(((|domainConditions| (|isDomain| EM |$EmptyMode|))
    (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|))))
  (|text| "\centerline{Drawing {\em z = f(x,y)}}\newline "
    "\centerline{where {{\em z} is the dependent variable and}}\newline "
  )

———
where \(x, y\) are the independent variables.

What function \(f\) which you like to draw?

|bcStrings| (55 "exp(cos(x-y)-sin(x*y))-2" |function| EM)

Enter dependent variable:

|bcStrings| (6 \(z\) |dependent| SY)

Enter independent variables and ranges:

\(\text{Variable:}\)

|x| \(\text{independent1}\ SY)

|text| \(\text{ranges \text{from:}}\)

|bcStrings| (9 -5 \(\text{from1}\ F))

|text| \(\text{to:}\)

|bcStrings| (9 5 \(\text{to1}\ F))

|text| \(\text{Variable:}\)

|bcStrings| (6 \(y\) \(\text{independent2}\ SY)

|text| \(\text{ranges \text{from:}}\)

|bcStrings| (9 -5 \(\text{from2}\ F))

|text| \(\text{to:}\)

|bcStrings| (9 5 \(\text{to2}\ F))

Optionally enter a title for your surface:

|bcStrings| (15 "" |title| SY)

|doneButton| "Continue" |bcDraw3DfunGen|)

----

defun bcDraw3DfunGen

[htpLabelInputString p1254]
[strconc p??]
[bcFinish p1243]
[bcDrawIt2 p1189]

— defun bcDraw3DfunGen —

(defun |bcDraw3DfunGen| (htPage)
  (let (fun dep ind1 from1 to1 ind2 from2 to2 title titlePart)
    (setq fun (|htpLabelInputString| htPage '|function|))
    (setq dep (|htpLabelInputString| htPage '|dependent|))
    (setq ind1 (|htpLabelInputString| htPage '|independent1|))
    (setq from1 (|htpLabelInputString| htPage '|from1|))
    (setq to1 (|htpLabelInputString| htPage '|to1|))
    (setq ind2 (|htpLabelInputString| htPage '|independent2|))
    (setq from2 (|htpLabelInputString| htPage '|from2|))
    (setq to2 (|htpLabelInputString| htPage '|to2|))
    (setq title (|htpLabelInputString| htPage '|title|)))

(cond
defun Draw Basic Command by 3D parameterized tube

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p???]

Pressing the Continue button will call the function bcDraw3DparGen due to this line:

\[ (\text{|doneButton| "Continue" |bcDraw3DparGen|}) \]

— defun bcDraw3Dpar —

(defun |bcDraw3Dpar| ()

|$EmptyMode p???$
\textcenterline{Drawing a parametrically defined curve:}
\textcenterline{\textemph{( f1(t), f2(t), f3(t) )}}
\textcenterline{in terms of three functions \textemph{f1}, \textemph{f2}, and \textemph{f3}.}

\textcenterline{and an independent variable \textemph{t}.}

\textindent{0}\textcenterline{Enter \textemph{the three \{em functions\} of the independent variable:}}

\textindent{0}\textcenterline{\textemph{{\em Function f1:}}}
\textindent{0}\textcenterline{(\textit{bcStrings} (42 "1.3*cos(2*t)*cos(4*t) + sin(4*t)*cos(t)" \textit{function1} EM))}

\textindent{0}\textcenterline{\textemph{{\em Function f2:}}}
\textindent{0}\textcenterline{(\textit{bcStrings} (42 "1.3*sin(2*t)*cos(4*t) - sin(4*t)*sin(t)" \textit{function2} EM))}

\textindent{0}\textcenterline{\textemph{{\em Function f3:}}}
\textindent{0}\textcenterline{(\textit{bcStrings} (42 "2.5*cos(4*t)" \textit{function3} EM))}

\textindent{0}\textcenterline{Enter \textemph{the independent variable and range:}}

\textindent{0}\textcenterline{\textemph{{\em Variable:}}}
\textindent{0}\textcenterline{(\textit{bcStrings} (6 \textit{t} \textit{ind} SY))}

\textindent{0}\textcenterline{ranges \textemph{from:}}
\textindent{0}\textcenterline{(\textit{bcStrings} (9 0 \textit{from1} F))}

\textindent{0}\textcenterline{to:}
\textindent{0}\textcenterline{(\textit{bcStrings} (9 "4*\%pi" \textit{to1} F))}

\textindent{0}\textcenterline{Optionally enter a \textemph{title} for your surface:}
\textindent{0}\textcenterline{(\textit{bcStrings} (15 "knot" \textit{title} S))}

\textindent{0}\textcenterline{(\textit{doneButton} "Continue" \textit{bcDraw3DparGen})))}

\textindent{0}\textcenterline{(\textit{htShowPage}()))}

---

defun bcDraw3DparGen

[\textit{htpLabelInputString} p1254]
[\textit{strconc} p778]
[bcFinish p1243]
[bcDrawIt2 p1189]

---

defun bcDraw3DparGen ---
defun Draw Basic Command by 3D parameterized function

[htInitPage p1262]
[htMakePage p1263]
[htMakeDoneButton p1284]
[htShowPage p1263]
[$EmptyMode p??]

Pressing the **Continue** button will call the function **bcDraw3Dpar1Gen** due to this line:

\[(|htMakeDoneButton| "Continue" '|bcDraw3Dpar1Gen|)\]

— defun **bcDraw3Dpar1** —
(defun |bcDraw3Dpar1| ()
 (declare (special |$EmptyMode|))
 (|htInitPage| "Draw Basic Command" NIL)
 (|htMakePage|
 '(((|domainConditions| (|isDomain| EM |$EmptyMode|)
 (|isDomain| F |Float|)) (|isDomain| SY |Symbol|))
 (|text| "\centerline{Drawing a parametrically defined surface:}\newline 
 "\centerline{{\em ( f1(u,v), f2(u,v), f3(u,v) )}}\newline 
 "\centerline{in terms of three functions \{\em f1}, \{\em f2}, and \{\em f3}\} newline 
 "\centerline{and two independent variables \{\em u} and \{\em v}\} \vspace{1}\newline\menuitemstyle{}\tab{2}
 "Enter the three \{\em functions\} of the independent variables:"

 (|text| . "\newline\tab{2})
 (|text| . "{\em Function f1:}"
 (|bcStrings| (43 "u*sin(v)" |function1| EM))
 (|text| . "\newline\tab{2})
 (|text| . "{\em Function f2:}"
 (|bcStrings| (43 "v*cos(u)" |function2| EM))
 (|text| . "\newline\tab{2})
 (|text| . "{\em Function f3:}"
 (|bcStrings| (43 "u*cos(v)" |function3| EM))
 (|text| . "\newline\menuitemstyle{}\tab{2}
 "Enter independent \{\em variables\} and ranges:"

 (|text| . "\newline\tab{2})
 (|text| . "{\em Variable 1:}"
 (|bcStrings| (5 |u| |ind1| SY))
 (|text| . "ranges \{\em from:\}"
 (|bcStrings| (9 "-%pi" |from1| F)) (|text| . "{\em to:}"
 (|bcStrings| (9 "%pi" |to1| F))
 (|text| . "\newline\tab{2})
 (|text| . "{\em Variable 2:}"
 (|bcStrings| (5 |v| |ind2| SY))
 (|text| . "ranges \{\em from:\}"
 (|bcStrings| (9 "-%pi/2" |from2| F))
 (|text| . "{\em to:}"
 (|bcStrings| (9 "%pi/2" |to2| F))
 (|text| \indent{0}\newline\menuitemstyle{}\tab{2} "
 "Optionally enter a \{\em title\} for your surface:"

 (|bcStrings| (15 "surface" |title| S))
 (|text| . "\indent{0})))
 (|htMakeDoneButton| "Continue" '|bcDraw3Dpar1Gen|)
 (|htShowPage|))

defun bcDraw3Dpar1Gen

[htpLabelInputString p1254]
[bcDrawIt2 p1189]
[strconc p??]
defun bcDraw3Dpar1Gen (htPage)
(let (fun1 fun2 fun3 ind1 from1 to1 ind2 from2 to2
     title r1 r2 surfacePart titlePart)
  (setq fun1 (htpLabelInputString htPage '|function1|))
  (setq fun2 (htpLabelInputString htPage '|function2|))
  (setq fun3 (htpLabelInputString htPage '|function3|))
  (setq ind1 (htpLabelInputString htPage '|ind1|))
  (setq from1 (htpLabelInputString htPage '|from1|))
  (setq to1 (htpLabelInputString htPage '|to1|))
  (setq ind2 (htpLabelInputString htPage '|ind2|))
  (setq from2 (htpLabelInputString htPage '|from2|))
  (setq to2 (htpLabelInputString htPage '|to2|))
  (setq title (htpLabelInputString htPage '|title|))
  (setq r1 (bcDrawIt2 ind1 from1 to1))
  (setq r2 (bcDrawIt2 ind2 from2 to2))
  (setq surfacePart (strconc "surface(" "{}" fun1 ",{}," fun2 ",{}," fun3 ")")
  (cond
    ((not (string= title ""))
     (setq titlePart (strconc "{}" "title ==\"" title \"\")
     (bcFinish "draw" surfacePart r1 r2 titlePart))
    (t (bcFinish "draw" surfacePart r1 r2))))

defun Series Basic Command

(htInitPage p1262)
(htMakePage p1263)
(htShowPage p1263)
[$EmptyMode p???]

---

---

---
defun Series Basic Command expand around a point

[htInitPage p1262]
[htMakePage p1263]
[htMakeDoneButton p1284]
[htShowPage p1263]
[$EmptyMode p??]

Pressing the Continue button will call the function bcSeriesExpansionGen due to this line:

((htMakeDoneButton "Continue" '\bcSeriesExpansionGen))

— defun bcSeriesExpansion —

(defun bcSeriesExpansion (a b)
  (declare (ignore a b))
  (declare (special |$EmptyMode|))
  (htInitPage "Series Expansion Basic Command" nil)
  (htMakePage
   '((\domainConditions (\isDomain EM |$EmptyMode|)
      (\isDomain EEM (\Expression |$EmptyMode|))
      (\isDomain S (\String)) (\isDomain SY (\Symbol))))
  )
  (text . "\newline ")
defun bcSeriesExpansionGen

(defun bcSeriesExpansionGen (htPage)
  (let (fun var point terms)
    (setq fun (htpLabelInputString| function| htPage))
    (setq var (htpLabelInputString| variable| htPage))
    (setq point (htpLabelInputString| point| htPage))
    (setq terms (strconc var = point)))

---

defun Series Basic Command series by formula

(defun Series Basic Command series by formula (htPage)
  (htMakeDoneButton |Continue| 'bcSeriesExpansionGen)
  (htShowPage))
(defun bcSeriesByFormula (a b)  
  (declare (ignore a b))  
  (htInitPage "Power Series Basic Command" NIL)  
  (htMakePage  
    '((|text| . "Select the kind of power series you want to create:")  
      (|text| . "\beginmenu") (|text| . "\item")  
      (bcLinks  
        ("\\menuitemstyle{Taylor Series}" " |bcTaylorSeries| |taylor|))  
      (|text| .  
        "\newline Series where the exponent ranges over the integers from a \textit{non-negative integer} value to plus infinity by an arbitrary \textit{positive integer} step size")  
      (|text| . "\item")  
      (bcLinks  
        ("\\menuitemstyle{Laurent Series}" " |bcLaurentSeries| |laurent|))  
      (|text| .  
        "\newline Series where the exponent ranges from an arbitrary \textit{integer} value to plus infinity by an arbitrary \textit{positive integer} step size")  
      (|text| . "\item")  
      (bcLinks  
        ("\\menuitemstyle{Puiseux Series}" " |bcPuiseuxSeries| |puiseux|))  
      (|text| .  
        "\newline Series where the exponent ranges from an arbitrary \textit{rational value} to plus infinity by an arbitrary \textit{positive rational number} step size")  
      (|text| . "\endmenu")  
    )  
  (htShowPage))  

---

defun Taylor Series Basic Command

[htInitPage p1262]  
[htMakePage p1263]  
[htShowPage p1263]  
[$EmptyMode p??]
Pressing the Continue button will call the function \texttt{bcTaylorSeriesGen} due to this line:

\begin{verbatim}
|doneButton| "Continue" |bcTaylorSeriesGen|)
\end{verbatim}

--- defun bcTaylorSeries ---

\begin{verbatim}
(defun \texttt{bcTaylorSeries} (a b)
  (declare (ignore a b))
  (declare (special \texttt{$\EmptyMode$}))
  (|htInitPage| "Taylor Series Basic Command" NIL)
  (|htMakePage|
   '(((|domainConditions| ((|isDomain| EM \texttt{$\EmptyMode$})
      ((|isDomain| EEM (|Expression| \texttt{$\EmptyMode$}))
      ((|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
      ((|bcStrings| (55 "1/factorial(i)" |formula| EM))
      (\texttt{\textbackslash{}blankline \textbackslash{}tab{}})
      (\texttt{\textbackslash{}menuitemstyle{}\textbackslash{}tab{2}}")
      (\texttt{\textbackslash{}text{} "Enter the formula for the general coefficient of the series"}
      (\texttt{\textbackslash{}text{} \textbackslash{}text{} "Enter the \textit{index variable} for your formula"}
      (\texttt{\textbackslash{}text{} \textbackslash{}tab{49}}") (\texttt{|bcStrings| (8 |i| |index| SY))
      (\texttt{\textbackslash{}text{} \textbackslash{}newline "}
      (\texttt{\textbackslash{}text{} \textbackslash{}menuitemstyle{}\textbackslash{}tab{2}}")
      (\texttt{\textbackslash{}text{} "Enter the \textit{power series variable}"}
      (\texttt{\textbackslash{}text{} \textbackslash{}tab{49}}") (\texttt{|bcStrings| (8 |x| |variable| SY))
      (\texttt{\textbackslash{}text{} \textbackslash{}newline "}
      (\texttt{\textbackslash{}text{} \textbackslash{}menuitemstyle{}\textbackslash{}tab{2}}")
      (\texttt{\textbackslash{}text{} "Enter the \textit{point} about which you want to expand"})
\end{verbatim}
"For Taylor Series, the exponent of the power series variable ranges from an \textit{initial value}, an arbitrary non-negative integer, to plus infinity; the \textit{step size} is any positive integer."

Enter the \textit{initial value} of the index (an integer)

Enter the \textit{step size} (a positive integer)

---

defun bcSeriesByFormulaGen
[bcNotReady p1247]

--- defun bcSeriesByFormulaGen ---

(defun |bcSeriesByFormulaGen| (htPage)
  (declare (ignore htPage))
  (|bcNotReady|))

---

defun Laurent Series Basic Command
[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]
Pressing the Continue button will call the function bcLaurentSeriesGen due to this line:

```
(defun bcLaurentSeriesGen
  (a b)
  (declare (special $EmptyMode) (ignore a b))
  (htInitPage "Laurent Series Basic Command" NIL)
  (htMakePage
    '((domainConditions
        (isDomain EM $EmptyMode)
          (isDomain EEM (Expression $EmptyMode))
          (isDomain S (String)) (isDomain I (Integer))
          (isDomain PI (PositiveInteger))
          (isDomain SY (Symbol)))
      (text . "Enter the formula for the general coefficient of the series")
      (bcStrings (55 "(-1)**(n - 1)/(n + 2)" |formula| EM))
      (text . "Enter the initial value")
      (text . "Enter the step size (a positive integer value, to plus infinity)
```
Enter the point about which you want to expand

Enter the initial value of the index (an integer)

Enter the step size (a positive integer)

Pressing the Continue button will call the function \texttt{bcPuiseuxSeriesGen} due to this line:
---

\[
\text{defun bcPuiseuxSeries} (a b)
\]

\[
\text{(declare (special |$EmptyMode$|) (ignore a b))}
\]

\[
\text{(|htInitPage| "Puiseux Series Basic Command" nil)}
\]

\[
\text{(|htMakePage|}
\]

\[
\text{\((|domainConditions| (|isDomain| EM |$EmptyMode$|))}
\]

\[
\text{(|isDomain| EEM (|Expression| |$EmptyMode$|))}
\]

\[
\text{(|isDomain| S (|String|)) (|isDomain| I (|Integer|))}
\]

\[
\text{(|isDomain| PI (|PositiveInteger|))}
\]

\[
\text{(|isDomain| RN (|Fraction| (|Integer|)))}
\]

\[
\text{(|isDomain| SY (|Symbol|))})
\]

\[
\text{|htShowPage|})
\]

---
defun bcTaylorSeriesGen
[bcSeriesGen p1219]

— defun bcTaylorSeriesGen —

(defun \(\text{bcTaylorSeriesGen}\) (htPage)
  (bcSeriesGen htPage))

-----

defun bcLaurentSeriesGen
[bcSeriesGen p1219]

— defun bcLaurentSeriesGen —

(defun \(\text{bcLaurentSeriesGen}\) (htPage)
  (bcSeriesGen htPage))

-----

defun bcPuiseuxSeriesGen
[bcSeriesGen p1219]

— defun bcPuiseuxSeriesGen —

(defun \(\text{bcPuiseuxSeriesGen}\) (htPage)
  (bcSeriesGen htPage))

-----

defun bcSeriesGen
[htpLabelInputString p1254]
[strconc p??]
[bcFinish p1243]

— defun bcSeriesGen —

(defun \(\text{bcSeriesGen}\) (htPage)
(let (step min formula index var point varPart minPart)
  (setq step (|htpLabelInputString| htPage '|step|))
  (setq min (|htpLabelInputString| htPage '|min|))
  (setq formula (|htpLabelInputString| htPage '|formula|))
  (setq index (|htpLabelInputString| htPage '|index|))
  (setq var (|htpLabelInputString| htPage '|variable|))
  (setq point (|htpLabelInputString| htPage '|point|))
  (setq varPart (strconc var " = " point))
  (setq minPart (strconc min ".."))
  (|bcFinish| "series" (strconc index " +-> " formula) varPart minPart step)))

defun Limit Basic Command

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]

What kind of limit do you want to compute?

[A real limit?]

The limit as the variable approaches a \em real\ value along the real axis

[A complex limit?]

The limit as the variable approaches a \em complex\ value along the complex axis
The limit as the variable approaches a \text{complex} value along any path in the complex plane"
\begin{verbatim}
|text| . "\indentrel{-17}"
|text| . "\endmenu")
\end{verbatim}

defun Real Limit Basic Command

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]

Pressing the Continue button will call the function \texttt{bcRealLimitGen} due to this line:

\begin{verbatim}
|doneButton| "Continue" |bcRealLimitGen|)
\end{verbatim}

— defun bcRealLimit —

\begin{verbatim}
defun |bcRealLimit| (a b)
(declare (special |$EmptyMode|) (ignore a b))
(\htInitPage| "Real Limit Basic Command" NIL)
(\htMakePage|
  '((\domainConditions| (\isDomain| EM |$EmptyMode|)
    (\isDomain| S ([String])) (\isDomain| F ([Float]))
    (\isDomain| SY ([Symbol]))))
  (\text| . "\newline ")
  (\text| . "\menuitemstyle{}\tab{2}")
  (\text| . "Enter the \text{function} you want to compute the limit of;")
\end{verbatim}
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

(defun Real Limit Basic Command options

[htpButtonValue p1252]
[htpLabelInputString p1254]
[bcFinish p1243]
[htInitPage p1262]
[htMakePage p1263]
[htpSetProperty p1254]
[htShowPage p1263]

— defun bcRealLimitGen —

(defun |bcRealLimitGen| (htPage)
(let ([p] [fun] [var] [loc] [page])
(cond
  ((not (eq (setq [p] (|htpButtonValue| htPage '|location|)) '|finitePoint|))
   (setq [fun] (|htpLabelInputString| htPage '|expression|))
   (setq [var] (|htpLabelInputString| htPage '|variable|))
   (setq [loc] (if (eq [p] '|plusInfinity|) "%plusInfinity" "%minusInfinity"))
   (|bcFinish| "limit" [fun] (strconc [var] " = " [loc])))
(t
(setq |page| (|htInitPage| "Real Limit Basic Command" nil))
(|htMakePage|
 '(((|text| . "Compute the limit")
 (|lispLinks|
  ("\menuitemstyle{From both directions}" " |bcRealLimitGen1| |both|)
  ("\menuitemstyle{From the right}" " |bcRealLimitGen1| |right|)
  ("\menuitemstyle{From the left}" " |bcRealLimitGen1| |left|))))
(|htpSetProperty| |page| '|fun|
 (|htpLabelInputString| htPage '|expression|)))
(|htpSetProperty| |page| '|var|
 (|htpLabelInputString| htPage '|variable|)))
(|htpSetProperty| |page| '|loc|
 (|htpLabelInputString| htPage '|point|)))
(|htShowPage|))))

---

defun bcRealLimitGen1

(htpProperty p1254)
[strconc p??]
[bcFinish p1243]

— defun bcRealLimitGen1 —

(defun |bcRealLimitGen1| (htPage key)
 (let (direction fun var loc varPart)
   (setq direction
     (cond
      ((eq key '|right|) "\right\"")
      ((eq key '|left|) "\left\"")
      (t nil)))
   (setq fun (|htpProperty| htPage '|fun|))
   (setq var (|htpProperty| htPage '|var|))
   (setq loc (|htpProperty| htPage '|loc|))
   (setq varPart (strconc var " = " loc))
   (|bcFinish| "limit" fun varPart direction)))

---

defun Complex Limit Basic Command

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
Pressing the **Continue** button will call the function **bcComplexLimitGen** due to this line:

```lisp
(doneButton) "Continue" (bcComplexLimitGen)
```

---

**defun bcComplexLimit** —

```lisp
(defun bcComplexLimit (a b)
  (declare (special $EmptyMode) (ignore a b))
  (htInitPage) "Complex Limit Basic Command" nil
  (htMakePage)
    '(
      (domainConditions
        (isDomain EM $EmptyMode)
        (isDomain SY $Symbol))
      (text . \newline\tab{2})
      (text . \menuitemstyle{}\tab{2}"
        "Enter the {\em function} you want to compute the limit of:")
      (text . \newline\tab{2})
      (bcStrings (40 "sin(a*x)/tan(b*x)" \expression EM))
      (text . \newline\tab{2})
      (bcStrings (20 0 \real F))
      (text . \newline Complex part:"
        "Enter the name of the {\em variable}: ")
      (text . \tab{37})
      (bcStrings (5 \x \variable SY))
    )
  (radioButtons (radioButtons (radioButtons))
    (text . \newline"
      "Compute the limit at"
      "A finite point:"
      (text . \newline Real part:"
        (bcStrings (20 0 \real F))
      )
      (text . \newline Complex part:"
    ))
```
defun bcComplexLimitGen

(htpLabelInputString p1254)
(htpButtonValue p1252)
[strconc p??]
[bcFinish p1243]

— defun bcComplexLimitGen —

(defun |bcComplexLimitGen| (htPage)
(let (fun var p real comp complexPart loc varPart)
(setq fun (|htpLabelInputString| htPage '|expression|))
(setq var (|htpLabelInputString| htPage '|variable|))
(setq loc
  (cond
   ((eq (setq p (|htpButtonValue| htPage '|location|)) '|finitePoint|)
    (setq real (|htpLabelInputString| htPage '|real|))
    (setq comp (|htpLabelInputString| htPage '|complex|))
    (setq complexPart
      (cond
       ((string= comp "0") "")
       ((string= comp "1") "%i")
       (t (strconc comp "*%i"))))
   (t "%infinity"))
  (cond
   ((string= real "0") (if (string= complexPart "") '|0| complexPart))
   ((string= complexPart "") real)
   (t (strconc real " + " complexPart)))))
(t "%infinity"))
(setq varPart (strconc var " = " loc))
(!!bcFinish| "complexLimit" fun varPart))

— initvars —
(setq $systemType nil)

---

defvar $numberOfEquations

---

initvars

(defvar $numberOfEquations 0)

---

defvar $solutionMethod

---

initvars

(defvar $solutionMethod nil)

---

defun Solve Basic Command

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]
[$EmptyMode p??]

---

defun bcSolve

(defun bcSolve ()
  (htInitPage "Solve Basic Command" nil)
  (htMakePage
   '((text . "What do you want to solve? ")
     (text . "\beginmenu")
     (text . "\item")
     (bcLinks)
     ("\menuitemstyle{A System Of Linear Equations}" "")
defun Linear Solve Basic Command

(defun bcLinearSolve (p nn)
  (declare (ignore p nn))
  (htInitPage "Basic Solve Command" NIL)
  (htMakePage
   '((\text . "How do you want to enter the equations?")
     (\text . "\beginmenu")
     (\text . "\item ")
     (\text . "\newline")
     (\bcLinks
       ("\menuitemstyle{Directly as equations}" "
         \bcLinearSolveEqns \{equations\}")
     (\text . "\item ")
     (\text . "\newline")
     (\bcLinks
       ("\menuitemstyle{In matrix form}" "
         \bcLinearSolveMatrix \{matrix\}")
     (\text . "\indentrel{16}\tab{0}"
     \spad{AX = B}, where \spad{A} is a matrix of coefficients and \spad{B} is a vector")
     (\text . "\indentrel{-16}\item ")
     (\text . "\endmenu"))
  (htShowPage))

---

How do you want to enter the equations?

- Directly as equations
- In matrix form $AX = B$, where $A$ is a matrix of coefficients and $B$ is a vector
defun Linear Solve Equations Basic Command

Pressing the Continue button will call the function bcLinearSolveEqns1 due to this line:

((htMakeDoneButton "Continue" 'bcLinearSolveEqns1))

--- defun bcLinearSolveEqns ---

(defun bcLinearSolveEqns (htPage p)
 (declare (ignore htPage p))
 ((htInitPage "Basic Solve Command" nil)
  (htMakePage
   '(((domainConditions (isDomain PI (PositiveInteger))))
    (inputStrings
     ;; "Enter the \{\em number\} of equations:" " 5 2
     (numberOfEquations PI)))
   (htMakeDoneButton "Continue" 'bcLinearSolveEqns1))
  (htShowPage)))

---

defun bcSystemSolve

---
Pressing the **Continue** button will call the function `bcSystemSolveEqns1` due to this line:

```lisp
(|htMakeDoneButton| "Continue" '|bcSystemSolveEqns1|)
```

---

**defun bcSystemSolve**

```lisp
(defun |bcSystemSolve| (htPage p)
  (declare (ignore htPage p))
  (|htInitPage| "Basic Solve Command" NIL)
  (|htMakePage|
    '(((|domainConditions| (|isDomain| PI (|PositiveInteger|)))
      (|inputStrings|
        ("Enter the \{\em number\} of equations:" " 5 2
        |numberOfEquations| PI)))
      (|htMakeDoneButton| "Continue" '|bcSystemSolveEqns1|)
    (|htShowPage|)))
```

---

**defun bcSolveSingle**

[htpSetProperty p1254]
[bcInputEquations p1231]

---

**defun bcSystemSolveEqns1**

[htpSetProperty p1254]
[bcInputEquations p1231]
--- defun bcSystemSolveEqns1 ---

(defun bcSystemSolveEqns1 (htPage)
  (|htpSetProperty| htPage '|systemType| '|polynomial|)
  (|htpSetProperty| htPage '|exitFunction| '|bcInputSolveInfo|)
  (|bcInputEquations| htPage '|exact|))

---

--- defun bcLinearSolveEqns1 ---

(defun bcLinearSolveEqns1 (htPage)
  (|htpSetProperty| htPage '|systemType| '|linear|)
  (|htpSetProperty| htPage '|exitFunction| '|bcLinearSolveEqnsGen|)
  (|bcInputEquations| htPage '|exact|))

---

--- defun bcInputSolveInfo ---

(defun bcInputSolveInfo (htPage)
  (let (page)
    (setq page (|htInitPage| "Solve Basic Command" (|htpPropertyList| htPage)))
    (|htpSetProperty| page '|numberOfEquations| (|htpProperty| htPage '|numberOfEquations|))
    (|htpSetProperty| page '|inputArea| (|htpInputAreaAlist| htPage))
    (|htMakePage|
      '|domainConditions| (|isDomain| PI (|PositiveInteger|)))
    (|text| . "What would you like?")
Solutions expressed in terms of *roots* of irreducible polynomials

Solutions expressed in terms of approximate real or complex *numbers*

Solutions expressed in terms of *radicals* if it is possible

---

defun bcInputEquations

[conc p??]
[stringimage p??]
[bcMakeLinearEquations p1234]
[bcMakeEquations p1234]
[htProperty p??]
[parse-integer p??]
[objValUnwrap p??]
[htInitPage p1262]
[htPropertyList p1253]
[htSetProperty p1254]
[htSay p??]
[htMakePage p1263]
[bcHt p1260]
[bcMakeUnknowns p1233]
[htMakeDoneButton p1284]
[htShowPage p1263]
[$EmptyMode p??]
--- defun bcInputEquations ---

(defun bcInputEquations (htPage solutionMethod)
  (labels (
    (f (i n linearp)
      (let (spacer prefix lnam rnam var)
        (setq spacer (cond ((> i 99) 0) ((> i 9) 1) (t 2)))
        (setq prefix
          (strconc "\newline\tab{2}{\em Equation " (stringimage i) ":"}))
        (setq prefix (strconc prefix "\space{" (stringimage spacer) "}"))
        (setq lnam (intern (strconc "l" (stringimage i))))
        (setq rnam (intern (strconc "r" (stringimage i))))
        (setq var (if linearp
                      (bcMakeLinearEquations i n)
                      (bcMakeEquations i n)))
        (cons
          (cons '|text| prefix)
          (list (list '|bcStrings| (list 30 var lnam 'p))
                '|text| . " = ")
          (list (list '|bcStrings| (list 5 '|0| rnam 'p)))))))
    (let (numEqs linearPred labelList equationPart page)
      (declare (special $EmptyMode |$bcParseOnly|))
      (setq numEqs
        (cond
          ((eq (htpProperty htPage '|systemType|) '|onePolynomial|) 1)
          (|$bcParseOnly|
            (parse-integer (htpLabelInputString htPage '|numberOfEquations|)))
          (t
            (objValUnwrap (htpLabelSpadValue htPage '|numberOfEquations|))))
      (setq linearPred (eq (htProperty htPage '|systemType|) '|linear|))
      (setq labelList
        (cond
          ((eql numEqs 1)
            '((|bcStrings| (42 (\text{"x^2+1" l1 p})) '|text| . " = ")
              (|bcStrings| (6 0 r1 P))))
          (t
            (loop for i from 1 to numEqs
              append (f i numEqs linearPred)))))))
    (setq equationPart
      (cons
        '(|domainConditions|
          (|isDomain| P (|Polynomial| |$EmptyMode|))
          (|isDomain| S (|String|))
          (|isDomain| PI (|PositiveInteger|)))))
    (setq page (htInitPage "Solve Basic Command" (htPropertyList htPage)))
    (ht SetProperty page '|numberOfEquations| numEqs)
    (ht SetProperty page '|solutionMethod| solutionMethod)
    (ht Say "\newline\menuitemstyle{}\tab{2}"
      (|bcInputEquations| i n)))))
(defun Create a variable string
  — defun bcCreateVariableString —
  (defun |bcCreateVariableString| (i)
    (format nil "x~a" i))

  — defun bcMakeUnknowns —
  (defun |bcMakeUnknowns| (number)
    (format nil "x~a")
    (loop for i from 1 to number collect (format nil "x~a" i))))
defun bcMakeEquations

(strconc p??)

(bcCreateVariableString p123)

(nreverse0 p??)

— defun bcMakeEquations —

(defun |bcMakeEquations| (i number)
  (if (eql number 1)
    (strconc (|bcCreateVariableString| 1) '|^2+1|))
  (progn
    (|bcCreateVariableString| i)
    (strconc
      (strconc (apply 'concat
        (let (t1)
          (do ((j 1 (1+ j))) ((> j number) (nreverse0 t1))
            (setq t1 (cons (strconc (|bcCreateVariableString| j) '+) t1))))
          '|1|)
        (strconc '-2* (strconc (|bcCreateVariableString| i) '|^2|))))

——

defun bcMakeLinearEquations

(bcCreateVariableString p123)

(strconc p??)

(nreverse0 p??)

— defun bcMakeLinearEquations —

(defun |bcMakeLinearEquations| (i number)
  (cond
    ((eql number 1) (|bcCreateVariableString| 1))
    ((eql number 2)
      (cond
        ((eql i 1)
          (strconc (|bcCreateVariableString| 1)
            (strconc '+ (|bcCreateVariableString| 2))))
        (t
          (strconc (|bcCreateVariableString| 1)
            (strconc '- (|bcCreateVariableString| 2)))))
      (t
        (strconc
          (strconc
            (apply 'concat
              (let (t1)
                (do ((j 1 (1+ j))) ((> j number) (nreverse0 t1))
                  (setq t1 (cons (strconc (|bcCreateVariableString| j) '+) t1))))
              '|1|)
            (-2* (strconc (|bcCreateVariableString| i) '|^2|))))))

——
(do (((j 1 (1+ j))) ((> j number) (nreverse0 t1)))
   (setq t1 (cons (strconc (|bcCreateVariableString| j) '+) t1))))
  't1)
(strconc '-2* (|bcCreateVariableString| i))))

defun bcInputEquationsEnd

If exitFunction is set, call it. [systemError p??]

— defun bcInputEquationsEnd —

(defun |bcInputEquationsEnd| (htPage)
  (let (fun)
    (if (setq fun (|htpProperty| htPage '|exitFunction|))
      (funcall fun htPage)
      (|systemError| nil))))

defun bcSolveEquationsNumerically

[htInitPage p1262]
[htMakePage p1263]
[htMakeDoneButton p1284]
[htShowPage p1263]
[htpPropertyList p1253]

— defun bcSolveEquationsNumerically —

(defun |bcSolveEquationsNumerically| (htPage p)
  (declare (ignore p))
  (|htInitPage| "Solve Basic Command" (|htpPropertyList| htPage))
  (|htMakePage|
    '((|text| . "What would you like?"))
    (|radioButtons| |choice|
      ("Real roots expressed as rational numbers" "|rr|
      ("Real roots expressed as floats" "|rf|
      ("Complex roots expressed as rational numbers" "|cr|
      ("Complex roots expressed as floats" "|cf|
      (|text| . "\vspace{1}\newline")
    (|inputStrings| ("Enter the number of desired {\em digits} of accuracy"
      "= 5 20 |acc| P1))))
  (|htMakeDoneButton| "Continue" '|bcSolveNumerically1|))
defun bcSolveNumerically1
[bcSolveEquations p1236]

— defun bcSolveNumerically1 —

(defun |bcSolveNumerically1| (htPage)
  (|bcSolveEquations| htPage '|numeric|))

defun bcSolveEquations
[htpLabelInputString p1254]
[htpButtonValue p1252]
[member p1048]
[strconc p??]
[htpProperty p1254]
[bcString2WordList p1246]
[bcwords2liststring p1246]
[bcGenEquations p1242]
[bcFinish p1243]

— defun bcSolveEquations —

(defun |bcSolveEquations| (htPage solutionMethod)
  (let (digits kind accString alist varpart r varlist varString eqnString name)
    (when (eq solutionMethod '|numeric|)
      (setq digits (|htpLabelInputString| htPage '|acc|))
      (setq kind (|htpButtonValue| htPage '|choice|))
      (setq accString
        (if (|member| kind '(|rf| |cf|))
          (strconc "1.e-" digits)
          (strconc "1/10**" digits))))
    (setq alist (|htpProperty| htPage '|inputArea|))
    (setq varpart (cadar alist))
    (setq r (cdr alist))
    (setq varlist (|bcString2WordList| varpart))
    (setq varString
      (if (cdr varlist)
(defun Linear Solve Basic Command trampoline

[bcReadMatrix p1182]
This routine is a trampoline. It calls bcReadMatrix passing the name of a call-back
routine bcLinearSolveMatrix1 to be called after the matrix has been read.

— defun bcLinearSolveMatrix —

(defun |bcLinearSolveMatrix| (htPage junk)
(declare (ignore htPage junk))
(|bcReadMatrix| '|bcLinearSolveMatrix1|))

— defun bcLinearSolveMatrix1 —

(defun |bcLinearSolveMatrix1| (htPage)
(let (page)
(setq page
  (|htInitPage| "Linear Solve Basic Command" (|htpPropertyList| htPage)))
(|htpSetProperty| page '|matrix| (|bcLinearExtractMatrix| htPage))
(|htMakePage|'
"The right side vector B is:"))
defun bcLinearExtractMatrix

(defun |bcLinearExtractMatrix| (htPage)
  (reverse (|htpInputAreaAlist| htPage)))

defun Linear Solve Basic Command options

(defun |bcLinearSolveMatrixInhomo| (htPage junk)
  (declare (ignore junk))
  (labels (}
(f (i)
 (let (spacer prefix name)
  (setq spacer (cond ((> i 99) 0) ((> i 9) 1) (t 2)))
  (setq prefix (strconc "{\em Coefficient " (stringimage i) ":"}))
  (unless (eql spacer 0)
    (setq prefix (strconc prefix " \space{" (stringimage spacer) "}")))
  (setq name (intern (strconc "c" (stringimage i))))
  (list prefix '|| 30 0 name 'p ))))
(let (nrows ncols labelList page)
 (declare (special |$EmptyMode|))
  (setq nrows (|htpProperty| htPage '|nrows|))
  (setq ncols (|htpProperty| htPage '|ncols|))
  (setq labelList (loop for i from 1 to ncols collect (f i)))
  (setq page
    (|htInitPage| "Linear Solve Basic Command" (|htpPropertyList| htPage)))
  (|htpSetProperty| page '|matrix| (|htpProperty| htPage '|matrix|))
  (|htpSetProperty| page '|nrows| nrows)
  (|htpSetProperty| page '|ncols| ncols)
  (|htMakePage|
    (list
      '(|domainConditions| (|isDomain| P (|Polynomial| |$EmptyMode|)))
      '(|text| . "Enter the right side vector B:")
      (cons
        (cons '|||inputStrings| labelList)
        (list
          '(|text| . \vspace{1}\newline Do you want:"
          '(|lispLinks|
            ("All the solutions?" " |bcLinearSolveMatrixInhomoGen| |all|
             ("A particular solution?" " |bcLinearSolveMatrixInhomoGen| |particular|)
            ))))))
    ))))
  ))))

---

defun bcLinearSolveMatrixInhomoGen
[bcLinearMatrixGen p1240]

— defun bcLinearSolveMatrixInhomoGen —

(defun |bcLinearSolveMatrixInhomoGen| (htPage key)
  (|bcLinearMatrixGen| htPage key))

---
defun bcLinearSolveMatrixHomo

(defun bcLinearSolveMatrixHomo (htPage key)
  (declare (ignore key))
  (bcLinearMatrixGen htPage 'homo))

---

defun bcLinearMatrixGen

(defun bcLinearMatrixGen (htPage key)
  (let (matform vector vecform form)
    (setq matform (bcMatrixGen htPage))
    (cond
      ((eq key 'homo)
       (bcFinish "nullSpace" matform))
      (t
       (setq vector
         (loop for x in (reverse (htpInputAreaAlist htpage))
           collect (elt x 1)))
       (setq vecform (bcVectorGen vector))
       (setq form (bcMkFunction "solve" matform (cons vecform nil)))
       (bcGen (if (eq key 'particular)
                 (strconc form ".particular")
                 form))))))

---
defun linearFinalRequest

[sayBrightly p??]
[bcQueryInteger p??]
[explainLinear p1241]

— defun linearFinalRequest —

(defun |linearFinalRequest| (nhh mat vect)
  (declare (ignore mat vect))
  (let (tt)
    (|sayBrightly| "Do you want more information on the meaning of the output")
    (|sayBrightly| " (1) no ")
    (|sayBrightly| " (2) yes ")
    (setq tt (|bcQueryInteger| 1 2 t))
    (cond
      ((eql tt 1) (|sayBrightly| "Bye Bye"))
      ((eql tt 2) (|explainLinear| nhh))))

———

defun explainLinear

[systemError p??]

— defun explainLinear —

(defun |explainLinear| (flag)
  (cond
    ((eq flag '|notHomogeneous|)
     "solve returns a particular solution and a basis for"
     "the vector space of solutions for the homogeneous part."
     "The particular solution is "failed" if one cannot be found.")
    ((eq flag '|homogeneous|)
     "solve returns a basis for"
     "the vector space of solutions for the homogeneous part")
    (t (|systemError| nil))))

———

defun finalExactRequest

[bcQueryInteger p??]
[sayBrightly p??]
[moreExactSolution p??]
[explainExact p??]

— defun finalExactRequest —

(defun finalExactRequest (equations unknowns)
  (let (tt)
    (|sayBrightly| "Do you like:")
    (|sayBrightly| " (1) the solutions how they are displayed")
    (|sayBrightly| " (2) to get ?????")
    (|sayBrightly| " (3) more information on the meaning of the output")
    (setq tt (|bcQueryInteger| 1 3 t))
    (cond
      ((eql tt 1) (|sayBrightly| "Bye Bye"))
      ((eql tt 2) (|moreExactSolution| equations unknowns))
      ((eql tt 3) (|explainExact| equations unknowns))))

defun bcLinearSolveEqnsGen

[htpInputAreaAlist p1253]
[htpLabelInputString p1254]
[bcString2WordList p1246]
[bcwords2liststring p1246]
[bcGenEquations p1242]
[bcFinish p1243]

— defun bcLinearSolveEqnsGen —

(defun bcLinearSolveEqnsGen (htPage)
  (let (vars varlist varString alist eqnString)
    (setq alist (|htpInputAreaAlist| htPage))
    (when (setq vars (|htpLabelInputString| htPage '|unknowns|))
      (setq varlist (|bcString2WordList| vars))
      (setq varString
        (if (cdr varlist) (|bcwords2liststring| varlist) (car varlist)))
      (setq alist (cdr alist))
      (setq eqnString (|bcGenEquations| alist))
      (|bcFinish| "solve" eqnString varString)))

defun bcGenEquations

[strconc p??]
[bcwords2liststring p1246]
— defun bcGenEquations —

(defun |bcGenEquations| (alist)
  (let (right left y eqnlist)
    (setq y alist)
    (loop while y do
      (setq right (elt (car y) 1))
      (setq y (cdr y))
      (setq left (elt (car y) 1))
      (setq y (cdr y))
      (setq eqnlist (cons (strconc left " = " right) eqnlist)))
    (if (cdr eqnlist)
      (|bcwords2liststring| eqnlist)
      (car eqnlist))))

---

defun Output the final formula

— defun bcFinish —

(defun |bcFinish| (&rest t1 &aux args arg name)
  (dsetq (name arg . args) t1)
  (|bcGen| (|bcMkFunction| name arg args)))

---

defun convert arguments into function call syntax

Convert verb—(bcMkFunction "test" "arg1" ("arg2" "arg3"))—to "test(arg1,arg2,arg3)"

— defun bcMkFunction —

(defun |bcMkFunction| (name arg args)
  (setq str
    (let ((result ""))
      (concatenate 'string arg
        (dolist (i args result)
          (when i
            (setq result (concatenate 'string result
              (concatenate 'string "," i))))))))
    (concatenate 'string name "(" str ")"))

---
defun bcString2HyString2

— defun bcString2HyString2 —

(defun bcString2HyString2 (s)
  (if (and (stringp s) (char= (elt s 0) #\"))
    (concatenate 'string "\\" s "\\\"")
    s))

——

defun bcString2HyString

— defun bcString2HyString —

(defun bcString2HyString (s) s)

——

defun find a character position in a string

— defun bcFindString —

(defun bcFindString (s i n char)
  (position char s :start i :end n))

——

defun Basic Command result page

[strconc p??]
[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]

— defun bcGen —

(defun bcGen (command)
  (let (string)
defun Basic Command

---

defun linkGen |

---

defun bcOptional

---

defun create a vertical space on a page

(defun bcvspace ()
  (\vspace{1}\newline ))

defun break a string into words

(defun bcString2WordList (string)
  (loop for i = 0 then (1+ j)
     as j = (position #\space string :start i)
     collect (subseq string i j)
     while j))

defun format words into a string

(defun bcwords2liststring (words)
  (format nil "\{\{A\}, \}\" words))

defun format a vector

(defun bcVectorGen (vec)
  (bcwords2liststring vec))
defun format an error message

[sayBrightlyNT p??]
[sayBrightly p??]

— defun bcError —

(defun bcError (string)
  (|sayBrightlyNT| "NOTE: ")
  (|sayBrightly| string))

— defun format intervals —

(defun format intervals

[strconc p??]

— defun bcDrawIt —

(defun bcDrawIt (ind a b)
  (strconc ind "=" a ".." b))

— defun Basic Command page not ready —

(defun Basic Command page not ready

[htInitPage p1262]
[htMakePage p1263]
[htShowPage p1263]

— defun bcNotReady —

(defun bcNotReady (htPage)
  (declare (ignore htPage))
  (|htInitPage| "Basic Command" NIL)
  (|htMakePage|
    '(((|text| . "{|centerline|\em This facility will soon be available}}")))
  (|htShowPage|))

— defun Basic Command page not ready —
defun pad a string with blanks
[\texttt{strconc \texttt{p??}}]
[\texttt{stringimage \texttt{p??}}]

--- defun \texttt{htStringPad} ---

(defun \texttt{htStringPad} (n w)
    (let (s ws)
        (setq s (\texttt{stringimage} n))
        (setq ws (\texttt{\#|} s))
        (strconc "\space{" (\texttt{stringimage} (- 1 (+ (- w ws) w))) "}\" s)))

---

defun construct a name string

Given (\texttt{"one" "two" "three"}) generate \texttt{"(one,two,three)"}
--- defun \texttt{stringList2String} ---

(defun \texttt{stringList2String} (x)
    (cond
        ((null x) "()")
        (t
            (setq str
                (let ((result ""))
                (concatenate 'string (car x)
                    (dolist (i (cdr x) result)
                        (setq result (concatenate 'string result
                            (concatenate 'string "," i))))))
                (concatenate 'string "(" str ")"))))

---

defun construct a name string

[\texttt{strconc \texttt{p??}}]
[\texttt{stringimage \texttt{p??}}]

--- defun \texttt{htMkName} ---

(defun \texttt{htMkName} (s n)
    (strconc s (\texttt{stringimage} n)))
;;; ht-util merge

defvar $bcParseOnly

    — initvars —
    (defvar |$bcParseOnly| t)

---

defvar $htLineList

    — initvars —
    (defvar |$htLineList| nil)

---

defvar $curpage

    — initvars —
    (defvar |$curPage| nil)

---

defvar $activePageList

    — initvars —
    (defvar |$activePageList| nil)
defun htpDestroyPage

— defun htpDestroyPage —

(defun htpDestroyPage| (pageName)
  (declare (special |$activePageList|))
  (SEQ (cond
    ((|member| pageName |$activePageList|)
     (EXIT (progn
          (setq pageName nil)
          (setq |$activePageList|
            (NREMOVE |$activePageList| pageName))))))))

defun htpName

— defun htpName —

(defun htpName| (htPage) (elt htPage 0))

defun htpSetName

— defun htpSetName —

(defun htpSetName| (htPage val) (setelt htPage 0 val))

defun htpDomainConditions

— defun htpDomainConditions —

(defun htpDomainConditions| (htPage) (elt htPage 1))
(defun htpSetDomainConditions (htPage val)
  (setelt htPage 1 val))

(defun htpDomainVariableAlist (htPage)
  (elt htPage 2))

(defun htpSetDomainVariableAlist (htPage val)
  (setelt htPage 2 val))

(defun htpDomainPvarSubstList (htPage)
  (elt htPage 3))

(defun htpSetDomainPvarSubstList (htPage val)
  (setelt htPage 3 val))
(defun htpSetDomainPvarSubstList (htPage val)
  (setelt htPage 3 val))

----------

defun htpRadioButtonAlist

—— defun htpRadioButtonAlist ——

(defun htpRadioButtonAlist (htPage)
  (elt htPage 4))

----------

defun htpButtonValue

—— defun htpButtonValue ——

(defun htpButtonValue (htPage groupName)
  (prog ()
    (return
      (SEQ (DO ((G166092
                  (LASSOC groupName
                      (|htpRadioButtonAlist| htPage)))
                 (CDR G166092))
              ((OR (ATOM G166092)
                   (progn (setq |buttonName| (car G166092)) nil))
               NIL)
        (SEQ (EXIT (COND
                       ((BOOT-EQUAL
                          (|stripSpaces|
                           (|htpLabelInputString| htPage
                            |buttonName|))
                          "t")
                        (EXIT (RETURN |buttonName|))))))))

----------

defun htpSetRadioButtonAlist

—— defun htpSetRadioButtonAlist ——
(defun htpSetRadioButtonAlist (htPage val)
  (setelt htPage 4 val))

---

defun htpInputAreaAlist

  — defun htpInputAreaAlist —

  (defun htpInputAreaAlist (htPage) (elt htPage 5))

---

defun htpSetInputAreaAlist

  — defun htpSetInputAreaAlist —

  (defun htpSetInputAreaAlist (htPage val)
    (setelt htPage 5 val))

---

defun htpAddInputAreaProp

  — defun htpAddInputAreaProp —

  (defun htpAddInputAreaProp (htPage label prop)
    (setelt htPage 5
      (cons (cons label (cons nil (cons nil (cons nil prop))))
        (elt htPage 5))))

---

defun htpPropertyList

  — defun htpPropertyList —
(defun htpPropertyList (htPage) (elt htPage 6))

---

defun htpProperty

— defun htpProperty —

(defun htpProperty (htPage propName)
  (LASSOC propName (elt htPage 6)))

---

defun htpSetProperty

— defun htpSetProperty —

(defun htpSetProperty (htPage propName val)
  (prog (pair)
    (return
      (progn
        (setq pair (assoc propName (elt htPage 6)))
        (cond
          (pair (rplacd pair val))
          (t
           (setelt htPage 6
             (cons (cons propName val) (elt htPage 6))))))))

---

defun htpLabelInputString

— defun htpLabelInputString —

(defun htpLabelInputString (htPage label)
  (prog (props s)
    (return
      (progn
        (setq props
          (assoc label (htpInputAreaAlist htPage)))
        (cond
(and props (stringp (setq s (elt props 0))))
  (cond
    ((equal s "") s)
    (t (trimString s)))
  (t nil)))))))

defun htpLabelFilteredInputString

— defun htpLabelFilteredInputString —

(defun htpLabelFilteredInputString (htPage label)
  (prog (props)
    (return
      (progn
        (setq props
          (LASSOC label (htpInputAreaAlist htPage)))
        (cond
          (props (cond
            ((and (> (length props) 5) (elt props 6))
              (funcall (symbol-function (elt props 6))
                (elt props 0)))
            (t (replacePercentByDollar (elt props 0)))))
          (t nil))))))

defun replacePercentByDollar,fn

— defun replacePercentByDollar,fn —

(defun replacePercentByDollar,fn (s i n)
  (prog (m)
    (return
      (SEQ (if (> i n) (EXIT ""))
        (if (> (setq m (charPosition \% s i)) n)
          (EXIT (SUBSTRING s i nil)))
        (EXIT (STRCONC (SUBSTRING s i (- m i)) "\$" (replacePercentByDollar,fn s (1+ m) n))))))))
defun replacePercentByDollar

— defun replacePercentByDollar —

(defun replacePercentByDollar (s)
  (replacePercentByDollar fn s 0 (maxindex s)))

defun htpSetLabelInputString

— defun htpSetLabelInputString —

(defun htpSetLabelInputString (htPage label val)
  (prog (props)
    (return
     (progn
       (progn
         (setq props
           (LASSOC label (htpInputAreaAlist htPage)))
         (cond
           (props (setelt props 0 (stringimage val)))
           (t nil)))))))

defun htpLabelSpadValue

— defun htpLabelSpadValue —

(defun htpLabelSpadValue (htPage label)
  (prog (props)
    (return
     (progn
       (setq props
         (LASSOC label (htpInputAreaAlist htPage)))
       (cond (props (elt props 1)) (t nil))))))
defun htpSetLabelSpadValue

— defun htpSetLabelSpadValue —

(defun |htpSetLabelSpadValue| (htPage label val)
  (prog (props)
    (return
      (progn
        (setq props
          (LASSOC label (|htpInputAreaAlist| htPage)))
        (cond (props (setelt props 1 |val|)) (t nil)))))))

——

defun htpLabelErrorMsg

— defun htpLabelErrorMsg —

(defun |htpLabelErrorMsg| (htPage label)
  (prog (props)
    (return
      (progn
        (setq props
          (LASSOC label (|htpInputAreaAlist| htPage)))
        (cond (props (elt props 2)) (t nil)))))))

——

defun htpSetLabelErrorMsg

— defun htpSetLabelErrorMsg —

(defun |htpSetLabelErrorMsg| (htPage label val)
  (prog (props)
    (return
      (progn
        (setq props
          (LASSOC label (|htpInputAreaAlist| htPage)))
        (cond (props (setelt props 2 val)) (t nil)))))))

——
defun htpLabelType

— defun htpLabelType —

(defun htpLabelType (htPage label)
  (prog (props)
    (return
     (progn
      (setq props
        (LASSOC label (htpInputAreaAlist htPage)))
      (cond (props (elt props 3)) (t nil))))))

-----

defun htpLabelDefault

— defun htpLabelDefault —

(defun htpLabelDefault (htPage label)
  (prog (msg props)
    (return
     (cond
      ((setq msg (htpLabelInputString htPage label))
       (cond
        ((equal msg "t") 1)
        ((equal msg "nil") 0)
        (t msg)))
      (t
       (setq props
         (LASSOC label (htpInputAreaAlist htPage)))
       (cond (props (elt props 4)) (t nil)))))))

-----

defun htpLabelSpadType

— defun htpLabelSpadType —

(defun htpLabelSpadType (htPage label)
  (prog (props)
    (return
     (progn
(setq props
  (LASSOC label (|htpInputAreaAlist| htPage)))
(cond (props (elt props 5)) (t nil))))

defun htpLabelFilter

— defun htpLabelFilter —

(defun |htpLabelFilter| (htPage label)
  (prog (props)
    (return
      (progn
        (setq props
          (LASSOC label (|htpInputAreaAlist| htPage)))
        (cond (props (elt props 6)) (t nil))))))

defun htpPageDescription

— defun htpPageDescription —

(defun |htpPageDescription| (htPage) (elt htPage 7))

defun htpSetPageDescription

— defun htpSetPageDescription —

(defun |htpSetPageDescription| (htPage pageDescription)
  (setelt htPage 7 pageDescription))
defun htpAddToPageDescription

--- defun htpAddToPageDescription ---

(defun htpAddToPageDescription (htPage pageDescrip)
  (setelt htPage 7
    (NCONC (NREVERSE (COPY-LIST pageDescrip)) (elt htPage 7))))

---

defun issue a single hypertex line or group of lines

--- defun iht ---

(defun iht (line)
  (declare (special $htLineList $newPage))
  (cond
    ($newPage nil)
    ((consp line)
      (setq $htLineList
        (NCONC (NREVERSE (mapStringize (COPY-LIST line)))
          $htLineList))
    (t
      (setq $htLineList
        (cons (basicStringize line) $htLineList)))))

---

defun bcHt

--- defun bcHt ---

(defun bcHt (line)
  (declare (special $curPage $newPage))
  (progn
    (iht line)
    (cond
      ((consp line)
        (cond
          ($newPage
            (htpAddToPageDescription $curPage
              (cons (cons ’text line) nil)))
      (t
        (setq $htLineList
          (cons (basicStringize line) $htLineList)))))

---
defun bcIssueHt

— defun bcIssueHt —

(defun bcIssueHt (line)
  (cond ((consp line) (htMakePage1 line)) (t (iht line))))

defun mapStringize

— defun mapStringize —

(defun mapStringize (z)
  (cond ((atom z) z)
        (t (rplaca z (basicStringize (car z)))
           (rplacd z (mapStringize (cdr z))) z)))

defun basicStringize

— defun basicStringize —

(defun basicStringize (s)
  (cond ((stringp s)
          (cond ((equal s "\$" "\%")
                  ((equal s "\{\em $\}" "\{\em \%\}")
                   (t s))))))
defun stringize

— defun stringize —

(defun stringize (s)
  (cond ((stringp s) s) (t (princ-to-string s))))

defun htInitPage

— defun htInitPage —

(defun htInitPage (title propList)
  (declare (special $curPage))
  (progn
    (htInitPageNoScroll propList title)
    (htSayStandard "$\beginscroll \\

    ($curPage$))

defun htAddHeading

— defun htAddHeading —

(defun htAddHeading (title)
  (declare (special $curPage))
  (progn
    (htNewPage title)
    ($curPage$))

  "$\beginscroll \\

    ($curPage$)\%")
  (t (princ-to-string s))))

((eq s '$) "$\%")
  (t (princ-to-string s))))
defun htShowPage

--- defun htShowPage ---

(defun |htShowPage| ()
  (|htSayStandard| "\endscroll")
  (|htShowPageNoScroll|))

defun show the page which has been computed

--- defun htShowPageNoScroll ---

(defun |htShowPageNoScroll| ()
  (prog (line)
    (declare (special |$htLineList| |$curPage| |$newPage|))
    (return
      (progn
        (|htSayStandard| "\\autobuttons")
        (|htpSetPageDescription| |$curPage|
          (NREVERSE (|htpPageDescription| |$curPage|)))
        (setq |$newPage| nil)
        (setq |$htLineList| nil)
        (|htMakePage| (|htpPageDescription| |$curPage|))
        (setq line (apply (|function| CONCAT) (NREVERSE |$htLineList|)))
        (|issueHT| line)
        (|endHTPage|))))

defun make a page given the description in itemList

--- defun htMakePage ---

(defun |htMakePage| (itemList)
  (declare (special |$curPage| |$newPage|))
  (progn
    (cond
      (|$newPage| (|htpAddToPageDescription| |$curPage| itemList)))
    (|htMakePage| itemList)))
defun htMakePage1

— defun htMakePage1 —

(defun htMakePage1 (itemList)
  (prog (itemType items)
    (return
      (SEQ (DO ((G166261 itemList (CDR G166261)) (G166253 NIL))
        ((OR (ATOM G166261)
            (PROGN (SETQ G166253 (CAR G166261)) NIL)
            (PROGN
              (PROGN
                (setq itemType (CAR G166253))
                (setq items (CDR G166253))
                G166253)
              NIL))
        NIL)
      (SEQ (EXIT (COND
                   ((eq itemType '|text|)
                    (iht items))
                   ((eq itemType '|lispLinks|)
                    (htLispLinks items))
                   ((eq itemType '|lispmemoLinks|)
                    (htLispMemoLinks items))
                   ((eq itemType '|bcLinks|)
                    (htBcLinks items))
                   ((eq itemType '|bcLinksNS|)
                    (htBcLinks items t))
                   ((eq itemType '|bcLispLinks|)
                    (htBcLispLinks items))
                   ((eq itemType '|radioButtons|)
                    (htRadioButtons items))
                   ((eq itemType '|bcRadioButtons|)
                    (htBcRadioButtons items))
                   ((eq itemType '|inputStrings|)
                    (htInputStrings items))
                   ((eq itemType '|domainConditions|)
                    (htProcessDomainConditions items))
                   ((eq itemType '|bcStrings|)
                    (htProcessBcStrings items))
                   ((eq itemType '|toggleButtons|)
                    (htProcessToggleButtons items))
                   ((eq itemType '|bcButtons|)
                    (htProcessBcButtons items))
                   ((eq itemType '|doneButton|)
                    (htProcessDoneButton items))
                   ((eq itemType '|doitButton|)
                    (htProcessDoitButton items))
                   (t
                    NIL)))))))
  NIL))
  (SEQ (EXIT (COND
               ((eq itemType '|text|)
                (iht items))
               ((eq itemType '|lispLinks|)
                (htLispLinks items))
               ((eq itemType '|lispmemoLinks|)
                (htLispMemoLinks items))
               ((eq itemType '|bcLinks|)
                (htBcLinks items))
               ((eq itemType '|bcLinksNS|)
                (htBcLinks items t))
               ((eq itemType '|bcLispLinks|)
                (htBcLispLinks items))
               ((eq itemType '|radioButtons|)
                (htRadioButtons items))
               ((eq itemType '|bcRadioButtons|)
                (htBcRadioButtons items))
               ((eq itemType '|inputStrings|)
                (htInputStrings items))
               ((eq itemType '|domainConditions|)
                (htProcessDomainConditions items))
               ((eq itemType '|bcStrings|)
                (htProcessBcStrings items))
               ((eq itemType '|toggleButtons|)
                (htProcessToggleButtons items))
               ((eq itemType '|bcButtons|)
                (htProcessBcButtons items))
               ((eq itemType '|doneButton|)
                (htProcessDoneButton items))
               ((eq itemType '|doitButton|)
                (htProcessDoitButton items))
               (t
                NIL))))))
  NIL))
  (SEQ (EXIT (COND
               ((eq itemType '|text|)
                (iht items))
               ((eq itemType '|lispLinks|)
                (htLispLinks items))
               ((eq itemType '|lispmemoLinks|)
                (htLispMemoLinks items))
               ((eq itemType '|bcLinks|)
                (htBcLinks items))
               ((eq itemType '|bcLinksNS|)
                (htBcLinks items t))
               ((eq itemType '|bcLispLinks|)
                (htBcLispLinks items))
               ((eq itemType '|radioButtons|)
                (htRadioButtons items))
               ((eq itemType '|bcRadioButtons|)
                (htBcRadioButtons items))
               ((eq itemType '|inputStrings|)
                (htInputStrings items))
               ((eq itemType '|domainConditions|)
                (htProcessDomainConditions items))
               ((eq itemType '|bcStrings|)
                (htProcessBcStrings items))
               ((eq itemType '|toggleButtons|)
                (htProcessToggleButtons items))
               ((eq itemType '|bcButtons|)
                (htProcessBcButtons items))
               ((eq itemType '|doneButton|)
                (htProcessDoneButton items))
               ((eq itemType '|doitButton|)
                (htProcessDoitButton items))
               (t
                NIL))))
(cons "unknown itemType" (cons itemType nil)))

---

defun htMakeErrorPage

— defun htMakeErrorPage —

(defun htMakeErrorPage (htPage)
  (prog (line)
    (declare (special $curPage $htLineList $newPage))
    (return
      (progn
        (setq $newPage nil)
        (setq $htLineList nil)
        (setq $curPage htPage)
        (htMakePage (htPageDescription htPage))
        (setq line (apply (function CONCAT) (NREVERSE $htLineList)))
        (issueHT line)
        (endHTPage)))))

---

defun htQuote

— defun htQuote —

(defun htQuote (s)
  (progn
    (iht "\"
    (iht s)
    (iht "\")))

---

defun htProcessToggleButtons

— defun htProcessToggleButtons —
(defun htProcessToggleButtons | buttons|)
  (prog (message info defaultValue buttonName)
    (return
     (SEQ (progn
       (iht "\newline\indent{5}\beginitems ")
       (DO ((G166302 buttons (CDR G166302))
           (G166286 nil))
         ((OR (ATOM G166302)
               (progn (setq G166286 (car G166302)) nil)
               (progn
                 (setq message (car G166286))
                 (setq info (CADR G166286))
                 (setq defaultValue (CADDR G166286))
                 (setq buttonName (CADDDR G166286))
                 G166286)
               nil))
         nil)
       nil)
     (SEQ (EXIT (progn
         (cond ((NULL (LASSOC buttonName
                       (|htpInputAreaAlist| |$curPage|)))
               (|setUpDefault| buttonName
                (cons '|button|
                  (cons defaultValue nil))))))
         (iht (cons
            "\item{\em\inputbox["
            (cons
              (|htpLabelDefault| |$curPage| buttonName)
              (cons "]{
                (cons buttonName
                  (cons
                    "}{\htbmfile{pick}}{\htbmfile{unpick}}\space{}" nil))))))
         (|bcIssueHt| message)
         (iht "\space{}}")
         (|bcIssueHt| info))))
       (iht "\enditems\indent{0} ")]))))

——

defun htProcessBcButtons

—— defun htProcessBcButtons ——
(defun |htProcessBcButtons| (buttons)
  (prog (defaultValue buttonName k)
    (declare (special |$curPage|))
    (return
      (SEQ (DO ((G166328 buttons (CDR G166328)) (G166317 nil))
        ((OR (ATOM G166328)
            (progn (setq G166317 (car G166328)) nil)
            (progn
              (setq defaultValue (car G166317))
              (setq buttonName (CADR G166317))
              (G166317)
              nil))
       nil)
      (SEQ (EXIT (progn
          (cond
            ((NULL (LASSOC buttonName
              (|htpInputAreaAlist| |$curPage|)))
              (|setUpDefault| buttonName
              (cons '\|button|
                (cons defaultValue nil))))
            (setq k
              (|htpLabelDefault| |$curPage|
              buttonName))
            (cond
              ((EQL k 0)
                (|iht| (cons "\off{
                  (cons buttonName
                  (cons "}" nil))))))
              ((EQL k 1)
                (|iht| (cons "\on{
                  (cons buttonName
                  (cons "}" nil))))))
            (t
                (|iht| (cons "\inputbox[
                  (cons
                    (|htpLabelDefault| |$curPage|
                    buttonName)
                    (cons "}"
                    (cons buttonName
                    (cons
                    nil)))))))))))))))
      (\htbmf{pick} (\htbmf{unpick}})
defun htProcessBcStrings

— defun htProcessBcStrings —

(defun htProcessBcStrings (strings)
  (prog (numChars default stringName spadType filter mess2)
    (declare (special $curPage)))
    (return
      (seq (do ((G166358 strings (cdr G166358))
                   (G166343 nil))
              ((or (atom G166358)
                   (progn (setq G166343 (car G166358)) nil)
                   (progn
                     (setq numChars (car G166343))
                     (setq default (cadr G166343))
                     (setq stringName (caddr G166343))
                     (setq spadType (cadddr G166343))
                     (setq filter (cadddr G166343))
                     G166343)
                   nil))
       nil)
     (seq (exit (progn
                   (setq mess2 "")
                   (cond
                     ((null (lassoc stringName (htpInputAreaAlist "$curPage")))
                      (setq default (cons 'string)
                        (cons numChars
                          (cons default
                            (cons spadType
                              (cons filter nil))))))
                   (cond
                     ((htpLabelErrorMsg "$curPage" stringName)
                      (iht (cons \"\centerline{\em \" (cons
                        (htpLabelErrorMsg "$curPage" stringName)
                      (cons "}\}" nil))))
                     (setq mess2
                      (concat mess2 (bcSadFaces)))
                      (htpSetLabelErrorMsg "$curPage" stringName nil)))
                     (iht (cons \"\inputstring{" (cons stringName (cons "){}" (cons numChars (cons "){}"))))
defun bcSadFaces

— defun bcSadFaces —

(defun bcSadFaces ()
  "\space{1}{\em\htbitmap{error}\htbitmap{error}\htbitmap{error}}")

defun htLispLinks

— defun htLispLinks —

(defun htLispLinks (&REST G166422 &AUX option links)
  (setq links (car G166422))
  (setq option (cdr G166422))
  (prog (t1 options indent message info func value call)
    (return
     (SEQ (progn
       (setq t1 (beforeAfter '|options| links))
       (setq links (car t1))
       (setq options (cadr t1))
       (setq indent (or (LASSOC '|indent| options) 5))
       (|iht| "\newline\indent{")
       (|iht| (stringize indent))
       (|iht| "}\beginitems")
       (DO ((G166403 links (CDR G166403)) (G166387 nil))
        ((or (atom G166403)
            (progn (setq G166387 (car G166403)) nil)
            (progn
              (setq message (car G166387))
              (setq info (cadr G166387))
              (setq func (caddr G166387))
              (setq value (cdddr G166387))
            ))
        )
      )))
  )
)
defun htLispMemoLinks

— defun htLispMemoLinks —

(defun htLispMemoLinks (links) (htLispLinks links t))

defun htBcLinks

— defun htBcLinks —

(defun htBcLinks (REST G166465 &AUX options links)
    (setq links (car G166465))
    (setq options (cdr G166465))
    (prog (skipStateInfo? t1 message info func value)
        (return
            (SEQ (progn
                (setq |skipStateInfo?| (IFCAR options))
                (setq t1 (|beforeAfter| 'options |links|))
                (setq links (car t1))
                (setq options (cdr t1))
                (DO ((G166447 links (CDR G166447)) (G166434 nil))
                    ((or (atom G166447))
                        (|iht| "\item["))
                    (setq call
                        (cond
                            ((IFCAR option)
                                "\lispmemolink")
                            (t
                                "\lispdownlink"))
                        (|htMakeButton| call message
                            (|mkCurryFun| func value))
                        (|iht| (cons "\space{}" nil))
                        (|bcIssueHt| info)))))))
                (|iht| "\enditems\indent{0} "))))}))

(progn (setq G166434 (car G166447)) nil)
(progn
  (setq message (car G166434))
  (setq info (cadr G166434))
  (setq func (caddr G166434))
  (setq value (cdddr G166434))
  G166434)
nil)
(SEQ (EXIT (progn
  (|htMakeButton|
    "\\lisplink" message
    (|mkCurryFun| func value)
    skipStateInfo?)
  (|bcIssueHt| info)))))))))

defun htBcLispLinks
  
  — defun htBcLispLinks —

(defun |htBcLispLinks| (links)
  (prog (t1 options message info func value)
    (return
      (SEQ (progn
        (setq t1 (|beforeAfter| '|options| links))
        (setq links (car t1))
        (setq options (cadr t1))
        (DO ((G166487 links (cdr G166487)) (G166474 nil))
          ((or (atom G166487)
              (progn (setq G166474 (car G166487)) nil)
              (progn
                (setq message (car G166474))
                (setq info (cadr G166474))
                (setq func (caddr G166474))
                (setq value (cdddr G166474))
                G166474)
              nil))
            nil)
      (SEQ (EXIT (progn
        (|htMakeButton| "\\lisplink"
          message
          (|mkCurryFun| func value))
        (|bcIssueHt| info))))))))
defun beforeAfter

— defun beforeAfter —

(defun beforeAfter (x u)
  (prog (y r)
    (return
      (SEQ (cons (prog (G166514)
                    (setq G166514 nil)
                    (return
                      (DO ((G166504 u (CDR G166504)))
                        ((or (atom G166504)
                            (progn
                              (progn
                                (setq y (car G166504))
                                (setq r (cdr G166504))
                                G166504)
                              nil)
                            (null (NEQUAL x y)))
                          (NREVERSE0 G166514))
                        (SEQ (EXIT (setq G166514 (cons y G166514)))))
                        (cons r nil)))))))

defun mkCurryFun

— defun mkCurryFun —

(defun mkCurryFun (fun val)
  (prog (name code)
    (return
      (progn
        (setq name (gentemp))
        (setq code
          (cons 'defun
            (cons name
              (cons '(arg)
                (cons
                  (cons 'apply
                    nil))))))))
(cons (mkq fun)
  (cons
    (cons 'cons
      (cons 'arg
        (cons (mkq val) nil)))
    nil)))
  nil)))))

(eval code)
(name))))

——

defun htRadioButtons

— defun htRadioButtons —

(defun |htRadioButtons| (G166546)
  (prog (groupName buttons boxesName message info buttonName defaultValue)
    (declare (special |$curPage|))
    (return
     (SEQ (progn
       (setq groupName (car G166546))
     (setq buttons (cdr G166546))
     (|htpSetRadioButtonAlist| |$curPage|
       (cons (cons groupName (|buttonNames| buttons))
         (|htpRadioButtonAlist| |$curPage|)))
     (setq boxesName (gentemp))
     (|iht| (cons "\newline\indent{5}\radioboxes{"
       (cons boxesName
         (cons "}{{\htbmf{pick}}}{{\htbmf{unpick}}}{\beginitems "
           nil))))
     (setq defaultValue "1")
     (DO ((G166568 buttons (cdr G166568))
       (G166540 nil))
       ((or (atom G166568)
         (progn (setq G166568 (car G166568)) nil)
         (progn
           (setq message (car G166540))
           (setq info (cadr G166540))
           (setq buttonName (caddr G166540))
           G166540)
         nil))
       (SEQ (EXIT (progn
         (cond
defun htBcRadioButtons

— defun htBcRadioButtons —

(defun |htBcRadioButtons| (G166594)
  (prog (groupName buttons boxesName message info buttonName defaultValue)
    (declare (special |$curPage|)))
    (return
     (SEQ (progn
       (setq groupName (car G166594))
       (setq buttons (cdr G166594))
       ([htpSetRadioButtonAlist] |$curPage|
        (cons (cons groupName (|buttonNames| buttons))
        ([htpRadioButtonAlist] |$curPage|)))
       (setq boxesName (gentemp))
       ([iht] (cons "\radioboxes{"
         (cons boxesName
          (cons "}{\htbmfile{pick}}{\htbmfile{unpick}} "
          nil))))
       (setq defaultValue "1")
       (DO ((G166616 buttons (cdr G166616))
          (G166588 nil)))
     )))
    )))
)
(((or (atom G166616)
  (progn (setq G166588 (car G166616)) nil)
  (progn
    (progn
      (setq message (car G166588))
      (setq info (cadr G166588))
      (setq buttonName (caddr G166588))
      G166588)
    nil)
  nil)
 SEQ (EXIT (progn
  (cond
    ((null (LASSOC buttonName
      (|htpInputAreaAlist| |$curPage|)))
     (|setUpDefault| buttonName
      (cons '|button|
        (cons defaultValue nil))
     (setq defaultValue
       "0"))
      (|iht| (cons
        "\\em\\radiobox[
        (cons
          (|htpLabelDefault| |$curPage| buttonName)
        (cons "]\\{"
          (cons buttonName
            (cons "]\\{"
              (cons boxesName
                (cons "]\\{" nil))))))))
     (|bcIssueHt| message)
     (|iht| "\\space{}")
     (|bcIssueHt| info))))))))

---

defun setUpDefault

--- defun setUpDefault ---

(defun |setUpDefault| (name props)
  (declare (special |$curPage|))
  (|htpAddInputAreaProp| |$curPage| name props))

---
defun buttonNames

— defun buttonNames —

(defun buttonNames (buttons)
  (prog (buttonName)
    (return
      (prog (G166645)
        (setq G166645 nil)
        (return
          (DO ((G166651 buttons (cdr G166651))
               (G166637 nil))
            ((or (atom G166651)
                 (progn (setq G166637 (car G166651)) nil)
                 (progn
                   (progn
                     (setq buttonName (caddr G166637))
                     G166637)
                   nil))
               (NREVERSE G166645))
            (SEQ (EXIT (setq G166645
                          (cons buttonName G166645))))))))
)

defun htInputStrings

— defun htInputStrings —

(defun htInputStrings (strings)
  (prog (mess1 numChars default stringName spadType filter mess2)
    (declare (special $curPage!))
    (return
      (SEQ (progn
        (iht "\newline\indent{5}\beginitems ")
        (DO ((G166685 strings (cdr G166685))
             (G166665 nil))
            ((or (atom G166685)
                 (progn (setq G166665 (car G166685)) nil)
                 (progn
                   (progn
                     (setq mess1 (car G166665))
                     (setq mess2 (cadr G166665))
                     (setq numChars (caddr G166665))
                     (setq default (cadddr G166665))
                     (setq stringName "")
                     (setq spadType "")
                     (setq filter "")
                     (setq mess2 "")
                     (setq mess1 "")
                     (setq default "")
                     (setq numChars "")
                     (setq stringName "")
                     (setq spadType "")
                     (setq filter ""))
               (NREVERSE G166645))
            (SEQ (EXIT (setq G166645
                          (cons buttonName G166645))))))))
(setq stringName
  (car (cdddr G166665)))
(setq spadType
  (cadr (cdddr G166665)))
(setq filter (cddr (cdddr G166665)))
G166665)
nil))
(SEQ (EXIT (progn
  (cond
    ((null (LASSOC stringName
        (|htpInputAreaAlist| |$curPage|)))
     (|setUpDefault| stringName
        (cons '|'string|
          (cons default
            (cons spadType
              (cons filter nil))))))))
  (cond
    ((|htpLabelErrorMsg| |$curPage| stringName)
     (|iht| (cons "\centerline{{{\em 
          (cons
            (|htpLabelErrorMsg| |$curPage| stringName)
            (cons "}}" nil)))))
     (setq mess2
       (CONCAT mess2 (|bcSadFaces|)))
     (|htpSetLabelErrorMsg| |$curPage| stringName nil))
     (|iht| "\item ")
     (|bcIssueHt| mess1)
     (|iht| (cons "\inputstring{{
          (cons stringName
            (cons "}){"
              (cons numChars
                (cons "}){"
                  (cons
                    (|htpLabelDefault| |$curPage| stringName)
                    (cons "}) " nil)))))))))
     (|bcIssueHt| mess2))))
     (|iht| "\enditems\indent{0}\newline "))))))

----------

(defun htProcessDomainConditions
— defun htProcessDomainConditions —

(defun htProcessDomainConditions (condList)
  (declare (special $curPage)))
  (progn
    (htpSetDomainConditions $curPage
      (renamePatternVariables condList))
    (htpSetDomainVariableAlist $curPage
      (computeDomainVariableAlist)))

——

defun renamePatternVariables

— defun renamePatternVariables —

(defun renamePatternVariables (condList)
  (declare (special $curPage $PatternVariableList))
  (progn
    (htpSetDomainPvarSubstList $curPage
      (renamePatternVariables1 condList nil
        $PatternVariableList))
    (substFromAlist condList (htpDomainPvarSubstList $curPage)))))

——

defun renamePatternVariables1

— defun renamePatternVariables1 —

(defun renamePatternVariables1 (condList substList patVars)
  (prog (restConds pattern t2 pv t3 cond nsubst)
    (declare (special $EmptyMode))
    (return
      (cond
        ((null condList) substList)
        (t (setq cond (car condList))
          (setq restConds (cdr condList))
          (cond
            ((or (and (consp cond) (eq (qcar cond) 'isDomain))
              (progn
                (setq t2 (qcdr cond))
                (and (consp t2)
(progn
  (setq pv (qcar t2))
  (setq t3 (qcdr t2))
  (and (consp t3)
    (eq (qcdr t3) nil)
    (progn
      (setq pattern
        (qcar t3))
      t)))))
(and (consp cond) (eq (qcar cond) '|ofCategory|)
  (progn
    (setq t2 (qcdr cond))
    (and (consp t2)
      (progn
        (setq pv (qcar t2))
        (setq t3 (qcdr t2))
        (and (consp t3)
          (eq (qcdr t3) nil)
          (progn
            (setq pattern
              (qcar t3))
            t)))))
(and (consp cond) (eq (qcar cond) '|Satisfies|)
  (progn
    (setq t2 (qcdr cond))
    (and (consp t2)
      (progn
        (setq pv (qcar t2))
        (setq t3 (qcdr t2))
        (and (consp t3)
          (eq (qcdr t3) nil)
          (progn
            (setq cond (qcar t3))
            t)))))
(cond
  ((equal pv |$EmptyMode|)
    (setq nsubst substList))
  (t
    (setq nsubst
      (cons (cons pv (car patVars)) substList))))
  (renamePatternVariables1 restConds nsubst
    (cdr patVars)))
  (t substList)))))))
defun substFromAlist

--- defun substFromAlist ---

(defun substFromAlist (z substAlist)
  (prog (pvar replace)
    (return
      (SEQ (progn
        (DO ((G166792 substAlist (cdr G166792))
             (G166783 nil))
           ((or (atom G166792)
                (progn (setq G166783 (car G166792)) nil)
                (progn
                  (setq pvar (car G166783))
                  (setq replace (cdr G166783))
                  (G166783 nil))
                nil)
           nil)
        (SEQ (EXIT (setq z (subst replace pvar z :test #'equal))))))))
    z)))

defun computeDomainVariableAlist

--- defun computeDomainVariableAlist ---

(defun computeDomainVariableAlist ()
  (prog (pvar)
    (declare (special _$curPage_))
    (return
      (SEQ (prog (G166813)
                (setq G166813 nil)
                (return
                  (DO ((G166819 (htpDomainPvarSubstList _$curPage_) (cdr G166819))
                       (G166805 NIL))
                    ((or (atom G166819)
                         (progn (setq G166805 (car G166819)) nil)
                         (progn
                           (setq pvar (cdr G166805))
                           (G166805)
                           NIL))))))
    )
    )
defun pvarCondList

— defun pvarCondList —

(defun |pvarCondList| (pvar)
  (declare (special |$curPage|))
  (NREVERSE
   ([|pvarCondList1| (cons pvar nil) nil
    ([lhtpDomainConditions| |$curPage|]))))

———

defun pvarCondList1

— defun pvarCondList1 —

(defun |pvarCondList1| (pvarList activeConds condList)
  (prog (cond restConds t2 pv t3 pattern)
    (return
     (cond
      (null condList) activeConds)
      (t (setq cond (car condList))
       (setq restConds (cdr condList))
       (cond
        ((and (consp cond)
          (progn
           (setq t2 (qcdr cond))
           (and (consp t2)
            (progn
             (setq pv (qcar t2))
             (setq t3 (qcdr t2))
             (and (consp t3)
              (eq (qcdr t3) nil)
              (progn
               (setq pattern (qcar t3)))))))))
      (t (setq pattern (cadr condList))))))
defun pvarsOfPattern

(defun pvarsOfPattern (pattern)
  (prog ()
    (declare (special $PatternVariableList))
    (return
      (SEQ (cond
        ((null (listp pattern)) nil)
        (t
          (prog (G166869)
            (setq G166869 nil)
            (return
              (DO ((G166875 (cdr pattern) (cdr G166875))
                (pvar nil))
                ((or (atom G166875)
                    (progn (setq pvar (car G166875)) nil))
                  (NREVERSE0 G166869))
              (SEQ (EXIT (cond
                            ((member pvar
                              $PatternVariableList)
                             (setq G166869
                                   (cons pvar G166869)))))
               ))))))))))

defun htMakeTemplates,substLabel

(defun htMakeTemplates,substLabel (i template)
  (SEQ (if (consp template)
            (EXIT (intern (CONCAT (car template) (princ-to-string i))
            )))
defun htMakeTemplates

— defun htMakeTemplates —
(defun |htMakeTemplates| (templateList numLabels)
  (prog ()
    (return
      (SEQ (progn
        (setq templateList
          (prog (G166895)
            (setq G166895 nil)
            (return
              (DO ((G166900 templateList
                (CDR G166900))
                (template nil))
                ((or (atom G166900)
                  (progn
                    (setq template (car G166900))
                    nil))
                (NREVERSE0 G166895))
              (SEQ (EXIT (setq G166895
                (cons
                  (|templateParts| template)
                G166895)))))
            (prog (G166910)
              (setq G166910 nil)
              (return
                (DO ((i 1 (1+ i)))
                  ((qsgreaterp i numLabels)
                    (NREVERSE0 G166910))
                  (SEQ (EXIT (setq G166910
                    (cons
                      (prog (G166922)
                        (setq G166922 nil)
                        (return
                          (DO ((G166927 templateList
                            (CDR G166927))
                            (template nil))
                          ((or (atom G166927)
                            (progn
                              (setq template
                              (cdr template))))))))))))))
  (prog (G166905)
    (setq G166905 nil)
    (return
      (DO ((G166906 templateList
        (CDR G166906))
        (template nil))
        ((or (atom G166906)
          (progn
            (setq template (car G166906))
            nil))
          (NREVERSE0 G166905))
      (SEQ (EXIT (setq G166905
        (cons
          (|templateParts| template)
        G166905)))))
  (prog (G166916)
    (setq G166916 nil)
    (return
      (DO ((i 1 (1+ i)))
        ((qsgreaterp i numLabels)
          (NREVERSE0 G166916))
        (SEQ (EXIT (setq G166916
          (cons
            (prog (G166928)
              (setq G166928 nil)
              (return
                (DO ((G166929 templateList
                  (CDR G166929))
                  (template nil))
                  ((or (atom G166929)
                    (progn
                      (setq template
                      (cdr template))))))))))))
defun templateParts

--- defun templateParts ---

(defun |templateParts| (template)
  (prog (i)
    (return
     (cond
       ((null (stringp template)) template)
       (t (setq i (SEARCH "%l" template))
         (cond
          ((null i) template)
          (t
           (cons (SUBSEQ template 0 i)
                 (SUBSEQ template (+ i 2))))))))))

---

defun htMakeDoneButton

--- defun htMakeDoneButton ---

(defun |htMakeDoneButton| (message func)
  (progn
   (\bcHt| "\newline\vspace{1}\centerline{"
     (cond
       ((equal message "Continue")
        (\bchMakeButton| "\lispdownlink" \\
                     '|\\\ContinueBitmap| func))

   ...)

   ...)

   ...)

   ...)

   ...)
defun htProcessDoneButton

— defun htProcessDoneButton —

(defun htProcessDoneButton (G166950)
  (prog (label func)
    (return
     (progn
      (setq label (car G166950))
      (setq func (cadr G166950))
      (|iht| "\newline\vspace{1}\centerline{"
        (cond
          ((equal label "Continue")
            (|htMakeButton| "\lispdownlink" |\\ContinueBitmap| func))
          ((equal label "Push to enter names")
            (|htMakeButton| "\lispdownlink" "\ControlBitmap{clicktoset}" func))
          t
            (|htMakeButton| "\lispdownlink" (CONCAT "\box{" label "}" func)))
         (|iht| "} ")))))

defun htMakeButton

— defun htMakeButton —

(defun htMakeButton (&REST G166990 &AUX options func message htCommand)
  (DSETQ (htCommand message func . options) G166990)
  (prog (skipStateInfo? id type)
    (declare (special |$curPage|)))
(return
  (seq (progn
    (setq skipStateInfo? (ifcar options))
    (iht (cons htCommand (cons "{" nil)))
    (bcIssueHt message)
    (cond
      (skipStateInfo?
        (iht (cons "}{}(|htDoneButton| '|"
          (cons func
            (cons "| "
              (cons (htpName |$curPage|
                (cons ")" nil)))))))))
      (t
        (iht (cons "}{}(|htDoneButton| '")
          (cons func
            (cons "| (progn " nil))))))
    (do ((G166977 (htpInputAreaAlist |$curPage|)
      (cdr G166977))
      (G166965 nil))
      ((or (atom G166977)
        (progn (setq G166965 (car G166977)) nil)
        (progn
          (setq id (car G166965))
          (setq type (car (cddddr G166965)))
          G166965)
        nil))
      nil)
    (seq (exit (progn
      (iht (cons "(|htpSetLabelInputString| "
        (cons (htpName |$curPage|)
          (cons "]""
            (cons id
              (cons "]" " nil))))))
      (cond
        ((eq type '|string|)
          (iht (cons "\stringvalue{"
            (cons id
              (cons "]""
                nil))))))
        (t
          (iht (cons "\boxvalue{"
            (cons id
              (cons "]""
                nil))))))
          (iht ""
            nil))))))))
      (iht (cons (htpName |$curPage|)
        (cons "})" nil))))))))))
defun bchtMakeButton

— defun bchtMakeButton —

(defun bchtMakeButton (htCommand message func)
 (prog (id type)
 (declare (special |$curPage|))
 (return
 (SEQ (progn
 (|bcHt| (cons htCommand
 (cons "{"
 (cons message
 (cons "}\{|"htDoneButton| '|'")
 (cons func
 (cons "| (progn "
 nil))))))))
 (DO ((G167004 (|htpInputAreaAlist| |$curPage|)
 (cdr G167004)
 (G166992 nil))
 ((or (atom G167004)
 (progn (setq G166992 (car G167004)) nil)
 (progn
 (setq id (car G166992))
 (setq type (car (cddddr G166992))
 G166992)
 nil))
 nil)
 (SEQ (EXIT (progn
 (|bcHt| (cons "|htpSetLabelInputString| "
 (cons (|htpName| |$curPage|)
 (cons "|" id
 (cons "| " nil))))))
 (cond
 ((eq type '|string|)
 (|bcHt| (cons
 "\"\"\"\"stringvalue{"
 (cons id
 (cons "\"\" nil))))
 (t
 (|bcHt| (cons
 "\"\"\"boxvalue{"
 (cons id
 (cons "\"\" nil))))
 (|bcHt| ") "))))))
defun htProcessDoitButton

— defun htProcessDoitButton —

(defun htProcessDoitButton (G167017)
  (prog (label command func fun)
    (return
     (progn
      (setq label (car G167017))
      (setq command (cadr G167017))
      (setq func (caddr G167017))
      (setq fun (\mkCurryFun\ func (cons command nil)))
      (|iht| "\newline\vspace{1}\centerline{\lispcommand\ DoItBitmap\ (\DoItBitmap\ }"
        fun)
      (|iht| "\vspace{2}\Select \ \UpButton{} \ to go back one page."
      (|iht| "\newline\Select \ \ExitButton{\QuitPage} \ to remove this window."
          )))

——

defun htMakeDoitButton

— defun htMakeDoitButton —

(defun htMakeDoitButton (label command)
  (declare (\special \$curPage))
  (progn
    (cond
      ((equal label "Do It")
       (|bcHt| "\newline\vspace{1}\centerline{\lispcommand{\DoItBitmap}{(|doDoitButton| ")
         (t
          (|bcHt| (cons "\newline\vspace{1}\centerline{\lispcommand{\box{" 
            (cons label
             (cons "})){(|doDoitButton| " 
                nil))}})))
          (|bcHt| (\htpName\ \$curPage))))

——
defun doDoitButton

— defun doDoitButton —

(defun |doDoitButton| (htPage command)
  (declare (ignore htPage))
  (executeInterpreterCommand command))

——

defun executeInterpreterCommand

— defun executeInterpreterCommand —

(defun |executeInterpreterCommand| (command)
  (progn
    (princ command)
    (terpri)
    (setCurrentLine command)
    (catch 'spad_reader (parseAndInterpret command))
    (princ (mkprompt))
    (finish-output)))

——

defun htDoneButton

— defun htDoneButton —

(defun |htDoneButton| (func htPage)
  (cond
defun typeCheckInputAreas

(defun typeCheckInputAreas (htPage)
  (prog (inputAlist stringName t2 t3 t4 t5 t6 t7 spadType t8 filter-condList string val errorCondition)
    (declare (special $bcParseOnly!))
    (return
      (SEQ (progn
        (setq inputAlist nil)
        (setq errorCondition nil)
        (DO ((G167160 (htpInputAreaAlist htPage)
              (cdr G167160))
             (entry nil))
           (or (atom G167160)
               (progn (setq entry (car G167160)) nil))
             nil)
      (SEQ (EXIT (cond
                    ((and (consp entry)
                      (progn
                        (setq stringName (QCAR entry))
                        (setq t2 (QCDR entry))
                        (and (consp t2)
                          (progn
                            (setq t3
                              (QCDR t2))
                            (and (consp t3)
                              (progn
                                (setq t4
                                  (QCDR t3))
                                (and (consp t4)
                                  (progn
                                    (setq t5
                                      (QCDR t4))
                                    (and (consp t5)
                                      (eq (QCAR t5) 
                                        'string)
                                      (progn
                                        (setq t6
                                          (QCDR t5))
                                      ....)))))))))))

-----
(setq t6 (QCDR t5))
(and (consp t6)
(progn
  (setq t7 (QCDR t6))
  (and (consp t7)
    (progn
      (setq spadType (QCAR t7))
      (setq t8 (QCDR t7))
      (and (consp t8)
        (eq (QCDR t8) nil)
        (progn
          (setq filter (QCAR t8))
          t))))))))))))))
(progn
  (setq condList (LASSOC (LASSOC spadType (|htpDomainPvarSubstList| httPage))
    (|htpDomainVariableAlist| httPage)))
  (setq string (|htpLabelFilteredInputString| httPage stringName))
  (cond
    ($bcParseOnly|
      (cond
        ((null (|ncParseFromString| string))
          (|htpSetLabelErrorMsg| httPage "Syntax Error"
            "Syntax Error")
          (t nil)))
      (t
        (setq val
defun checkCondition
  (defun |checkCondition| (s1 string condList)
    (prog (pred t2 t3 pvar t4 pattern val type data newType)
      (return
       (cond
        ((and (consp condList) (eq (QCDR condList) nil)
          (progn
            (setq t2 (QCAR |condList|))
            (and (consp t2)
              (eq (QCAR t2) '|Satisfies|)
              (progn
                (setq t3 (QCDR t2))
                (and (consp t3)
                  (progn
                    (setq pvar (QCAR t3))
                    (setq t4 (QCDR t3))
                    (AND (consp t4)
                      (eq (QCDR t4) nil)
                      (progn
                        (setq pred (QCAR t4))
                        (t))))))))
        (setq val (funcall pred string))
        (cond
          ((stringp val) val)
          (t (cons '([String]) ([wrap| s1]))))
        ((null (and (consp condList) (eq (QCDR condList) nil))
          (progn
            (setq t2 (QCAR condList))))
        )))
    )))

|checkCondition|
(\htpLabelInputString|
  \htPage stringName)
(string condList))

\checkCondition\n\htpLabelInputString\n\htPage stringName\n(string condList))

(iter (cond
  ((stringp val)
   (setq errorCondition t)
   (\htpSetLabelErrorMsg| htPage stringName val))
  (t
   (\htpSetLabelSpadValue| htPage stringName val)))))))

errorCondition))))

---

defun checkCondition

— defun checkCondition —

(defun |checkCondition| (s1 string condList)
  (prog (pred t2 t3 pvar t4 pattern val type data newType)
    (return
     (cond
      ((and (consp condList) (eq (QCDR condList) nil)
        (progn
          (setq t2 (QCAR |condList|))
          (and (consp t2)
            (eq (QCAR t2) '|Satisfies|)
            (progn
              (setq t3 (QCDR t2))
              (and (consp t3)
                (progn
                  (setq pvar (QCAR t3))
                  (setq t4 (QCDR t3))
                  (AND (consp t4)
                    (eq (QCDR t4) nil)
                    (progn
                      (setq pred (QCAR t4))
                      (t)))))))))
      (setq val (funcall pred string))
      (cond
       ((stringp val) val)
       (t (cons '([String]) ([wrap| s1]))))
      ((null (and (consp condList) (eq (QCDR condList) nil))
        (progn
          (setq t2 (QCAR condList))))
      )))
  )))

|checkCondition|
(\htpLabelInputString|
  \htPage stringName)
(string condList))

\checkCondition\n\htpLabelInputString\n\htPage stringName\n(string condList))

(iter (cond
  ((stringp val)
   (setq errorCondition t)
   (\htpSetLabelErrorMsg| htPage stringName val))
  (t
   (\htpSetLabelSpadValue| htPage stringName val)))))))

errorCondition))))

---

defun checkCondition

— defun checkCondition —

(defun |checkCondition| (s1 string condList)
  (prog (pred t2 t3 pvar t4 pattern val type data newType)
    (return
     (cond
      ((and (consp condList) (eq (QCDR condList) nil)
        (progn
          (setq t2 (QCAR |condList|))
          (and (consp t2)
            (eq (QCAR t2) '|Satisfies|)
            (progn
              (setq t3 (QCDR t2))
              (and (consp t3)
                (progn
                  (setq pvar (QCAR t3))
                  (setq t4 (QCDR t2))
                  (AND (consp t4)
                    (eq (QCDR t4) nil)
                    (progn
                      (setq pred (QCAR t4))
                      (t))))))))
      (setq val (funcall pred string))
      (cond
       ((stringp val) val)
       (t (cons '([String]) ([wrap| s1]))))
      ((null (and (consp condList) (eq (QCDR condList) nil))
        (progn
          (setq t2 (QCAR condList))))
      )))
  )))
(and (consp t2)
  (eq (QCAR t2) 'isDomain))
(progn
  (setq t3 (QCDR t2))
  (and (consp t3)
  (progn
    (setq pvar (QCAR t3))
    (setq t4
      (QCDR t3))
    (and (consp t4)
      (eq (QCDR t4) nil)
      (progn
        (setq pattern
          (QCAR t4))
        t)))))))))

(|systemError|
  "currently invalid domain condition")
((equal |pattern| '(|String|))
 (cons '(|String|) (|wrap| s1)))
(t (setq val (|parseAndEval| string))
 (cond
   (stringp val)
   (cond
     ((equal val "Syntax Error ")
      "Error: Syntax Error ")
     (t (|condErrorMsg| pattern))))
(t (setq type (car val))
 (setq data (cdr val))
 (setq newType
   (catch 'spad_reader
     (|resolveTM| type pattern)))
 (cond
   ((null newType) (|condErrorMsg| pattern))
   (t (|coerceInt| val newType))))))))

---

defun condErrorMessage

--- defun condErrorMessage ---

defun |condErrorMessage| (type)
 (prog (typeString)
   (return
    (progn
      (setq typeString (|form2String| type))
      (cond
((consp typeString)
 (setq typeString
     (apply (|function| CONCAT) typeString)))
(CONCAT "Error: Could not make your input into a "
     typeString)))

---

defun parseAndEval

— defun parseAndEval —

(defun |parseAndEval| (string)
 (prog (|$InteractiveMode| $boot $spad |$e| |$QuietCommand|)
     (declare (special |$InteractiveMode| $boot $spad |$e|
             |$QuietCommand|))
   (return
     (progn
       (setq |$InteractiveMode| t)
       (setq $boot nil)
       (setq $spad t)
       (setq |$e| |$InteractiveFrame|)
       (setq |$QuietCommand| t)
       (|parseAndEval1| string))))

---

defun parseAndEval1

— defun parseAndEval1 —

(defun |parseAndEval1| (string)
 (let (v syntaxError pform val)
     (setq syntaxError nil)
     (setq pform
         (progn
           (setq v
               (|applyWithOutputToString| '|ncParseFromString| (cons string nil)))
           (cond
             ((car v) (car v))
             (t (setq syntaxError t) (cdr v)))))
     (cond
       (syntaxError "Syntax Error "))
(setq val
  (|applyWithOutputToString| '|processInteractive|
   (cons pform (list nil))))
(cond
  ((car val) (car val))
  (t "Type Analysis Error")))
(t nil)))

defun oldParseString
  — defun oldParseString —
(defun |oldParseString| (string)
  (prog (tree)
    (return
     (progn
      (setq tree
        (|applyWithOutputToString| '|string2SpadTree|
         (cons string nil)))
      (cond
       ((car tree)
        (|parseTransform| (postTransform (car tree))))
       (t (cdr tree)))))))

defun makeSpadCommand
  — defun makeSpadCommand —
(defun |makeSpadCommand| (&REST G167322 &AUX z)
  (setq z G167322)
  (prog (opForm lastArg argList)
    (return
     (SEQ (progn
       (setq opForm (CONCAT (car z) "\"\""))
       (setq lastArg (|last| z))
       (setq z (cdr z))
       (setq argList nil)
       (DO (((G167306 1 (cdr G167306)) (arg nil))
defun htMakeInputList

    — defun htMakeInputList —

(defun |htMakeInputList| (stringList)
    (prog (lastArg argList)
        (return
         (SEQ (progn
              (setq lastArg (|last| stringList))
            (setq argList nil)
            (DO ((G167328 stringList (cdr G167328)) (arg nil))
                ((or (atom G167328)
                      (progn (setq arg (car G167328)) nil)
                     (null (NEQUAL arg lastArg)))
                  nil)
              (SEQ (EXIT (setq argList
                            (cons
                             (CONCAT arg ", "
                              argList)))))))
            (setq argList (NREVERSE (cons lastArg argList)))
            (CONCAT opForm (apply (|function| CONCAT) argList)
             "))))))))

defun bracketString

    — defun bracketString —

(defun |bracketString| (string)
(CONCAT "[" string "]")

---

defun quoteString

— defun quoteString —

(defun quoteString (string)
  (CONCAT "\" string "\\")

---

defvar $funnyQuote

— initvars —

(defvar $funnyQuote \Rubout)

---

defvar $funnyBacks

— initvars —

(defvar $funnyBacks \200)

---

defun htEscapeString

— defun htEscapeString —

(defun htEscapeString (str)
  (declare (special $funnyBacks $funnyQuote))
  (progn
    (CONCAT "\" string "\\")
  )

---
(setq str (SUBSTITUTE "$funnyQuote" #" str))
(SUBSTITUTE "$funnyBacks" #\ str)))

defun htsv

---
defun htsv
---

(defun htsv ()
  (startHTPage 50)
  (htSetVars)))

---
defun htSetVars

---
defun htSetVars
---

(defun htSetVars ()
  (declare (special $setOptions $lastTree $path))
  (progn
    (setq $path nil)
    (setq $lastTree nil)
    (cond
      ((NEQUAL 0 (LASTATOM $setOptions))
        (htMarkTree $setOptions 0))
      (htShowSetTree $setOptions)))

---
defun htShowSetTree

---
defun htShowSetTree
---

(defun htShowSetTree (setTree)
  (prog (page okList maxWidth1 maxWidth2 tabset1 tabset2 label links)
    (declare (special $path))
    (return
      (SEQ (progn
(setq $path|
    (TAKE (SPADDIFFERENCE (LASTATOM setTree))
      $path|))
(setq page (htInitPage (mkSetTitle) nil))
(htpSetProperty page 'setTree setTree)
(setq links nil)
(setq maxWidth1 (setq maxWidth2 0))
(SEQ (DO ((G167379 setTree (cdr G167379))
        (setData nil))
    ((or (atom G167379)
    (progn
      (setq setData (car G167379))
      nil))
    nil)
(SEQ (EXIT (cond
        ((satisfiesUserLevel| (elt setData 2))
        (EXIT (progn
          (setq okList
            (cons setData okList))
          (setq maxWidth1
            (max
              (#| (PNAME (elt setData 0)))
              maxWidth1))
          (setq maxWidth2
            (max
              (htShowCount| (STRINGIMAGE
                (elt setData 1)))
              maxWidth2)))))
          (setq maxWidth1 (max 9 maxWidth1))
          (setq maxWidth2 (max 41 maxWidth2))
          (setq tabset1 (STRINGIMAGE maxWidth1))
          (setq tabset2 (STRINGIMAGE
            (SPADDIFFERENCE
             (+ maxWidth2 maxWidth1) 1)))
          (htSay| "\tab{2}\newline Variable\tab{"
            (STRINGIMAGE
              (+ maxWidth1
                (quotient maxWidth2 3)))
            "}Description\tab{"
            (STRINGIMAGE
              (+ (+ maxWidth2 maxWidth1) 2))
            "}Value\newline\beginitems ")
        (DO ((G167392 (reverse okList) (CDR G167392))
            (setData nil))
        ((or (atom G167392)
            (progn

(setq setData (car G167392))
(nil)
(SEQ (EXIT (progn
  (|htSay| "\\item")
  (setq label
    (STRCONC "\\menuitemstyle{"
     (elt setData 0)
     "}\\"))
  (setq links
    (cons label
      (cons
        (cons '|text|
          (cons "\\tab{"
            (cons tabset1
              (cons "}"
                (cons
                  (elt setData 1)
                  (cons "\\tab{"
                    (cons tabset2
                      (cons "}{\\em "
                        (cons
                          (|htShowSetTreeValue|
                           setData)
                          (cons "})" nil))))))))
      nil)
    (cons '|htShowSetPage|
      (cons (elt setData 0) nil))))))
( |htMakePage|
  (cons
    (cons '|bcLispLinks|
      (cons links
        (cons '|options|
          (cons '|indent| . 0) nil))))
    nil)))))))
(|htSay| "\\enditems") (|htShowPage|))))))

defun htShowCount

— defun htShowCount —
(defun |htShowCount| (s)
  (prog (m i count)
    (return
     (SEQ (progn
       (setq m (|#| s))
       (cond
        ((> 8 m) (m 1))
        (t (setq i 0) (setq count 0)
         (DO () ((NULL (> (- m 7) i)) nil)
          (SEQ (EXIT (cond
            ((and (equal (elt s i) #\{)
             (equal (elt s (1+ i)) #\})
             (equal (elt s (+ i 2)) #\e)
             (equal (elt s (+ i 3)) #\m))
             (setq i (+ i 6)))
             (t (setq i (1+ i))
             (setq count (1+ count))))))))
          (+ count (- m i))))))))

---

defun htShowSetTreeValue

--- defun htShowSetTreeValue ---

(defun |htShowSetTreeValue| (setData)
  (prog (st)
    (return
     (progn
       (setq st (elt setData 3))
       (cond
        ((eq st 'function)
         (|object2String| (FUNCALL (elt setData 4) '|%display%|)))
        ((eq st 'integer)
         (|object2String| (|eval| (elt setData 4))))
        ((eq st 'string)
         (|object2String| (|eval| (elt setData 4))))
        ((eq st 'literals)
         (|object2String|
          (|translateTrueFalse2YesNo| (|eval| (elt setData 4))))
        (eq st 'tree) "...")
        (t (|systemError|))))))

---
defun mkSetTitle

— defun mkSetTitle —

(defun mkSetTitle ()
  (declare (special $path))
  (STRCONC "Command \emset 
   ([listOfStrings2String] "$path")")
)

defun listOfStrings2String

— defun listOfStrings2String —

(defun listOfStrings2String (u)
  (cond
    ((null u) "")
    (t
     (STRCONC ([listOfStrings2String] (cdr u)) " 
      ([stringize] (car u))
    ))
  ))

defun htShowSetPage

— defun htShowSetPage —

(defun htShowSetPage (htPage branch)
  (prog (setTree setData st)
    (declare (special $path))
    (return
      (progn
        (setq setTree (htpProperty htPage '|setTree|))
        (setq $path|
          (cons branch
            (TAKE (- (LASTATOM setTree)) $path|))
          (setq setData (|assoc| branch setTree))
          (cond
            ((null setData) (|systemError| "No Set Data"))
            (t (setq st (elt setData 3)))
          ))
    ))
  )
(cond
  ((eq st 'function)
   (|htShowFunctionPage| htPage setData))
  ((eq st 'integer)
   (|htShowIntegerPage| htPage setData))
  ((eq st 'literals)
   (|htShowLiteralsPage| htPage setData))
  ((eq st 'tree)
   (|htShowSetTree| (elt setData 5)))
  ((eq st 'string)
   (|htSetNotAvailable| htPage
    "set compiler")
   (t (|systemError| "Unknown data type")))))))

defun htShowLiteralsPage

— defun htShowLiteralsPage —

(defun |htShowLiteralsPage| (htPage setData)
  (|htSetLiterals| htPage (elt setData 0) (elt setData 1)
   (elt setData 4) (elt setData 5) '|htSetLiteral|))

defun htSetLiterals

— defun htSetLiterals —

(defun |htSetLiterals| (htPage name message variable values functionToCall)
  (prog (page links)
    (return
     (SEQ (progn
       (setq page
         (|htInitPage| "Set Command"
          (|htpPropertyList| htPage)))
         (|htSetProperty| page variable variable)
         (|bcHt| (cons \"\centerline{Set {\em \" name
           (cons name
             (cons }\newline nil)))
         (|bcHt| (cons html Description: } "
           (cons message
defun htSetLiteral

— defun htSetLiteral —

(defun |htSetLiteral| (htPage val)
  (progn
    (|htInitPage| "Set Command" nil)
    (set (|htpProperty| htPage '|variable|) (translateYesNo2TrueFalse|val|) (|htKill| htPage val)))

———
(defun htShowIntegerPage

--- defun htShowIntegerPage ---

(defun |htShowIntegerPage| (htPage setData)
  (prog (page message t1)
    (declare (special |$htFinal| |$htInitial|))
    (return
      (progn
        (setq page
          ([htInitPage| (|mkSetTitle|)
            (|htpPropertyList| htPage))
          ([htpSetProperty| page 'variable| (elt setData 4))
          ([bcHt| (cons "\\centerline{Set {\em "
                (cons (elt setData 0)
                  (cons "}\\newline" nil))]))
        (setq message (elt setData 1))
        ([bcHt| (cons "{\em Description: } "
              (cons message
                (cons "\\newline\\vspace{1} "
                  nil))))
        (setq t1 (elt setData 5))
        (setq |$htInitial| (car t1))
        (setq |$htFinal| (cadr t1))
        (cond
          ((equal |$htFinal| (+ |$htInitial| 1))
            ([bcHt| "Enter the integer {\em "]
            ([bcHt| (|stringize| |$htInitial|))
            ([bcHt| "} or {\em "]
            ([bcHt| (|stringize| |$htFinal|))
            ([bcHt| ":")])
          (null |$htFinal|)
            ([bcHt| "Enter an integer greater than {\em "]
            ([bcHt| (|stringize| (- |$htInitial| 1)))
            ([bcHt| ":")])
          (t ([bcHt| "Enter an integer between {\em "]
            ([bcHt| (|stringize| |$htInitial|))
            ([bcHt| "} and {\em "]
            ([bcHt| (|stringize| |$htFinal|))
            ([bcHt| ":")]))
        ([htMakePage| (|domainConditions| (|Satisfies| S chkRange))
          (cons 'bcStrings|)
          (cons (cons 5
            (cons 'eval (elt setData 4))
            (cons 'value (cons 'S nil))))
          nil))
        ([htSetvarDoneButton| "Select to Set Value"])
      )
    ))
  )
)
defun htSetInteger

— defun htSetInteger —

(defun htSetInteger (htPage)
  (prog (val)
    (return
     (progn
       (htInitPage (mkSetTitle) nil)
       (setq val
         (chkRange (htpLabelInputString htPage '|value|)))
       (cond
         ((null (integerp val))
          (errorPage htPage
           (cons "Value Error" nil))
         (t (set (htpProperty htPage '|variable|) val)
          (htKill htPage val))))))

---

defun htShowFunctionPage

— defun htShowFunctionPage —

(defun htShowFunctionPage (htPage setData)
  (prog (fn)
    (return
     (cond
      ((setq fn (elt setData 6)) (funcall fn htPage))
      (t (htpSetProperty htPage '|setData| setData)
       (htpSetProperty htPage '|parts| (elt setData 5))
       (htShowFunctionPageContinued htPage)))))
(defun htShowFunctionPageContinued
  
  ;; defun htShowFunctionPageContinued
  (defun |htShowFunctionPageContinued| (htPage)
    (prog (parts setData phrase kind variable checker
            initValue restParts page currentValue)
      (return
       (progn
         (setq parts (|htpProperty| htPage '|parts|))
         (setq setData (|htpProperty| htPage '|setData|))
         (setq phrase (caar parts))
         (setq kind (cadar parts))
         (setq variable (caddar parts))
         (setq checker (car (cdddar parts)))
         (setq initValue (cdr (cdddar parts)))
         (setq restParts (cdr parts))
         (|htpSetProperty| htPage '|variable| variable)
         (|htpSetProperty| htPage '|checker| checker)
         (|htpSetProperty| htPage '|parts| restParts)
         (cond
           ((eq kind 'literals)
            (|htSetLiterals| htPage (elt setData 0) phrase
                        variable checker '|htFunctionSetLiteral|))
           (t
            (setq page
              (|htInitPage| (|mkSetTitle|)
              (|htpPropertyList| htPage)))
              (|bcHt| (cons "\centerline{Set {\em 
                              (cons (elt setData 0)
                              (cons "}\newline" nil))})
              (|bcHt| (cons "{\em Description: } 
                              (cons (elt setData 1)
                              (cons "\newline\vspace{1} "
                              nil)))))
              (setq currentValue (eval variable))
              (|htMakePage|
              (cons (cons '|domainConditions|
                (cons (cons '|text| phrase)
                  (cons (cons '|inputStrings|
                    (cons "" "
                    nil)))))
            (t))
          ))
        ))
      ))
    ))
  )
defun htSetvarDoneButton

— defun htSetvarDoneButton —

(defun htSetvarDoneButton (message func)
  (progn
    (|bcHt| "\\newline\\vspace{1}\\centerline{"
    (cond
      ((OR (equal message "Select to Set Value")
       (equal message "Select to Set Values"))
       (|bchtMakeButton| "\\lisplink"
           "\\ControlBitmap{clicktoset}" func)
      (t
       (|bchtMakeButton| "\\lisplink"
           (CONCAT "\\fbox{" message "}\"
           func))
    (|bcHt| "} "))))

defun htFunctionSetLiteral

— defun htFunctionSetLiteral —

(defun htFunctionSetLiteral (htPage val)
  (progn
    (|htInitPage| "Set Command" nil)
    (set (|htpProperty| htPage '|variable|)
        (translateYesNo2TrueFalse| val))
    (|htSetFunCommandContinue| htPage val)))
defun htSetFunCommand

— defun htSetFunCommand —

(defun |htSetFunCommand| (htPage)
  (prog (variable checker value)
    (return
      (progn
        (setq variable (|htpProperty| htPage '|variable|))
        (setq checker (|htpProperty| htPage '|checker|))
        (setq value
          (|htCheck| checker
            (|htpLabelInputString| htPage '|value|)))
        (set variable value)
        (|htSetFunCommandContinue| htPage value))))

———

defun htSetFunCommandContinue

— defun htSetFunCommandContinue —

(defun |htSetFunCommandContinue| (htPage value)
  (prog (parts t2 t3 predicate restParts continue)
    (return
      (progn
        (setq parts (|htpProperty| htPage '|parts|))
        (setq continue
          (cond
            ((null parts) nil)
            ((and (consp parts)
              (progn
                (setq t2 (QCAR parts))
                (and (consp t2)
                  (eq (QCAR t2) '|break|))
                (progn
                  (setq t3 (QCDR t2))
                  (AND (consp t3)
                    (eq (QCDR t3) nil)
                    (progn
                      (setq predicate
                        (QCAR t3))
                      t))))))
            (progn
              (setq restParts (QCDR parts))
```
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

defun htKill

— defun htKill —

(defun htKill (htPage value)
  (declare (ignore htPage))
  (prog (string)
    (declare (special $path))
    (return
     (progn
       (htInitPage "System Command" nil)
       (setq string
         (STRCONC "{\em set 
            (listOfStrings2String
              (cons value $path))
         }\})))
       (htMakePage
         (cons '('text|"{Here is the AXIOM system command you could have issued:}
          "\vspace{2}\newline\centerline{\tt"
            (cons (cons 'text|string) nil)))
       (htMakePage
         ('(\text|"})\vspace{1}\newline\rm")))
       (htSay
         "\vspace{2}{Select \ \UpButton{} \ to go back.}"
       (htSay
         "\newline{Select \ \ExitButton{QuitPage} \ to remove this window.}"
       (htProcessDoitButton
         (cons "Press to Remove Page"
           (cons "" (cons 'htDoNothing nil)))
       (htShowPage)))))


defun htSetNotAvailable

— defun htSetNotAvailable —
(defun htSetNotAvailable (htPage whatToType)
  (prog (page string)
    (return
     (progn
       (setq page
         (htInitPage "Unavailable Set Command"
           (htpPropertyList htPage)))
       (htInitPage "Unavailable System Command" nil)
       (setq string
         (STRCONC \{em " whatToType \}
           ")")
       (htMakePage
         (cons '(text \vspace{1}\newline"\vspace{2}\newline\centerline{\tt
           "Sorry, but this system command is not available through HyperDoc. Please directly issue this command in an AXIOM window for more information:}"
           \vspace{1}\newline")
         (cons (cons 'text string) nil)))
       (htMakePage
         '((text \"Press to Remove Page\n           (cons "" (cons 'htDoNothing nil))))
       (htProcessDoitButton
         ((text \"Press to Remove Page\n           (cons "" (cons 'htDoNothing nil))))
       (htShowPage))))

------

defun htDoNothing

    defun htDoNothing —

(defun htDoNothing (htPage command)
  (declare (ignore htPage command))
  nil)

------

defun htCheck

    defun htCheck —

(defun htCheck (checker value)
  (cond
    ((consp checker) (htCheckList checker (parseWord value)))
    (t (funcall checker value))))

------
defun parseWord

— defun parseWord —

(defun |parseWord| (x)
  (prog ()
    (return
      (SEQ (cond
        ((stringp x)
          (cond
            ((prog (G167588)
              (setq G167588 t)
              (return
                (DO ((G167594 nil (null G167588))
                    (G167595 (maxindex x))
                    (i 0 (1+ i)))
                ((OR G167594 (QSGREATERP i G167595))
                  G167588)
                (SEQ (EXIT (setq G167588
                              (AND G167588
                                   (digitp (elt x i))))))))
                (parse-integer x))
              (t (intern x)))))
          (t x))))))

defun htCheckList

— defun htCheckList —

(defun |htCheckList| (checker value)
  (prog (n t2 m)
    (return
      (progn
        (cond
          (((|member| value '|y| |ye| |yes| Y YE YES))
            (setq value '|yes|))
          (((|member| value '|n| |no| N NO)) (setq value '|no|))
          (cond
            ((and (consp checker)
              (progn
                (setq n (qcar checker))
                (setq t2 (qcdr checker))))

            )))))))
(and (consp t2) (eq (QCDR t2) nil)
  (progn (setq m (QCAR t2)) t)))
(integerp n))
(cond
  ((eql m (1+ n))
   (cond ((|member| value checker) value) (t n)))
  ((null m)
   (cond
    ((and (integerp value) (>= value n)) value)
    (t n)))
  ((integerp m)
   (cond
    ((and (integerp value) (>= value n)
        (<= value m))
      value)
    (t n))))))
  ((|member| value checker) value)
  (t (car checker))))))))

---

(defun translateYesNoToTrueFalse
  (cond
    ((eq x '|yes|) t)
    ((eq x '|no|) nil)
    (t x)))

---

(defun chkNameList
  (defun |chkNameList| (x)
    (prog (u parsedNames)
      (return
       (SEQ (progn
         (setq u (|bcString2ListWords| x))
         (setq parsedNames

---
(prog (G167635)
  (setq G167635 nil)
  (return
    (DO ((G167640 u (CDR G167640))
          (x nil))
      ((or (atom G167640)
            (progn
              (setq x (car G167640))
              nil))
       (NREVERSE0 G167635))
    (SEQ (EXIT (setq G167635
                  (cons (|ncParseFromString| x)
                        G167635))))))

(cond
  ((prog (G167646)
         (setq G167646 t)
         (return
          (DO ((G167652 nil (NULL G167646))
               (G167653 parsedNames (CDR G167653))
               (x nil))
              ((OR G167652 (ATOM G167653)
                  (progn (setq x (car G167653)) nil))
               G167646)
              (SEQ (EXIT (setq G167646
                           (AND G167646 (identp x))))))))
    parsedNames)
  (t
   "Please enter a list of identifiers separated by blanks"))))))

defun chkPosInteger

  — defun chkPosInteger —

(defun |chkPosInteger| (s)
  (prog (u)
    (return
      (cond
        ((and (setq u (|parseOnly| s)) (integerp u) (> u 0))
         u)
        (t "Please enter a positive integer"))))

  ———
defun chkOutputFileName

      — defun chkOutputFileName —

(defun |chkOutputFileName| (s)
  (cond
   ((|member| (|bcString2WordList| s) '(CONSOLE |console|))
    '|console|)
   (t (|chkDirectory| s))))

----------

defun chkDirectory

      — defun chkDirectory —

(defun |chkDirectory| (s) s)

----------

defun chkNonNegativeInteger

      — defun chkNonNegativeInteger —

(defun |chkNonNegativeInteger| (s)
  (prog (u)
    (return
     (cond
      ((and (setq u (|ncParseFromString| s)) (integerp u) (>= u 0))
       u)
     (t "Please enter a non-negative integer"))))

----------

defun chkRange

      — defun chkRange —
(defun |chkRange| (s)
  (prog (u)
    (declare (special |$htFinal| |$htInitial|))
    (return
      (cond
        ((and (setq u (|ncParseFromString| s)) (integerp u)
          (>= u |$htInitial|)
          (or (null |$htFinal|) (<= u |$htFinal|)))
         u)
        ((null |$htFinal|)
         (STRCONC "Please enter an integer greater than "
          (|stringize| (- |$htInitial| 1))))
        (t
         (STRCONC "Please enter an integer between "
          (|stringize| |$htInitial|) " and "
          (|stringize| |$htFinal|)))))))

defun chkAllNonNegativeInteger

defun chkAllNonNegativeInteger| (s)
  (prog (u)
    (return
      (or (and (setq u (|ncParseFromString| s))
        (|member| u '(|a| |al| |all| A AL ALL)) 'ALL)
        (|chkNonNegativeInteger| s)
        "Please enter {\em all} or a non-negative integer")))

defun htMakePathKey,fn

defun htMakePathKey,fn| (a b)
  (SEQ (if (null b) (EXIT a))
    (EXIT (|htMakePathKey,fn| (STRCONC a "." (PNAME (car b)))
            (cdr b))))
defun htMakePathKey

— defun htMakePathKey —

(defun htMakePathKey (path)
  (cond
   ((null path) (systemError "path is not set"))
   (t
    (intern (htMakePathKey fn (PROPNAME (car path)) (cdr path))))))

---

defun htMarkTree

— defun htMarkTree —

(defun htMarkTree (tree n)
  (SEQ (progn
         (rplacd (LASTTAIL tree) n)
         (SEQ (DO ((G167706 tree (cdr G167706)) (branch nil))
              ((OR (ATOM G167706)
                   (progn (setq branch (car G167706)) nil))
               nil)
         (SEQ (EXIT (cond
                      ((eq (elt branch 3) 'tree)
                       (EXIT (htMarkTree (elt branch 5) (1+ n))))))))))

---

defun htSetHistory

— defun htSetHistory —

(defun htSetHistory (htPage)
  (prog (msg data)
    (return
      (progn
        (setq msg
          'when the history facility is on (yes), results of computations are saved in memory)
        (setq data
          (cons 'history
            ()))])

')
defun htSetOutputLibrary

--- defun htSetOutputLibrary ---

(defun |htSetOutputLibrary| (htPage)
  (|htSetNotAvailable| htPage ")set compiler output"))

---

defun htSetInputLibrary

--- defun htSetInputLibrary ---

(defun |htSetInputLibrary| (htPage)
  (|htSetNotAvailable| htPage ")set compiler input"))

---

defun htSetExpose

--- defun htSetExpose ---

(defun |htSetExpose| (htPage)
  (|htSetNotAvailable| htPage ")set expose"))

---
defun htSetOutputCharacters

— defun htSetOutputCharacters —

(defun |htSetOutputCharacters| (htPage)
  (|htSetNotAvailable| htPage "set output characters")
)

defun htSetLinkerArgs

— defun htSetLinkerArgs —

(defun |htSetLinkerArgs| (htPage)
  (|htSetNotAvailable| htPage "set fortran calling linker")
)

defun htSetCache

— defun htSetCache —

(defun |htSetCache| (&REST G167749 &AUX options htPage)
  (declare (special |$valueList| |$path|))
  (setq htPage (car G167749))
  (setq options (cdr G167749))
  (progn
    (setq |$path| '(|functions| |cache|))
    (setq htPage (|htInitPage| (|mkSetTitle|) nil))
    (setq |$valueList| nil)
    (|htMakePage|
      '(|text| "Use this system command to cause the AXIOM interpreter to 'remember' "
        "past values of interpreter functions. "
        "To remember a past value of a function, the interpreter "
        "sets up a {\em cache} for that function based on argument values. "
        "When a value is cached for a given argument value, its value is gotten "
        "from the cache and not recomputed. Caching can often save much "
        "computing time, particularly with recursive functions or functions that "
        "are expensive to compute and that are called repeatedly "
        "with the same argument." "{vspace}\|\newline" )
      (|domainConditions|
        (|Satisfies| S chkNameList)))
  )
"Enter below a list of interpreter functions you would like specially cached."
"Use the name \textit{all} to give a default setting for all interpreter functions."
\vspace{1}\newline
"Enter \textit{all} or a list of names (separate names by blanks):"

\begin{verbatim}
(inputStrings ("" "" 60 "all" names S))
(doneButton "Push to enter names" [htCacheAddChoice]))
\end{verbatim}

---

\textbf{defun} htCacheAddChoice

\textbf{— defun} htCacheAddChoice —

\begin{verbatim}
(defun htCacheAddChoice (htPage)
  (prog (names page)
    (declare (special \$valueList))
    (return
     (SEQ (progn
       (setq names
         (|bcString2WordList| (|htpLabelInputString| htPage '|names|)))
       (setq \$valueList
         (cons (|listOfStrings2String| names) \$valueList))
       (cond
         ((null names) (|htCacheAddQuery|))
         ((null (cdr names)) (|htCacheOne| names))
         (t (setq page (|htInitPage| (|mkSetTitle|) nil))
           (|htpSetProperty| page '|names| names)
           (|htMakePage|
             '(((domainConditions| (|Satisfies| ALLPI chkAllPositiveInteger))
               (|text|)
               "For each function, enter below a {\em cache length}, a positive integer."
               "This number tells how many past values will be cached."
               "A cache length of {\em 0} means the function won't be cached."
               "To cache all past values, enter {\em all}."
               "\vspace{1}\newline"
               "For each function name, enter {\em all} or a positive integer:"
             ))
             (DO ((i 1 (QSADD1 i))
                   (G167755 names (CDR G167755)) (name nil))
               ((or (atom G167755)
                 (progn (setq name (car G167755)) nil)))
           ))
         )))
\end{verbatim}
defun htMakeLabel

— defun htMakeLabel —

(defun |htMakeLabel| (prefix i)
  (intern (strconc prefix (|stringize| i))))

— defun htCacheSet —

(defun |htCacheSet| (htPage)
  (prog (names num n name val)
    (declare (special |$cacheCount| |$cacheAlist|))
    (return
     (SEQ (progn
       (setq names (|htpProperty| htPage '|names|))
       (DO ((i 1 (QSADD1 i))
            (G167785 names (CDR G167785)) (name nil))))
(or (atom G167785)
  (progn (setq name (car G167785)) nil))
nil)
(SEQ (EXIT (progn
  (setq num
    (|chkAllNonNegativeInteger|
    (htpLabelInputString| htPage
      ([htMakeLabel| "c"
       i))))))
  (setq |$cacheAlist|
    (ADDASSOC (intern name) num
      |$cacheAlist|))))
(cond
  ((setq n (LASSOC '|all| |$cacheAlist|))
   (setq |$cacheCount| n)
   (setq |$cacheAlist|
     ([deleteAssoc| '|all| |$cacheAlist|])))
  (htInitPage| "Cache Summary" nil)
  (bcHt| "In general, interpreter functions ")
  (bcHt| (cond
    ((EQL |$cacheCount| 0)
      '|will {\em not} be cached.|)
    (t (bcHt| "cache ")
      (htAllOrNum| |$cacheCount|)
      "} values.\)\)))
  (bcHt| \vspace{1}\newline )
  (cond
    (|$cacheAlist|
     (DO ((G167801 |$cacheAlist| (cdr G167801))
        (G167774 nil))
      ((or (atom G167801)
        (progn
          (setq G167774 (car G167801))
          nil)
        (progn
          (setq name (car G167774))
          (setq val (CDR G167774))
          (G167774)
          nil))
      nil)
    (SEQ (EXIT (cond
      ((NEQUAL val |$cacheCount|)
       (progn
         ([bcHt| \\newline function \{\em Name\} will cache "]
          (htAllOrNum| val)
          (bcHt| "} values\)\))))))
    (htProcessDoitButton|)
(cons "Press to Remove Page"
  (cons "" (cons 'htDoNothing nil)))
'(htShowPage)))

---

defun htAllOrNum

— defun htAllOrNum —

(defun htAllOrNum (val)
  (bcHt (cond
    ((eq val 'all) "\em all")
    ((eql val 0) "\em no")
    (t
     (STRCONC "the last \em 
               ([stringize val])))
  )))

---

defun htCacheOne

— defun htCacheOne —

(defun htCacheOne (names)
  (prog (page)
    (return
     (progn
       (setq page (htInitPage (mkSetTitle) nil))
       (htsetProperty page 'names names)
       (htMakePage
        '(domainConditions
          (Satisfies ALLPI [chkAllPositiveInteger]))
        (text) "Enter below a \em cache length, a positive integer. "
        "This number tells how many past values will 
        be cached. To cache all past values, 
        enter \em all."
        "\vspace{1}\newline 
        (inputStrings)
        "Enter \em all or a positive integer:" " 5 10 
        [c1] [ALLPI])))
       (htSetvarDoneButton "Select to Set Value"
                         'htCacheSet)
       (htShowPage)))))))
defvar $historyDisplayWidth

— initvars —
(defvar $historyDisplayWidth 120)

defvar $newline

— initvars —
(defvar $newline #\Newline)

defun downlink

— defun downlink —
(defun downlink (page)
  (declare (special $saturn))
  (cond
   ($saturn (downlinkSaturn page))
   (t (htInitPage "Bridge" nil)
      (htSay "\replacepage{" page "}")
      (htShowPage)))))

defun downlinkSaturn

— defun downlinkSaturn —
(defun downlinkSaturn (fn)
  (prog (line u n lines)
(return
  (SEQ (progn
    (setq u (|dbReadLines| fn))
    (setq lines "")
    (DO ()
      (if (null (and (consp u)
            (progn
              (setq line (QCAR u))
              (setq u (QCDR u))
            t)))
        nil)
    (SEQ (EXIT (progn
      (setq n (MAXINDEX line))
      (cond
        ((> 1 n) nil)
        ((equal (elt line 0) #\%)
          nil)
        (t
          (setq lines
            (STRCONC lines line))))))))
  (|issueHTSaturn| lines))))

---

defun dbNonEmptyPattern

— defun dbNonEmptyPattern —

(defun |dbNonEmptyPattern| (pattern)
  (cond
    ((null pattern) "*")
    (t (setq pattern (STRINGIMAGE pattern))
      (cond ((> (|#| pattern) 0) pattern) (t "*"))))

---

defun htSystemVariables,gn

— defun htSystemVariables,gn —

(defun |htSystemVariables,gn| (t1 al)
  (prog (class key options)
    (declare (special |$heading| |$levels|)))
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

\begin{verbatim}
(defun htSystemVariables,fn (t1 al firstTime)
  (declare (special $heading$))
  (SEQ (if (atom t1) (EXIT al))
    (if firstTime (setq $heading$ (\text{opOf} (car t1))) nil)
    (EXIT ([htSystemVariables,fn| options al nil]))
    (if (eq key 'function)
      (EXIT (cons (cons $heading| t1) al)))
    (EXIT (|systemError| key))))

(defun htSystemVariables,displayOptions
  (name class variable val options)
  (SEQ (if (eq class 'integer)
    (EXIT (SEQ (\text{htMakePage} (cons (cons 'bcLispLinks (cons (cons
  
\end{verbatim}

— defun htSystemVariables,fn —

(defun \text{htSystemVariables,fn} (t1 al firstTime)
  (declare (special $heading$))
  (SEQ (if (atom t1) (EXIT al))
    (if firstTime (setq $heading$ (\text{opOf} (car t1))) nil)
    (EXIT ([htSystemVariables,fn| options al nil]))
    (if (eq key 'function)
      (EXIT (cons (cons $heading| t1) al)))
    (EXIT (|systemError| key))))

— defun htSystemVariables,displayOptions —

(defun \text{htSystemVariables,displayOptions} (name class variable val options)
  (SEQ (if (eq class 'integer)
    (EXIT (SEQ (\text{htMakePage}
      (cons (cons 'bcLispLinks
        (cons (cons
          (cons
          (cons


(cons 'text
(cons (elt options 0)
 (cons "="
 (cons
 (or (elt options 1)
  "=")
 nil)))))
nil)
(cons "=
(cons
'|htSetSystemVariableKind|
(cons
 (cons variable
  (cons name
   (cons 'parse-integer nil))))
 nil)))))
nil))
(|htMakePage|
 '(((|domainConditions|
  ((isDomain) INT (|Integer|))))))
(EXIT (|htMakePage|
 (cons (cons '|bcStrings|
  (cons
  (cons 5
   (cons (STRINGIMAGE val)
    (cons name (cons 'INT nil))))
  nil))
 nil)))))))
(if (eq class 'string)
 (EXIT (|htSay| "{"\em " val
 "}\space{1}{")"}))
(EXIT (DO ((G167913 options (cdr G167913)) (x nil))
 ((or (atom G167913)
   (progn (setq x (car G167913)) nil))
 nil)
 (SEQ (if (or (or (equal val x)
   (and (eq val t)
    (eq x '|on|)))
  (and (null val) (eq x '|off|)))
  (EXIT (|htSay| "{"\em " x
 "}\space{1}{")"}))
 (EXIT (|htMakePage|
 (cons (cons 'bcLispLinks|
  (cons
  (cons x
   (cons "=
    (cons '|htSetSystemVariable|
     (cons
      (cons variable
       )
    nil))))
 nil)))))
 nil)))
 (EXIT (|htSay| "{"\em " val
 "}\space{1}{")"}))
(EXIT (DO ((G167913 options (cdr G167913)) (x nil))
 ((or (atom G167913)
   (progn (setq x (car G167913)) nil))
 nil)
 (SEQ (if (or (or (equal val x)
   (and (eq val t)
    (eq x '|on|)))
  (and (null val) (eq x '|off|)))
  (EXIT (|htSay| "{"\em " x
 "}\space{1}{")"}))
 (EXIT (|htMakePage|
 (cons (cons 'bcLispLinks|
  (cons
  (cons x
   (cons "=
    (cons '|htSetSystemVariable|
     (cons
      (cons variable
       )
    nil))))
 nil)))))
 nil)))
 (EXIT (|htSay| "{"\em " val
 "}\space{1}{")"}))
(EXIT (DO ((G167913 options (cdr G167913)) (x nil))
 ((or (atom G167913)
   (progn (setq x (car G167913)) nil))
 nil)
 (SEQ (if (or (or (equal val x)
   (and (eq val t)
    (eq x '|on|)))
  (and (null val) (eq x '|off|)))
  (EXIT (|htSay| "{"\em " x
 "}\space{1}{")"}))
 (EXIT (|htMakePage|
 (cons (cons 'bcLispLinks|
  (cons
  (cons x
   (cons "=
    (cons '|htSetSystemVariable|
     (cons
      (cons variable
       )
    nil))))
 nil)))))
 nil)));
defun htSystemVariables, functionTail

— defun htSystemVariables, functionTail —

(defun \htSystemVariables,\functionTail\ |name class var valuesOrFunction|) (prog \(val\)
  (return
    (SEQ (setq \val\ (\eval\ var))
      (if (atom valuesOrFunction)
        (EXIT (SEQ (\htMakePage\)
          '(((domainConditions|)
            (isDomain\ \STR\ (String|))))
          (\htMakePage\)
            (cons \(cons \ (\bcLinks|)
              (cons \ (cons "reset"
                (cons ""
                  (cons \ (\htSetSystemVariableKind|)
                    \(cons\)
                      \(cons\ var
                        \(cons\ name\ \(cons\ nil\ nil))
                      \nil))
                    \nil))
                \nil))
            \(EXIT\ \(\htMakePage\)
              (cons \ (\bcStrings|)
                \(cons\)
                  \(cons\ 30
                    \(cons\ (STRINGIMAGE\ \val)
                      \(cons\ name\ \(cons\ valuesOrFunction\ nil))
                    \nil)
                  \nil))
              (EXIT \(\htSystemVariables,\displayOptions\ \name\ class\ var\ val\ valuesOrFunction)))))))
defun htSystemVariables

— defun htSystemVariables —

(defun htSystemVariables ()
  (prog (levels heading classlevel table name
            message key variable options func lastHeading
            t1 msg class var valuesOrFunction val)
    (DECLARE (SPECIAL levels heading setOptions UserLevel
                     fullScreenSysVars))
    (return
      (seq (cond
        ((null fullScreenSysVars) (htSetVars))
        (t (setq classlevel UserLevel)
            (setq levels '(compiler development interpreter))
            (setq heading nil)
            (do () ((null (nequal classlevel (car levels))) nil)
                (seq (exit (setq levels (cdr levels))))
            (setq table
                  (nreverse (htSystemVariables fn setOptions nil
                                      t)))
        (htInitPage "System Variables" nil)
        (htSay "\beginmenu")
        (setq lastHeading nil)
        (do ((G167961 table (cdr G167961)) (G167879 nil))
            ((or (atom G167961)
                 (progn (setq G167879 (car G167961)) nil)
                 (progn
                   (progn
                     (setq heading (car G167879))
                     (setq name (cadr G167879))
                     (setq message (caddr G167879))
                     (setq key (car (cddddr G167879)))
                     (setq variable (cadr (cddddr G167879)))
                     (setq options (caddr (cddddr G167879)))
                     (setq func (cadddr (cddddr G167879)))
                     G167879)
                   nil))
            (seq (exit (progn
                        (htSay "\newline\item ")
                        (cond
                          ((equal heading lastHeading)
                           (htSay "\"tab\{8\}")
                          (t
                           (htSay heading
                              "\"tab\{8\}")
                           (setq lastHeading heading))))))}}
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

(htSay "\em " name
 '})\tab{22}| message)
(htSay "\tab{80}"
 (cond
  ((eq key 'function)
   (cond
    ((null options)
     (|htMakePage|
      (cons
       (cons '|bcLinks|
       (cons
        (cons "reset"
        (cons ""
         (cons func (cons nil nil)))
       nil))
      nil)))
    (t
     (setq t1 (car options))
     (setq msg (car t1))
     (setq class (cadr t1))
     (setq var (caddr t1))
     (setq valuesOrFunction (cadddr t1))
     (|htSystemVariables,functionTail|
      name class var valuesOrFunction)
     (DO
      ((G167971 (cdr options)
       (cdr G167971))
       (option nil))
      ((or (atom G167971)
       (progn
        (setq option (car G167971))
        (nil))
       nil)
       (SEQ
        (EXIT
         (cond
          ((and (consp option)
            (eq (QCAR option)
            '|break|))
           '|skip|)
          (t
           (setq msg (car option))
           (setq class (cadr option))
           (setq var (caddr option))
           (setq valuesOrFunction
            (cadddr option))
           (|htSay| "\\newline\\tab{22}"
           msg
           "\\tab{80}"
           (|htSystemVariables,functionTail|)}"}
defun htSetSystemVariableKind

— defun htSetSystemVariableKind —

(defun htSetSystemVariableKind (htPage G168009)
  (prog (variable name fun value)
    (return
      (progn
        (setq variable (car G168009))
        (setq name (cadr G168009))
        (setq fun (caddr G168009))
        (setq value (htpLabelInputString htPage name))
        (cond
          ((and (stringp value) fun)
           (setq value (funcall fun value)))
          (set variable value)
          (htSystemVariables))))))

——

defun htSetSystemVariable

— defun htSetSystemVariable —

(defun htSetSystemVariable (htPage G168030)
  (declare (ignore htPage))
  (prog (name value)
    (return
      (progn
        (setq name (car G168030))
        (setq value (cadr G168030))
        (setq value
          (cond
            ((eq value '|on|) t)
((eq value 'off) nil)
(t value))
(set name value)
(htSystemVariables))))))

defun htGloss

— defun htGloss —

(defun htGloss (pattern)
  (htGlossPage nil
    (or (dbNonEmptyPattern pattern) "*" t))

— defun htGlossPage —

(defun htGlossPage (htPage pattern tryAgain?)
  (prog ($wildCard $key) filter grepForm results defstream
    lines heading k tick)
  (declare (special $wildCard $key $tick))
  (return
    (SEQ (progn
      (setq $wildCard #\*)
      (cond
        ((equal pattern "*")
         (downlink 'GlossaryPage)))
        (t (setq filter (pmTransFilter pattern))
           (setq grepForm (mkGrepPattern filter 'none))
           (setq $key 'none)
           (setq results (applyGrep grepForm 'gloss))
           (setq defstream
             (make-instream
              (STRCONC (getEnv "AXIOM"
                          "/algebra/glossdef.text")))))
    (setq lines
      (gatherGlossLines results defstream))
    (setq heading
      (cond
((equal pattern "")
 "Glossary")
((null lines)
 (cons "No glossary items match {\em "
    (cons pattern
     (cons "}" nil))))
(t
 (cons "Glossary items matching {\em "
    (cons pattern
     (cons "}" nil))))
)
(cond
 ((null lines)
 (cond
   ((and tryAgain? (> (|#| pattern) 0))
    (cond
      ((equal
       (elt pattern
        (setq k (MAXINDEX pattern)))
       #\s)
       (|htGlossPage| htPage
        (SUBSTRING pattern 0 k) t))
      ((upper-case-p (elt pattern 0))
       (|htGlossPage| htPage (downcase pattern) nil))
      t
       (|errorPage| htPage
        (cons "Sorry"
         (cons nil
          (cons
           (cons \\centerline{
              (append heading
               (cons "}" nil)))
           nil))))))))
(t
 (|errorPage| htPage
 (cons "Sorry"
  (cons nil
   (cons \\centerline{
      (append heading
       (cons "}" nil)))
   nil))))
 (t
  (|htInitPageNoScroll| nil heading)
  (|htSay| "\\beginscroll\\beginmenu")
  (DO ((G168058 lines (cdr G168058))
   (line nil))
    ((or (atom G168058)
     (progn (setq line (car G168058)) nil))
     nil)
defun gatherGlossLines

— defun gatherGlossLines —

(defun |gatherGlossLines| (results defstream)
  (prog (n keyAndTick byteAddress line k pointer def x
          j nextPointer xtralines acc)
        (declare (special |$tick|))
        (return
         (SEQ (progn
                (setq acc nil)
                (DO ((G168098 results (cdr G168098))
                  (declare (special |$tick|))
                  (return
                   (SEQ (progn
                          (setq tick
                                (= |charPosition| |$tick| line 1))
                          (|htSay| "\item{\em \menuitemstyle{}}\tab{0}{\em "
                             (|escapeString| (SUBSTRING line 0 tick))
                             "} 
                             (SUBSTRING line (1+ tick) nil)))))))))
        (|htSay| "\endmenu ")
        (|htSay| "\endscroll\newline ")
        (|htMakePage|
         (cons (cons '|bcLinks|
                (cons (cons "Search"
                       (cons ""
                        (cons '|htGlossSearch| (cons nil nil))))
                  nil))
         nil)
        (|htSay| " for glossary entry matching ")
        (|htMakePage|
         (cons (cons '|bcStrings|
                (cons (cons 24
                       (cons "*
                        (cons '|filter| (cons 'em nil))))
                  nil))
         nil))
        (|htShowPageNoScroll|)))))))))))
(keyline nil)
((or (atom G168098)
  (progn (setq keyline (car G168098)) nil))
 nil)
(SEQ (EXIT (progn
  (setq n
    (|charPosition| |$tick| keyline 0))
  (setq keyAndTick
    (SUBSTRING keyline 0 (1+ n)))
  (setq byteAddress
    ([string2Integer]
      (SUBSTRING keyline (1+ n) nil)))))
(file-position defstream byteAddress)
(setq line (readline defstream))
(setq k
  (|charPosition| |$tick| line 1))
(setq pointer
  (SUBSTRING line 0 k))
(setq def
  (SUBSTRING line (1+ k) nil))
(setq xtralines nil)
(DO ()
  ((null (and (null (eofp defstream))
    (setq x
      (readline defstream))
    (setq j
      (|charPosition| |$tick| x 1))
    (setq nextPointer
      (SUBSTRING x 0 j))
    (equal nextPointer
      pointer)))
 nil)
(SEQ (EXIT
  (setq xtralines
    (cons
      (SUBSTRING x (1+ j) nil)
      xtralines)))))
(setq acc
  (cons
    (STRCONC keyAndTick def
      (prog (G168110)
        (setq G168110 
          (return
            (DO
              ((G168115
                (NREVERSE xtralines)
              )
            )
          )
        ))
      acc
    ))))
defun htGlossSearch

— defun htGlossSearch —

(defun |htGlossSearch| (htPage junk)
  (declare (ignore junk))
  (htGloss (htpLabelInputString htPage '|filter|)))

defun htGreekSearch

— defun htGreekSearch —

(defun |htGreekSearch| (filter)
  (prog (ss s names matches nonmatches)
    (return
      (SEQ (progn
        (setq ss (dbNonEmptyPattern| filter|))
        (setq s (pmTransFilter| ss|))
        (cond
          ((and (consp s) (eq (QCAR s) '|error|))
            (bcErrorPage| s|))
          ((null s)
            (errorPage| nil|))
          (acc))))))))

(reverse acc))))))

(CDR G168115))
(G168081 nil))
((OR (ATOM G168115)
  (progn
    (setq G168081
      (car G168115))
    nil))
  G168110)
  (SEQ
    (EXIT
      (setq G168110
        (STRCONC G168110
          G168081)))))
  acc))))))))

——

——
(cons (cons "Missing search string" nil)
(cons nil
(cons "\vspace{2}\centerline{To select one of the greek letters:}\newline "
(cons "\centerline{First enter a search key into the input area}\newline "
(cons "\centerline{Then move the mouse cursor to the work \{\em search\} and click}" nil)))))
(t (setq filter (|patternCheck| s))
(setq names
'(\alpha \beta \gamma \delta \epsilon \zeta \eta \theta \iota \kappa \lambda \mu \nu \pi))
(DO ((G168149 names (CDR G168149)) (x nil))
((or (atom G168149)
(progn (setq x (car G168149)) nil)) nil)
(SEQ (EXIT (cond
((|superMatch?| filter (PNAME x))
(setq matches
(cons x matches)))
(t
(setq nonmatches
(cons x nonmatches))))))
(setq matches (NREVERSE matches))
(setq nonmatches (NREVERSE nonmatches))
(|htInitPage| "Greek Names" nil)
(cond
((null matches)
(|htInitPage|
(cons "Greek names matching search string \{\em "
(cons ss (cons ")" nil))
nil)
(|htSay| \"\vspace{2}\centerline{Sorry, but no greek letters match your search string}\centerline{Click on the up-arrow to try again}\"
(|htShowPage|))
(t
(|htInitPage|
(cons "Greek letters matching search string \{\em "
(cons ss (cons ")" nil))
nil)
(cond
((null nonmatches)
(|htInitPage|
(cons "The greek letters that \{\em match\} your search string \{\em "
ss "};:\")
(t
defun htTextSearch

— defun htTextSearch —

(defun |htTextSearch| (filter)
  (prog (s lines matches nonmatches)
    (return
      (SEQ (progn
          (setq s
            (|pmTransFilter| (|dbNonEmptyPattern| filter)))
          (cond
            ((and (consp s) (eq (QCAR s) '|error|))
             (|bcErrorPage| s))
            ((null s)
             (|errorPage| nil)
             (cons (cons "Missing search string" s))
             nil)
To select one of the lines of text: 
First, enter a search key into the input area, then move the mouse cursor to the work "search" and click.

(t (setq filter s)
 (setq lines
 "{{\em Fruit flies} *like* a {{\em banana and califlower ears.}}"
 (cons
 "{{\em Sneak Sears Silas with Savings Snatch}}"
 nil)))
(DO ((G168191 lines (cdr G168191)) (x nil))
 (or (atom G168191)
 (progn (setq x (car G168191)) nil)
 nil)
 (SEQ (EXIT (cond
 (|superMatch?| filter x)
 (setq matches
 (cons x matches)))
 (t
 (setq nonmatches
 (cons x nonmatches)))))))
(setq matches (NREVERSE matches))
(setq nonmatches (NREVERSE nonmatches))
(|htInitPage| "Text Matches" nil)
(cond
 (null matches)
 (|htInitPage|
 (cons "Lines matching search string {{\em "
 (cons s (cons "}" nil)))
 (nil)
 (|htSay|
 '\|vspace{2}\|centerline{Sorry, but no lines match your search string}\|centerline{{\em | s
 '}}\|centerline{Click on the up-arrow to try again}}\)
 (|htShowPage|))
(t
 (|htInitPage|
 (cons "Lines matching search string {{\em "
 (cons s (cons "}" nil)))
 nil)
 (cond
 (nonmatches
 (|htSay| "The lines that {{\em match} your search string {{\em "
 s ":"})}))
 (t
CHAPTER 74. HYPERDOC BASIC COMMAND SUPPORT

(defun htTutorialSearch
defun htTutorialSearch (pattern)
  (prog (s source target lines t1 name title)
    (return
      (SEQ (progn
        (setq s
          (or (|dbNonEmptyPattern| pattern)
            (return
              (|errorPage| nil
                (cons "Empty search key"
                  (cons nil
                    (cons "\vspace{3}\centerline{You must enter some search string" nil)))))))))
        (setq s (|mkUnixPattern| s)))))
defun mkUnixPattern

— defun mkUnixPattern —

(defun mkUnixPattern (s)
  (prog (starPositions k u)
    (declare (special |$wild|))
    (return
      (SEQ (progn
        (setq u (|mkUpDownPattern| s))
        (setq starPositions
          (reverse (prog (G168264)
            (setq G168264 nil)
            (return
              (DO
                ((G168270 (+ (- 1)
                   (MAXINDEX u)))
                (i 1 (QSADD1 i)))))))
      (mkUnixPattern))))))
((QSGREATERP i G168270)
 (NREVERSE0 G168264))
 (SEQ
  (EXIT
   (cond
    ((equal (elt u i) |$wild|)
     (setq G168264 (cons i G168264))))))))

(DO ((G168277 starPositions (cdr G168277))
   (i nil))
   ((or (atom G168277)
    (progn (setq i (car G168277)) nil)) nil)
   (SEQ (EXIT (setq u
     (STRCONC (SUBSTRING u 0 i)
      ".*" (SUBSTRING u (1+ i) nil)))))))

(cond
  ((NEQUAL (elt u 0) |$wild|)
   (setq u (STRCONC "[^a-zA-Z]" u)))
  (t (setq u (SUBSTRING u 1 nil))))

(cond
  ((NEQUAL (elt u (setq k (MAXINDEX u))) |$wild|)
   (setq u (STRCONC u "[^a-zA-Z]")))
  (t (setq u (SUBSTRING u 0 k)))
  u))))))

|---
Chapter 75

Browser Support Code

75.1 Pages Initiated from HyperDoc Pages

defun conPage

[form2HtString p??]
[downcase p??]
[lassq p??]
[downlink p1324]
[conPageFastPath p1344]
[kPage p??]
[ySearch p??]
[$conArgstrings p??]

— defun conPage —

(defun |conPage| (&rest arglist)
  (let ((|$conArgstrings| form da pageName line a b)
    (declare (special |$conArgstrings|))
    (setq a (car arglist))
    (setq b (cdr arglist))
    (setq form (cond ((atom a) (cons a b)) (t a)))
    (setq |$conArgstrings| (loop for x in (cdr a) collect (|form2HtString| x)))
    (cond ((null (atom a)) (setq a (car a))))
    (setq da (downcase a))
    (cond
      ((setq pageName
        (lassq da
          '(|type| . |CategoryType|)
          (|union| . |DomainUnion|)
          (|record| . |DomainRecord|)
          (|mapping| . |DomainMapping|)
        )))
defun gets line quickly for constructor name or abbreviation

(defun |conPageFastPath| (x)
  (let (s name entry lineNumber)
    (declare (special |$lowerCaseConTb|))
    (setq s (stringimage x))
    (unless (> (|#| s) (|charPosition| #\* s 0)) ; quit if name has * in it
      (setq name (cond ((stringp x) (intern x)) (t x)))
      (setq entry (hget |$lowerCaseConTb| name))
      (when entry
        ;'dbLineNumbers property is set by function dbAugmentConstructorDataTable
        (if (setq lineNumber (lassq '|dbLineNumber| (cddr entry)))
          (|dbRead| lineNumber)
          (|conPageConEntry| (car entry)))))))

—— defun conPageFastPath ——

defun conPageConEntry

(defun |buildLIdbbConEntry| (x)
  (let (s name entry lineNumbers)
    (when entry
      ;'dbLineNumbers property is set by function dbAugmentConstructorDataTable
      (if (setq lineNumbers (lassq '|dbLineNumber| (cddr entry)))
        (|dbRead| lineNumbers)
        (|conPageConEntry| (car entry)))))))

—— defun conPageConEntry ——
75.1. PAGES INITIATED FROM HYPERDOC PAGES

(defun conPageConEntry (entry)
  (let ((|$conname| |$conform| |$exposed?| |$doc| |$kind|))
    (declare (special |$conname| |$conform| |$exposed?| |$doc| |$kind|))
    (setq |$conname| nil)
    (setq |$conform| nil)
    (setq |$exposed?| nil)
    (setq |$doc| nil)
    (setq |$kind| nil)
    ([buildLibdbConEntry entry])))

---

defun kdPageInfo

[htSay p??]
[nequal p??]
[bcHt p1260]
[stringimage p??]
[htSaturnBreak p??]
[htSayStandard p??]
[kPageArgs p??]
[length p??]
[extractFileNameFromPath p??]
[subseq p??]
[getdatabase p1010]
[htSay p??]
[htMakePage p1263]

— defun kdPageInfo —

(defun kdPageInfo (name abbrev nargs conform signature file?)
  (let (sourceFileName filename)
    ([htSay] '\{\sf | name "}"
    (when (nequal abbrev name) ([bcHt] (list '|' has abbreviation | abbrev))))
    (when file? (bcHt (list " is a source file.")))
    (cond
      ((eql nargs 0)
       (when (nequal abbrev name) ([bcHt] ".")))
      (t
       (when (nequal abbrev name) ([bcHt] " and"))
       ([bcHt]
        (if (eql nargs 1)
         " takes one argument:"
         (list '|' takes | (stringimage nargs) '| arguments:|))))
       ([htSaturnBreak])
      ([htSayStandard] "\indentrel{2}"
       ([htMakePage p1263])
      ([htMakePage p1263])))

---
(when (> nargs 0) (kPageArgs| conform signature))
(\htSayStandard| "\indentrel{-2}")
(when (char= (elt name (1- (|#| name))) #\&)
  (setq name (subseq name 0 (1- (|#| name)))))
(setq sourceFileName (getdatabase (intern name) 'sourcefile))
(setq filename (|extractFileNameFromPath| sourceFileName))
(when (nequal filename "")
  (\htSayStandard| "\newline{}")
  (\htSay| "The source code for the constructor is found in "))
(\htMakePage|
  (list (list '|text| \unixcommand{ filename }{\$AXIOM/lib/SPADEDIT "
      sourceFileName " " name "")}))
(when (nequal nargs 0) (\htSay| "."))
(\htSaturnBreak|))

| defun |kArgPage|

(defun |kArgPage| (htPage arg)
  (let (conform op args domname source n typeForm domTypeForm descendants rank)
    (setq conform (\htpProperty| htPage '|conform|))
    (setq op (car conform))
    (setq args (cdr conform))
    (setq domname (\htpProperty| htPage '|domname|))
    (setq source (cddar (\getConstructorModemap| op)))
    (setq n (\position| arg args))
    (setq typeForm (\sublisFormal| args (elt source n))))
    (setq domTypeForm (\mkDomTypeForm| typeForm conform domname))
    (setq descendants (\domainDescendantsOf| typeForm domTypeForm))
    (\htpSetProperty| htPage '|cAlist| descendants)
    (setq rank
      (unless (> n 4) (elt '(|First| |Second| |Third| |Fourth| |Fifth|) n)))
    (\htpSetProperty| htPage '|rank| rank)
    (\htpSetProperty| htPage '|thing| "argument")
    (\dbShowCons| htPage '|names|)))
defun reportCategory

(htSay p??)
(categoryParts p??)
(bcConform p??)
(bcPred p??)
(bcConPredTable p??)
(reportAO p1348)

— defun reportCategory —

(defun reportCategory (conform typeForm arg)
  (let (lt1 conlist attrlist oplist)
    (htSay "Argument {\em " arg "}")
    (setq lt1 (categoryParts conform typeForm t))
    (setq conlist (car lt1))
    (setq attrlist (cadr lt1))
    (setq oplist (cddr lt1))
    (htSay " must ")
    (cond
      (conlist
        (htSay "belong to ")
        (cond
          ((and (consp conlist) (eq (qcdr conlist) nil))
            (htSay " category ")
            (bcConform (caar conlist))
            (bcPred (cdar conlist)))
          (t
            (htSay " categories:"))
            (bcConPredTable conlist (opOf conform))
            (htSay \"newline "))))
    (cond
      (attrlist
        (when conlist (htSay " and "))
        (reportAO "attribute" attrlist)
        (htSay \"newline ")))
    (cond
      (oplist
        (when (or conlist attrlist) (htSay " and "))
        (reportAO "operation" oplist))))
defun reportAO

(let (op sig pred attr ops sigs)
  (htSay "have " kind ":")
  (dolist (item oplist)
    (setq op (car item))
    (setq sig (cadr item))
    (setq pred (cddr item))
    (htSay '\newline )
    (when (eql (|= oplist) 1) (htSay "\centerline{")
      (cond
        ((string= kind "attribute")
          (setq attr (form2String (cons op sig)))
          (satDownLink attr (list "(|attrPage| '|" attr "|)") )))
      (t
        (setq ops (escapeSpecialChars (stringimage op)))
        (setq sigs (form2HtString (cons '|Mapping| sig)))
        (satDownLink ops (list "(|opPage| '|" ops "| |" sigs "|)"))
        (htSay ":\")
        (bcConform (cons '|Mapping| sig)))
    (when (eql (|= oplist) 1) (htSay "}") ))
  (htSay '\newline )))

------

defun mkDomTypeForm

(sublis p)
[mkDomTypeForm p]
[hasIndent p]

------

(defun |mkDomTypeForm| (typeForm conform domname)
  (cond
    (domname (sublisis (cdr domname) (cdr conform) typeForm))
    (and (consp typeForm) (eq (qcar typeForm) '|Join|))
      (cons '|Join|)
(loop for t1 in (qcdr typeForm) collect
   (|mkDomTypeForm| t1 conform domname)))
((null (|hasIdent| typeForm)) typeForm))

---

defun domainDescendantsOf

[systemError p??]
[simpHasPred p??]
[quickAnd p??]
[domainsOf p??]
[ifcdr p??]
[qcar p??]
[qcdr p??]
[assoc p??]
[listSort p??]
[function p??]
[delete p??]

--- defun domainDescendantsOf ---

(defun |domainDescendantsOf| (conform domform)
  (labels (
     (catScreen (r alist)
       (let (t1 item pred pred1 npred)
         (dolist (x r)
           (unless (and (consp x) (member (qcar x) '(attribute signature)))
             (|systemError| x))
           (setq alist
             (dolist (anitem alist (nreverse0 t1))
               (setq item (car anitem))
               (setq pred (cdr anitem))
               (when (and
                  (setq pred1 (|simpHasPred| (list '|has| item x)))
                  (setq npred (|quickAnd| pred1 pred)))
                 (setq t1 (cons (cons item npred) t1))))))))
   (jfn (arg domlist)
     (let (y r item pred u keepList alist)
       (setq y (car arg))
       (setq r (cdr arg))
       (setq alist (|domainsOf| y (ifcdr domlist)))
       (dolist (x r)
         (setq domlist (ifcdr domlist))
         (when (and (consp x) (eq (qcar x) 'category) (consp (qcdr x))))
   )
   )

; keep only those domains that appear in ALL parts of Join

(setq alist (catScreen (cddr x) alist))
(setq keepList nil)
(dolist (dom (|domainsOf| x (ifcar domlist)))
  (setq item (car dom))
  (setq pred (cdr dom))
  (when (setq u (|assoc| item alist))
    (setq keepList
      (cons (cons item (|quickAnd| (cdr u) pred)) keepList)))
  (setq alist keepList))
(dolist (pair alist)
  (rplacd pair (|simpHasPred| (cdr pair))))
(|listSort| ([function] glesseqp) alist)))
(if (consp conform)
  (cond
    ((eq (qcar conform) '|Join|)
      (jfn
        (|delete| '(|Type| |Object|) (qcdr conform))
        (|delete| '(|Type| |Object|) (ifcdr domform))))
    ((eq (qcar conform) 'category) nil)
    (t (|domainsOf| conform domform)))
    (|domainsOf| conform domform))))

There are 8 parts of an htPage:

1. kind
2. name
3. nargs
4. xflag
5. sig
6. args
7. abbrev
8. comments

### 75.2 Branches of Constructor Page

defun kiPage

[htpProperty p1254]
[mkConform p1374]
--- defun kiPage ---

(defun |kiPage| (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name nargs args conform domname heading page)
    (declare (special |$conformsAreDomains|))
    (setq lt1 (|htpProperty| htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq args (sixth lt1))
    (setq conform (|mkConform| kind name args))
    (setq domname (|kDomainName| htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
        (|errorPage| htPage domname))
      (t
        (setq heading
          (list "Description of " (make-string (opOf (|capitalize| kind) "\sf " name args) nil) "\})")
        (setq page (|htInitPage| heading (|htCopyProplist| htPage)))
        (setq |$conformsAreDomains| domname)
        (|dbShowConsDoc1| htPage conform nil)
        (|htShowPage|))))

---

defun kePage

(defun |kePage| (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name nargs args conform domname heading page)
    (declare (special |$conformsAreDomains|))
    (setq lt1 (|htpProperty| htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq args (sixth lt1))
    (setq conform (|mkConform| kind name args))
    (setq domname (|kDomainName| htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
        (|errorPage| htPage domname))
      (t
        (setq heading
          (list "Description of " (make-string (opOf (|capitalize| kind) "\sf " name args) nil) "\})")
        (setq page (|htInitPage| heading (|htCopyProplist| htPage)))
        (setq |$conformsAreDomains| domname)
        (|dbShowConsDoc1| htPage conform nil)
        (|htShowPage|))))
(defun |kePage| (htPage junk)
  (declare (ignore junk))
  (let ((|$conformsAreDomains| lt1 kind name nargs args constring domname
         conform conname heading data conlist attrlist oplist prefix page)
    (declare (special |$conformsAreDomains|))
    (setq lt1 (|htpProperty| htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq args (sixth lt1))
    (setq constring (concat name args))
    (setq domname (|kDomainName| htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
       (|errorPage| htPage domname))
      (t
       (|htpSetProperty| htPage '|domname| domname)
       (setq |$conformsAreDomains| domname)
       (setq conform (|mkConform| kind name args))
       (setq conname (|opOf| conform))
       (setq heading
            (list (|capitalize| kind) " {\sf ")
             (if domname ((|form2HtString| domname nil t) constring) "} " ))
       (setq data
            (|sublisFormal|
             (or (ifcdr domname) (cdr conform))))}
(defun getConstructorExports (or domname conform) t))
(setq conlist (car data))
(setq attrlist (cadr data))
(setq oplist (cddr data))
(when domname
  (dolist (x conlist) (rplac (cdr x) (simpHasPred (cdr x))))
  (dolist (x attrlist) (rplac (cddr x) (simpHasPred (cddr x))))
  (dolist (x oplist) (rplac (cddr x) (simpHasPred (cddr x))))
(setq prefix
  (pluralSay (+ (+ (cond conlist) (cond attrlist)) (cond oplist))
             "Export" "Exports"))
(setq page
  (htInitPage (append prefix (cons " of " heading))
              (htCopyProplist htPage)))
(htSayStandard "\beginmenu ")
(htSetProperty page 'data data)
(when conlist
  (htMakePage
   (list
    (list '|bcLinks|
      (list (menuItemButton) " " '|dbShowCons1| conlist '|names|))))
  (htSayStandard "\tab{2}"
     (htSay "All attributes and operations from:")
     (bcConPredTable conlist (opOf conform) (cdr conform)))
(when attrlist
  (when conlist (htBigSkip))
  (kePageDisplay page "attribute" (kePageOpAlist attrlist)))
(when oplist
  (when (or conlist attrlist) (htBigSkip))
  (kePageDisplay page "operation" (kePageOpAlist oplist))
  (htSayStandard " \endmenu ")
  (htShowPage)))

---

defun kePageOpAlist
[lassoc p??]
[insertAlist p??]
[zeroOneConvert p??]

--- defun kePageOpAlist ---

defun |kePageOpAlist| (oplist)
(let (op sig pred u opAlist)
  (dolist (item oplist)
    (setq op (car item))
    
      ---
(setq sig (cadr item))
(setq pred (cddr item))
(setq u (assoc op opAlist))
(setq opAlist
  (insertAlist (zeroOneConvert op)  
    (cons (list sig pred) u)  
    opAlist))))

---

(defun kePageDisplay

[|length p??|]  
[|htpSetProperty p1254|]  
[|htMakePage p1263|]  
[|menuButton p??|]  
[|htSayStandard p??|]  
[|htSay p??|]  
[|stringimage p??|]  
[|pluralize p??|]  
[|htSaySaturn p??|]  
[|dbGatherData p??|]  
[|dbSowOpItems p??|]

— defun kePageDisplay —

(defun |kePageDisplay| (htPage which opAlist)
  (let (count total expandProperty data)
    (setq count (kar opAlist))
    (cond
      ((eql count 0) nil)
      (t
        (setq total
          (apply #'+ (loop for entry in opAlist collect (kar (cdr entry))))))
        (if (string= which "operation"
            (|htpSetProperty| htPage '|opAlist| opAlist)
            (|htpSetProperty| htPage '|attrAlist| opAlist))
        (setq expandProperty
          (if (string= which "operation"
              '|expandOperations|
              '|expandAttributes|)
              (|htpSetProperty| htPage expandProperty '|lists|)
              (|htMakePage|)
              (list
                (list '|bcLinks| (list (|menuButton|) "" '|dbShowOps| which '|names|)))
                (|htSayStandard| "\\\tab{2}"))


(unless (= count total)
  (if (eql count 1)
      (|htSay| "1 name for ")
      (|htSay| (stringimage count) " names for ")))
(if (> total 1)
  (|htSay| (stringimage total) " " (|pluralize| which)
      " are explicitly exported:")
  (|htSay| "1 " which " is explicitly exported:"))
(\htSaySaturn| ”\\"
(setq data (|dbGatherData| htPage opAlist which '|names|)
  (|dbShowOpItems| which data nil))))

---

defun ksPage

(htpProperty p1254]
[kDomainName p1370]
[errorPage p??]
[form2HtString p??]
[htpSetProperty p1254]
[htInitPageNoScroll p??]
[htCopyProplist p??]
[htSay p??]
[htSayStandard p??]
[dbSearchOrder p1356]
[dbShowCons p1387]

— defun ksPage —

(defun |ksPage| (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name nargs domname heading domain conform page u)
    (setq lt1 (|htpProperty| htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq domname (|kDomainName| htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
        (|errorPage| htPage domname))
      (t
        (setq heading
          (if (null domname)
              (|htpProperty| htPage '|heading|)
              (list "\sf " (|form2HtString| domname nil t) ")")))))
  (when domname
defun dbSearchOrder

[opOf p??]
[dbInfovec p??]
[getdatabase p1010]
[simpCatPredicate p??]
[sublis p??]
[kTestPred p1386]
[devaluate p??]
[kFormatSlotDomain p??]
[dbSubConform p1386]
[dbAddChain p1387]
[$domain p??]
[$infovec p??]
[$predvec p??]

— defun dbSearchOrder —

(defun |dbSearchOrder| (conform domname |$domain|)
(declare (special |$domain|))
(let (|$infovec| name u catpredvec catinfo catvec p pred pak catform res
catforms t1)
(declare (special |$infovec| |$predvec|))
(setq conform (or domname conform))
(setq name (=opOf conform))
(setq |$infovec| (dbInfovec name))
(when |$infovec|
  (setq u (elt |$infovec| 3)))
(setq $predvec
  (if $domain| (elt $domain| 3) (getdatabase name 'predicates)))
(setq catpredvec (car u))
(setq catinfo (cadr u))
(setq catvec (caddr u))
(setq catforms
  (dotimes (i (maxindex catvec) (nreverse0 t1))
    (cond
      ((progn
        (setq pred
          (simpCatPredicate)
          (progn
            (setq p
              (sublislis (cdr conform) #$FormalMapVariableList|
                (|kTestPred| (elt catpredvec i)))
            (if $domain| (eval p) p)))
          (when (and domname (contained '$ pred))
            (setq pred (subst domname '$ pred :test #'equal)))
        (and (setq pak (elt catinfo i)) pred))
        (setq t1
          (cons
            (cons
              (cond
                ((and pak (null (identp pak)))
                  (|devaluate| pak))
                (t
                  (setq catform (|kFormatSlotDomain| (elt catvec i)))
                  (setq res (|dbSubConform| (cdr conform)
                    (cons pak (cons '$ (cdr catform)))))
                  (when domname (setq res (subst domname '$ res :test #'equal)))
                  res)
                pred)
              t1))))))
  (append (|dbAddChain| conform) catforms)))

---

defun kcPage

[htpProperty p1254]
[kDomainName p1370]
[qcar p??]
[errorPage p??]
[opOf p??]
[form2HtString p??]
[htInitPage p1262]
[htCopyProplist p??]
--- defun kcPage ---

(defun kcPage (htPage junk)
  (declare (ignore junk))
  (let ((lt1 kind name nargs xpart domname conform conname heading page message)
        (declare (special $defaultPackageNamesHT))
        (setq lt1 (htpProperty htPage 'parts)))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq xpart (fourth lt1))
    (setq domname ((kDomainName htPage kind name nargs))
        (cond
         ((and (consp domname) (eq (qcar domname) 'error))
          (errorPage htPage domname))
         (t
          (setq conform (htpProperty htPage 'conform))
          (setq conname (opOf conform))
          (setq heading
t           (if (null domname)
               (htpProperty htPage 'heading)
               (list "{%sf " (form2HtString domname nil t ) "}")))
          (setq page
t           (htInitPage (cons "Cross Reference for " heading)
                        (htCopyPropList htPage)))
          (when domname
t           (htsetProperty htPage 'domname domname)
           (htsetProperty htPage 'heading heading))
          (when (and (string= kind "category")
             (dbpHasDefaultCategory? xpart))
          (htSay "This category has default package ")
          (bcCon (concat name (char '&) "") )
          ))
  )
defun kcpPage

(defvar htpProperty p1254)
(defvar kDomainName p1370)
(defvar errorPage p??)
(defvar qcar p??)
(defvar form2HtString p??)
(defvar htpSetProperty p1254)
(defvar opOf p??)
(defvar htInitPage p1262)
(defvar htCopyProplist p??)
(defvar parentsOf p??)
(defvar sublislis p??)
(defvar dbShowCons p1387)

--- defun kcpPage ---

(defun kcpPage (htPage junk)
  (declare (ignore junk))
  (let ((lt1 kind name nargs domname heading conform conname page parents choice)
        (setq lt1 (htpProperty htPage 'parts)))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq domname (kDomainName htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) 'error))
       (errorPage htPage domname))
      (t
       (setq heading
            (if (null domname)
                (htpProperty htPage 'heading)
                (list "{\sf " (form2HtString domname nil t) "}")))
       (when domname
        (htpSetProperty htPage 'domname domname)
        (htpSetProperty htPage 'heading heading))
       (setq conform (htpProperty htPage 'conform))
       (setq conname (opOf conform))
       (setq page
            (htInitPage (cons "Parents of " heading) (htCopyProplist htPage)))
       (setq parents (parentsOf conname))
       (when domname
        (setq parents (sublislis (cdr domname) (cdr conform) parents))
        (htpSetProperty htPage 'cAlist parents)
        (htpSetProperty htPage 'thing "parent")
        (setq choice (if domname '(parameters 'names)))
---

defun reduceAlistForDomain
[sublislis p??]
[simpHasPred p??]
[reverse0 p??]

-- defun reduceAlistForDomain --

(defun reduceAlistForDomain (alist domform conform)
(let (pred result)
  (setq alist (sublislis (cdr domform) (cdr conform) alist))
  (dolist (pair alist)
    (rplacd pair (simpHasPred (cdr pair) domform)))
  (dolist (pair alist (reverse0 result))
    (setq pred (cdr pair))
    (when pred (setq result (cons pair result))))))

---

defun kcaPage
[kcaPage1 p1363]
[ancestorsOf p??]

-- defun kcaPage --

(defun kcaPage (htPage junk)
  (declare (ignore junk))
  (kcaPage1 htPage "category" " an "
    "ancestor" (function ancestorsOf) nil))

---

defun kcdPage
[kcaPage1 p1363]
[descendantsOf p??]

-- defun kcdPage --
(defun |kcdPage| (htPage junk)
   (declare (ignore junk))
   ([|kcaPage1| htPage "category" " a "
     "descendant" ([function] [descendantsOf] t))
)

(defun kcdoPage
  [kcdoPage p1363]
  [domainsOf p??]
  
  — defun kcdoPage —

  (defun |kcdoPage| (htPage junk)
     (declare (ignore junk))
     ([|kcaPage1| htPage "domain" " a "
       "descendant" ([function] [domainsOf] nil))
)

(defun kcaPage1
  [htpProperty p1254]
  [kDomainName p1370]
  [errorPage p??]
  [form2HtString p??]
  [htpSetProperty p1254]
  [opOf p??]
  [augmentHasArgs p1365]
  [listSort p??]
  [function p??]
  [dbShowCons p1387]
  
  — defun kcaPage1 —

  (defun |kcaPage1| (htPage kind article whichever fn isCatDescendants?)
     (declare (ignore article))
     (let (lt1 name nargs domname heading conform conname ancestors choice)
          (setq lt1 ([htpProperty] htPage '|parts|))
          (setq kind (first lt1))
          (setq name (second lt1))
          (setq nargs (third lt1))
          (setq domname ([kDomainName] htPage kind name nargs))
)
defun kccPage

(htpProperty p1254)
[kDomainName p1370]
[qcar p??]
[errorPage p??]
[form2HtString p??]
[htpSetProperty p1254]
[opOf p??]
[htInitPage p1262]
[htCopyProplist p??]
[augmentHasArgs p1365]
[childrenOf p??]
[reduceAlistForDomain p1362]
[dbShowCons p1387]

— defun kccPage —

(defun |kccPage| (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name nargs domname heading conform conname page children)
    (setq lt1 (htpProperty htPage '|parts|))
    ...
(setq kind (first lt1))
(setq name (second lt1))
(setq nargs (third lt1))
(setq domname (|kDomainName| htPage kind name nargs))
(cond
  ((and (consp domname) (eq (qcar domname) '|error|))
   (|errorPage| htPage domname))
  (t
   (setq heading
     (if (null domname)
       (|htpProperty| htPage '|heading|)
       (list "\{sf " (|form2HtString| domname nil t) "\}")))
   (when domname
    (|htpSetProperty| htPage '|domname| domname)
    (|htpSetProperty| htPage '|heading| heading))
   (setq conform (|htpProperty| htPage '|conform|))
   (setq conname (|opOf| conform))
   (setq page
     (|htInitPage| (cons "Children of " heading) (|htCopyProplist| htPage)))
   (setq children (|augmentHasArgs| (|childrenOf| conform) conform))
   (when domname
    (setq children (|reduceAlistForDomain| children domname conform)))
   (|htpSetProperty| htPage '|cAlist| children)
   (|htpSetProperty| htPage '|thing| "child")
   (|dbShowCons| htPage '|names|)))))

defun augmentHasArgs

[opOf p??]
[kdr p??]
[length p??]
[nreverse0 p??]
[extractHasArgs p??]
[getConstructorForm p??]

— defun augmentHasArgs —

(defun |augmentHasArgs| (alist conform)
  (let (conname args n name p result pred)
    (setq conname (|opOf| conform))
    (setq args (kdr conform))
    (cond
      (args
        (setq n (|#| args))
        (dolist (item alist (nreverse0 result)))
        (if (null item)
            (setq name (|form2HtString| name nil t))
            (setq name (|form2HtString| (list (car name) item) nil t)))
        (setq name (|form2HtString| name nil t))
        (setq name (|form2HtString| (list (car name) "", item) nil t))
        (setq name (|form2HtString| name nil t))
        (setq name (|form2HtString| (list (car name) "", item) nil t))
        (setq name (|form2HtString| name nil t))
        (setq name (|form2HtString| (list (car name) "", item) nil t)))
      (else
        (setq name (|form2HtString| name nil t))
        (setq name (|form2HtString| (list (car name) "", item) nil t))
        (setq name (|form2HtString| name nil t))
        (setq name (|form2HtString| (list (car name) "", item) nil t))
        (setq name (|form2HtString| name nil t))
        (setq name (|form2HtString| (list (car name) "", item) nil t)))
    (when name
      (setq name (|form2HtString| name nil t))
      (setq name (|form2HtString| (list (car name) "", item) nil t))
      (setq name (|form2HtString| name nil t))
      (setq name (|form2HtString| (list (car name) "", item) nil t))
      (setq name (|form2HtString| name nil t))
      (setq name (|form2HtString| (list (car name) "", item) nil t))))
    (setq conform (|htpProperty| htPage '|conform|))
    (setq conname (|opOf| conform))
    (setq page
      (|htInitPage| (cons name "Children of " heading) (|htCopyProplist| htPage)))
    (setq children (|augmentHasArgs| (|childrenOf| conform) conform))
    (when domname
      (setq children (|reduceAlistForDomain| children domname conform)))
    (|htpSetProperty| htPage '|cAlist| children)
    (|htpSetProperty| htPage '|thing| "child")
    (|dbShowCons| htPage '|names|))

---

---
(setq name (car item))
(setq p (cdr item))
(setq pred
  (if (consp (|extractHasArgs| p))
      p
    (|quickAnd| p
      (cons 'hasArgs
        (take n (kdr (|getConstructorForm| (|opOf| name)))))))))
(setq result (cons (cons name pred) result))))
(t alist))

---

defun kcdePage

[htpProperty p1254]
[concat p1047]
[nequal p??]
[ncParseFromString p1067]
[opOf p??]
[getDependentsOfConstructor p1367]
[getConstructorForm p??]
[htpSetProperty p1254]
[dbShowCons p1387]

— defun kcdePage —

(defun |kcdePage| (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name args conname constring conform pakname domlist cAlist)
    (setq lt1 (|htpProperty| htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq args (sixth lt1))
    (setq conname (intern name))
    (setq constring (concat name args))
    (setq conform
      (if (nequal kind "default package")
        (|ncParseFromString| constring)
        (cons (intern name) (cdr (|ncParseFromString| (concat #\d args))))))
    (setq pakname (|opOf| conform))
    (setq domlist (|getDependentsOfConstructor| pakname))
    (setq cAlist
      (loop for x in domList collect (cons (|getConstructorForm| x) t)))
    (|htpSetProperty| htPage '|cAlist| cAlist)
    (|htpSetProperty| htPage '|thing| "dependent")
    (|dbShowCons| htPage '|names|))))
defun getDependentsOfConstructor

(defun getDependentsOfConstructor (con)
 (let (stream val)
   (setq stream
     (readLibPathFast (pathname (list 'dependents 'database 'a))))
   (setq val (rread con stream nil))
   (rshut stream)
   val))

defun kcuPage

(defun kcuPage (htPage junk)
 (declare (ignore junk))
 (let (lt1 kind name args conname constring conform pakname domlist cAlist)
   (setq lt1 (htpProperty htPage 'parts))
   (setq kind (first lt1))
   (setq name (second lt1))
   (setq args (sixth lt1))
   (setq conname (intern name)))
(setq constring (concat name args))
(setq conform
  (if (neq kind "default package")
      (|ncParseFromString| constring)
      (cons (intern name)
        (cdr (|ncParseFromString| (concat #\d args))))))
(setq pakname
  (if (string= kind "category")
      (intern (concat name #\&))
      (|opOf| conform)))
(setq domlist (|getUsersOfConstructor| pakname))
(setq cAlist
  (loop for x in domlist collect (cons (|getConstructorForm| x) t)))
(|htpSetProperty| htPage '|cAlist| cAlist)
(|htpSetProperty| htPage '|thing| "user")
(|dbShowCons| htPage '|names|))

defun getUsersOfConstructor

|defun getUsersOfConstructor |

defun kcnPage

defun kcnPage
75.2. BRANCHES OF CONSTRUCTOR PAGE

(defun kcnPage (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name nargs domname heading conform pakname domlist cAlist)
    (setq lt1 (htpProperty htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq domname (kDomainName htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
       (errorPage htPage domname))
      (t
       (setq heading
            (if (null domname)
                (htpProperty htPage '|heading|)
                (list "{{\sf " (form2HtString domname nil t) "}}")))
            (if domname
                (htpSetProperty htPage '|domname| domname)
                (htpSetProperty htPage '|heading| heading))
        (setq conform (htpProperty htPage '|conform|))
        (setq pakname
            (if (string= kind "category")
                (intern (concat (pname conname) #\&))
                (opOf conform)))
        (setq domlist (getImports pakname))
        (when domname
          (setq domlist
              (sublis (cons domname (cdr domname))
                (cons '('$ (cdr conform)) domlist)))
        (setq cAlist (loop for x in domList collect (cons x t)))
        (htpSetProperty htPage '|cAlist| cAlist)
        (htpSetProperty htPage '|thing| "benefactor")
        (dbShowCons htPage '|names|))))
  — defun kcnPage —
defun koPageInputAreaUnchanged?

(htpLabelInputString p)
(concat p)
(stringimage p)
(htpProperty p)

— defun koPageInputAreaUnchanged? —

(defun |koPageInputAreaUnchanged?| (htPage nargs)
  (equal
   (loop for i from 1 to nargs
         collect
           ((htpLabelInputString| htpPage (intern (concat "*" (stringimage i))))
            (htpProperty| htpPage '|inputAreaList|))))

—

defun kDomainName

(htpSetProperty p)
(htpLabelInputString p)
(getdatabase p)
(kArgumentCheck p)
[kdr p]
(concat p)
(unabbrev p)
[mkConform p]
[kisvalidType p]
[dbMkEvalable p]
[spad-reader p]
[&PatternVariableList p]

— defun kDomainName —

(defun |kDomainName| (htPage kind name nargs)
  (let (inputAreaList conname args n argTailPart argString typeForm
tenualTypeForm)
    (htpSetProperty| htpPage '|domname| nil)
    (setq inputAreaList
      (loop for i from 1 to nargs for var in |$PatternVariableList| do
        collect ((htpLabelInputString| htpPage var))
      (htpSetProperty| htpPage '|inputAreaList| inputAreaList)
      (setq conname (intern name))
      (setq args
        (loop for x in inputAreaList
          ...)
for domain? in (cdr (getdatabase conname 'cosig))
collect (or (kArgumentCheck domain? x) nil))
(when (some #'identity (loop for x in args collect (null x)))
  (cond
    ((> (setq n (apply #'+ (loop for x in args collect (if x 1 0)))) 0)
     (list '|error| nil "\centerline{You gave values for only \{\em "
           n " \} of the \{\em " (\#| args) "\}"
         "\centerline{parameters of \{\sf " name
           "\}\}vspace{1}\centerline{Please enter either \{\em all\} or "
         "\{\em none\} of the type parameters\}"
        nil)
    (t
      (setq argString
        (cond
          ((null args) "()")
          (t
           (setq argTailPart
             (apply #'concat
               (loop for x in (kdr args) collect (concat (cons "," x)))))
           (apply #'concat (list "(" (car args) argTailPart ")"))))
      (setq typeForm
        (or (catch 'spad_reader (unabbrev (mkConform kind name argString)))
            (list '|error| '|invalidType| (concat name argString)))))
      (if (null (setq evaluatedTypeForm (kisValidType typeForm)))
        (list '|error| '|invalidType| (concat name argString))
        (dbMkEvalable evaluatedTypeForm)))))
  )
  )
  )))
}

——

defun kArgumentCheck

[conSpecialString? p??]
[kdr p??]
[stringimage p??]
[opOf p??]
[form2String p??]

— defun kArgumentCheck —

(defun |kArgumentCheck| (domain? s)
  (let (form)
    (cond
    (((string= s "") nil)
     ((and domain? (setq form (conSpecialString? s)))
      (if (null (kdr form))
        (list (stringimage (opOf form)))
        (list (form2String form))))
    )
    )
    ))
(t (list s))))


defun dbMkEvalable

(defun dbMkEvalable (form)
  (let (op kind)
    (setq op (car form))
    (setq kind (getdatabase op 'constructorkind))
    (if (eq kind '|category|)
       form
       (|mkEvalable| form))))


defun topLevelInterpEval

(defun topLevelInterpEval (x)
  (let (|$ProcessInteractiveValue| |$noEvalTypeMsg|)
    (declare (special |$ProcessInteractiveValue| |$noEvalTypeMsg|))
    (setq |$ProcessInteractiveValue| t)
    (setq |$noEvalTypeMsg| t)
    (|processInteractive| x nil)))


defun kisValidType

(defun kisValidType

(defun kisValidType (x)
  (let (|$ProcessInteractiveValue| |$noEvalTypeMsg|)
    (declare (special |$ProcessInteractiveValue| |$noEvalTypeMsg|))
    (setq |$ProcessInteractiveValue| t)
    (setq |$noEvalTypeMsg| t)
    (|processInteractive| x nil)))
--- defun kisValidType ---

(defun kisValidType (typeForm)
  (let (|$ProcessInteractiveValue| |$noEvalTypeMsg| it1)
    (declare (special |$ProcessInteractiveValue| |$noEvalTypeMsg|))
    (setq |$ProcessInteractiveValue| t)
    (setq |$noEvalTypeMsg| t)
    (setq it1 (catch 'spad_reader (|processInteractive| typeForm nil)))
    (when (and (consp it1) (consp (qcar it1)))
      (|member| (caar it1) '(domain |SubDomain|)))
    (and (|kCheckArgumentNumbers| (qcdr it1)) (qcdr it1))))

---

defun kCheckArgumentNumbers

--- defun kCheckArgumentNumbers ---

(defun |kCheckArgumentNumbers| (tt)
  (let (conname args cosig)
    (setq conname (car tt))
    (setq args (cdr tt))
    (setq cosig (kdr (getdatabase conname 'cosig)))
    (every #’identity
      (loop for domain? in cosig for x in args
        collect (if domain? (|kCheckArgumentNumbers| x) t)))))))

---

defun parseNoMacroFromString

---
— defun parseNoMacroFromString —

(defun parseNoMacroFromString (s)
  (setq s
    (|next| (|function| |ncloopParse|)
    (|next| (|function| |lineoftoks|)
    (incString| s))))
  (if (|StreamNull| s)
    nil
    (|pf2Sex| (cadar s))))

——

defun mkConform

(nequal p??]
   [parseNoMacroFromString p1373]
   [sayBrightlyNT p??]
   [pp p??]
   [systemError p??]
   [ncParseFromString p1067]
   [concat p1047]

— defun mkConform —

(defun mkConform (kind name argString)
  (let (form parse)
    (cond
      ((nequal kind "default package")
        (setq form (concat name argString))
        (setq parse (|parseNoMacroFromString| form))
        (cond
          (null parse)
            (|sayBrightlyNT| "Won't parse: ")
            (|pp| form)
            (|systemError| "Keywords in argument list?")
            ((atom parse) (cons parse nil))
            (t parse))
      (t (cons (intern name) (cdr (|ncParseFromString| (concat #\d argString)))))))

——
75.3  OPERATION PAGE FOR A DOMAIN FORM FROM SCRATCH

defun conOpPage

(dbCompositeWithMap p1377)
[htpProperty p1254]
[conOpPage1 p1375]
[dbExtractUnderlyingDomain p1377]

— defun conOpPage —

(defun |conOpPage| (htPage conform)
 (declare (ignore conform))
 (let (updown domname)
 (setq updown (|dbCompositeWithMap| htPage))
 (cond
   ((string= updown "DOWN")
    (setq domname (|htpProperty| htPage '|domname|))
    (|conOpPage1| (|dbExtractUnderlyingDomain| domname)
     (list (cons '|updomain| domname))))
   (t
    (setq domname (|htpProperty| htPage '|updomain|))
    (|conOpPage1| domname nil)))))

---

defun conOpPage1

(ifcar p??)
[opOf p??]
[dbSpecialOperations p1398]
[conPageFastPath p1344]
[dbXParts p??]
[concat p1047]
[mkConform p1374]
[capitalize p??]
[ncParseFromString p1067]
[dbSourceFile p??]
[isExposedConstructor p820]
[htInitPage p1262]
[htpSetProperty p1254]
[lassoc p??]
[ifcdr p??]
[koPage p1377]
[$Primitives p??]
— defun conOpPage1 —

(defun conOpPage1 (&rest args)
  (let (bindingsAlist conname domname line parts name sig args isFile kind
constring capitalKind signature sourceFileName emString heading page
selectedOperation a b options conform)
    (declare (special #$Primitives$))
    (setq conform (car args))
    (setq options (cdr args))
    (setq bindingsAlist (ifcar options))
    (setq conname (opOf conform))
    (cond
      ((member conname #$Primitives$) (dbSpecialOperations conname))
      (t
        (setq domname (unless (atom conform) conform))
        (setq line (conPageFastPath conname))
        (setq parts (dbXParts line 7 1))
        (setq kind (first parts))
        (setq name (second parts))
        (setq sig (fifth parts))
        (setq args (sixth parts))
        (setq isFile (null kind))
        (setq kind (or kind "package"))
        (rplaca parts kind)
        (setq constring (concat name args))
        (setq conform (mkConform kind name args))
        (setq capitalKind (capitalize kind))
        (setq signature (ncParseFromString sig))
        (setq sourceFileName (dbSourceFile (intern name)))
        (setq emString (list "\{sf constring \}"))
        (setq heading (cons capitalKind (cons " " emString)))
        (unless (isExposedConstructor conname)
          (setq heading (cons "Unexposed " heading)))
        (setq page (htInitPage heading nil))
        (htSetProperty page '|isFile| t)
        (htSetProperty page '|fromConOpPage1| t)
        (htSetProperty page '|parts| parts)
        (htSetProperty page '|heading| heading)
        (htSetProperty page '|kind| kind)
        (htSetProperty page '|domname| domname)
        (htSetProperty page '|conform| conform)
        (htSetProperty page '|signature| signature)
        (when
          (setq selectedOperation (lassoc '|selectedOperation| (ifcdr options)))
          (htSetProperty page '|selectedOperation| selectedOperation))
        (loop for item in bindingsAlist
          collect (htSetProperty page (car item) (cdr item)))
        (koPage page "operation"))))
defun dbCompositeWithMap
  (htpProperty htPage dbExtractUnderlyingDomain)

  (defun dbCompositeWithMap (htPage)
    (let (domain opAlist)
      (cond
        ((htpProperty htPage 'updomain) "UP")
        (t
          (setq domain (htpProperty htPage 'domname))
          (cond
            (null domain) nil)
            (t
              (setq opAlist (htpProperty htPage 'opAlist))
              (when
                (dbExtractUnderlyingDomain (htpProperty htPage 'domname))
                "DOWN"))))))

defun dbExtractUnderlyingDomain
  (kdr isValidType)

  (defun dbExtractUnderlyingDomain (domain)
    (some #'identity
      (loop for x in (kdr domain) when (isValidType x) collect x)))

75.4  Operation Page from Main Page

defun koPage
--- defun koPage ---

(defun koPage (htPage which)
  (let ((lt1 kind name nargs args constring conname u IT1 domname headingString
         heading)
        (setq lt1 (htpProperty htPage '|parts|))
        (setq kind (first lt1))
        (setq name (second lt1))
        (setq nargs (third lt1))
        (setq args (sixth lt1))
        (setq constring (concat name args))
        (setq conname (intern name))
        (setq IT1 (setq u (htpProperty htPage '|domname|)))
        (setq domname (cond ((and (consp IT1) (equal (qcar IT1) conname)
                                   (or (eq (htpProperty htPage '|fromConOpPage1|) t)
                                       (koPageInputAreaUnchanged? htPage nargs)))
                               u)
             (t (kDomainName htPage kind name nargs)))
        (setq domname (cond ((and (consp domname) (eq (qcar domname) '|error|))
                              (errorPage htPage domname))
             (t (koPageFromKKPage htPage domname))
        (setq headingString (if domname (form2HtString domname nil t) constring))
        (setq heading (list (capitalize kind) " {\sf " headingString "} "))
        (htpSetProperty htPage '|domname| domname)
        (htpSetProperty htPage '|which| which)
        (htpSetProperty htPage '|heading| heading)
        (koPageAux htPage which domname heading)))))

-------

defun koPageFromKKPage

[koPageAux p1379]
[htpProperty p1254]

--- defun koPageFromKKPage ---
defun koPageFromKKPage
  (htPage ao)
  (koPageAux htPage ao (htpProperty htPage 'domname))
  (htpProperty htPage 'heading))

------

defun koPageAux

(htpSetProperty p1254)
  [koAttrs p??]
  [koOps p??]
  [assoc p??]
  [systemError p??]
  [dbShowOperationsFromConform p??]

  defun koPageAux

  (defun koPageAux (htPage which domname heading)
    (let (conform selectedOperation opAlist)
      (htpSetProperty htPage 'which which)
      (setq domname (htpProperty htPage 'domname))
      (setq conform (htpProperty htPage 'conform))
      (setq heading (htpProperty htPage 'heading))
      (setq opAlist
        (cond
          ((string= which "attribute") (koAttrs conform domname))
          ((string= which "general operation") (koOps conform domname t))
          (t (koOps conform domname)))))
      (cond
        ((setq selectedOperation (htpProperty htPage 'selectedOperation))
        (setq opAlist
          (list (or (assoc selectedOperation opAlist) (systemError))))))
      (dbShowOperationsFromConform htPage which opAlist)))

------

defun koPageAux1

(htpProperty p1254)
  [dbShowOperationsFromConform p??]

  defun koPageAux1

  (defun koPageAux1 (htPage opAlist)
    (let (which)
(setq which (|htpProperty| htPage '|which|))
(|dbShowOperationsFromConform| htPage which opAlist)))

defun koaPageFilterByName

[htpLabelInputString p1254]
[koaPageFilterByCategory p??]
[pmTransFilter p??]
[htpProperty p1254]
[dbGetInputString p??]
[superMatch? p??]
[downcase p??]
[stringimage p??]
[htpSetProperty p1254]

— defun koaPageFilterByName —

(defun |koaPageFilterByName| (htPage functionToCall)
  (let (filter which opAlist)
    (cond
      ((string= (|htpLabelInputString| htPage '|filter|) "")
       (|koaPageFilterByCategory| htPage functionToCall))
      (t
       (setq filter (|pmTransFilter| (|dbGetInputString| htPage)))
       (setq which (|htpProperty| htPage '|which|))
       (setq opAlist
         (loop for x in (|htpProperty| htPage '|opAlist|)
           when ((|superMatch?| filter (downcase (stringimage (car x))))
             collect x))
       (|htpSetProperty| htPage '|opAlist| opAlist)
       (funcall functionToCall htPage nil))))

75.5 Get Constructor Documentation

defun dbConstructorDoc,hn

[length p??]
[sublis p??]
[$FormalMapVariableList p??]
defun dbConstructorDoc,hn

(defvar dbConstructorDoc,hn (sig)
  (declare (special |$sig| |$args|)))
(and (equal (#| |$sig|) (#| sig|))
    (equal |$sig| (sublislis |$args| |$FormalMapVariableList| |$sig|)))

defun dbConstructorDoc,gn

(defvar dbConstructorDoc,gn (arg)
  (let (op alist sig doc)
    (declare (special |$op|))
    (setq op (car arg))
    (setq alist (cdr arg))
    (and |$op|
      (some #'identity
        (loop for item in alist when (dbConstructorDoc,hn (car item))
          collect (or (cdr item) ""))))))

defun dbConstructorDoc

(defvar dbConstructorDoc,fn (conform |$op| |$sig|)
  (declare (special |$op| |$sig|))
  (dbConstructorDoc,fn conform))
defun dbDocTable

(defvar $docTable p1382)
(defvar $docTableHash p1383)

(defun dbDocTable (conform)
  (let ((|$docTable| table)
        (declare (special |$docTable| |$docTableHash|))
        (cond
           ((setq table (hget |$docTableHash| conform))
            table)
           (t
            (setq |$docTable| (make-hashtable 'id))
            (loop for x in (|originsInOrder| conform) do (|dbAddDocTable| x))
            (|dbAddDocTable| conform)
            (hput |$docTableHash| conform |$docTable|)
            |$docTable||)))

defun originsInOrder

(defvar getdatabase p1010)
(defvar assocleft p1382)
(defvar ancestorsOf p1382)
(defvar parentsOf p1382)
(defvar originsInOrder p1382)
(defvar insert p1382)

(defun originsInOrder (conform)
  (let (con argl acc)
    (setq con (car conform))
    (setq argl (cdr conform))
    (cond
       ((eq (getdatabase con 'constructor-kind) '|category|)
        (assocleft (|ancestorsOf| conform nil)))
       (t
        (setq acc (assocleft (|parentsOf| conform)))))

  (defun |dbDocTable| (conform)
    (let (|$docTableHash|)
      (declare (special |$docTable| |$docTableHash|))
      (cond
         ((setq table (hget |$docTableHash| conform)))
         (t
          (setq |$docTable| (make-hashtable 'id))
          (loop for x in (|originsInOrder| conform) do (|dbAddDocTable| x))
          (|dbAddDocTable| conform)
          (hput |$docTableHash| conform |$docTable|)
          |$docTable||))

— defun dbDocTable —

— defun originsInOrder —
(loop for x in acc do
  (loop for y in (originsInOrder x) do
    (setq acc (insert y acc)))
  acc)))

---

(defun dbAddDocTable
  [opOf p??]
  [getConstructorForm p??]
  [sublislis p??]
  [getdatabase p1010]
  [hput p1044]
  [hget p1044]
  ['$docTable p??]

  --- defun dbAddDocTable ---

(defun |dbAddDocTable| (conform)
  (let (conname storedArgs op alist op1 sig doc tmp)
    (declare (special |$docTable|))
    (setq conname (opOf conform))
    (setq storedArgs (cdr (getConstructorForm conname)))
    (setq tmp (sublislis (cons '$ (cdr conform)) (cons '%' storedArgs)
      (getdatabase (opOf conform) 'documentation)))
    (loop for item in tmp do
      (setq op (car item))
      (setq alist (cdr item))
      (setq op1
        (cond
          ((eq op '(|Zero|)) 0)
          ((eq op '(|One|)) 1)
          (t op)))
      (loop for item1 in alist do
        (setq sig (first item1))
        (setq doc (second item1))
        (hput |$docTable| op1 (cons (cons conform alist)
          (hget |$docTable| op1)))))))

---
defun dbGetDocTable,hn

[sublis p??]
[kdr p??]
[qcdr p??]
[qcar p??]
[$which p??]
[$conform p??]
[$sig p??]
[$FormalMapVariableList p??]

— defun dbGetDocTable,hn —

(defun |dbGetDocTable,hn| (arg)
(let (sig doc alteredSig pred r)
(declare (special |$which| |$conform| |$sig| |$FormalMapVariableList|))
(setq sig (car arg))
(setq doc (cdr arg))
(if (string= |$which| "attribute")
(and (consp sig) (eq (qcar sig) '|attribute|) (equal (qcdr sig) |$sig|)
(doc)
(progn
(setq pred
(and
(eq1 (|^| |$sig|) (^| sig))
(setq alteredSig
(sublis (kdr |$conform|) |$FormalMapVariableList| sig))
(equal alteredSig |$sig|)))
(when (and pred doc
(and (consp doc) (eq (qcar doc) '|constant|)) (qcdr doc) doc)
,'(""))))))

— defun dbGetDocTable,gn —

defun dbGetDocTable,gn

[lastatom p??]
[dbGetDocTable,hn p1384]
[$conform p??]

(defun |dbGetDocTable,gn| (u)
(let (code p comments)
(declare (special |$conform|)))
(setq |$conform| (car u))
(when (atom |$conform|) (setq |$conform| (list |$conform|)))
(setq code (lastatom u))
(setq comments
  (some #'identity
    (loop for entry in (cdr u)
      when (setq p (|dbGetDocTable,hn| entry))
        collect p)))
(when comments (cons |$conform| (cons (car comments) code))))

-----

defun dbGetDocTable

[stringimage p??]
[string2Integer p??]
[dbConstructorDoc p1381]
[qcdr p??]
[hget p1044]
[dbGetDocTable,gn p1384]
[$sig p??]
[$which p??]
[$conform p??]
[$op p??]

--- defun dbGetDocTable ---

(defvar |dbGetDocTable| (op |$sig| docTable |$which| aux)
  (declare (special |$sig| |$which|))
  (let (doc origin)
    (declare (special |$conform| |$op|))
    (when (and (null (integerp op)) (digitp (elt (setq s (stringimage op)) 0)))
      (setq op (|string2Integer| s)))
    (cond
      ((and (consp aux) (consp (qcar aux)))
        (setq doc (|dbConstructorDoc| (car aux) |$op| |$sig|)))
      (setq origin (if (qcdr aux) (cons '|ifp| aux) (car aux))))
      (cons origin doc))
    (t
     (some #'identity
       (loop for x in (hget docTable op)
          collect (|dbGetDocTable,gn| x))))))

-----
defun kTestPred

[|testBitVector| |simpHasPred| |$predvec| |$domain|]

— defun kTestPred —

(defun |kTestPred| (n)
 (declare (special |$predvec| |$domain|))
 (cond
  ((eql n 0) t)
  (|$domain| (|testBitVector| |$predvec| n))
  (t (|simpHasPred| (elt |$predvec| (1- n))))))

—

defun dbAddChainDomain

[|dbInfovec|]
[|dbSubConform|]
[|kFormatSlotDomain|]
[|devaluate|]
[|$infovec|]

— defun dbAddChainDomain —

(defun |dbAddChainDomain| (conform)
 (let (name args template form)
   (declare (special |$infovec|))
   (setq name (car conform))
   (setq args (cdr conform))
   (setq |$infovec| (|dbInfovec| name))
   (when |$infovec|
     (setq template (elt |$infovec| 0))
     (when (setq form (elt template 5))
       (|dbSubConform| args (|kFormatSlotDomain| (|devaluate| form))))))

—

defun dbSubConform

[|position|]
[|dbSubConform|]
75.6. CONSTRUCTOR PAGE MENU

(defun dbSubConform
  (args u)
  (let (n y)
    (declare (special $FormalMapVariableList))
    (cond
      ((atom u)
       (if (>= (setq n (position u $FormalMapVariableList)) 0)
        (elt args n)
        u))
      ((and (consp u) (eq (car u) 'local) (consp (cdr u)) (eq (cddr u) nil))
        (setq y (cadr u))
        (dbSubConform args y))
      (t
       (loop for x in u collect (dbSubConform args x)))))))

(defun dbAddChain
  (conform)
  (let (u)
    (when (setq u (dbAddChainDomain conform))
      (unless (atom u)
        (cons (cons u t) (dbAddChain u))))))

75.6 Constructor Page Menu

defun dbShowCons

defun dbShowCons (rest args &aux options key htPage)
  (let (cAlist filter abbrev? conname subject u htPage key options)
    (declare (special \$exposedOnlyIfTrue))
    (setq htPage (first args))
    (setq key (second args))
    (setq options (cddr args))
    (setq cAlist (htpProperty htPage \'cAlist)))
  (cond
    ((eq key \'filter)
      (setq filter
        (pmTransFilter (or (ifcar options) (dbGetInputString htPage))))
      (cond
       ((and (consp filter) (eq (car filter) \'error))
        (bcErrorPage filter))
     (t
      (setq abbrev? (eq (htpProperty htPage \'exclusion) \'abbrs))
      (setq u
        (loop for x in cAlist
          when (progn
            (setq conname (caar x))
            (setq subject (if abbrev? (constructor? conname) conname))
            (superMatch? filter (downcase (stringimage subject))))
          collect x))
      (cond
       (null u)
        (emptySearchPage "constructor" filter))
      (t
        (setq htPage (htInitPageNoScroll (htCopyProplist htPage))
          (htSetProperty htPage \'cAlist u)
          (dbShowCons htPage (htpProperty htPage \'exclusion)))))))
  (t
    (when (member key (\'exposureOn |exposureOff\))
      (setq \$exposedOnlyIfTrue (eq key \'exposureOn))
      (setq key (htpProperty htPage \'exclusion))))
(\texttt{|dbShowCons1| htPage cAlist key))))

\textbf{defun conPageChoose}

\begin{verbatim}
(defun conPageChoose (conname)
  (let (cAlist)
    (setq cAlist (list (cons (\texttt{|getConstructorForm|} conname) t)))
    (\texttt{|dbShowCons1| nil cAlist ’names))))
\end{verbatim}

\textbf{defun dbShowCons1}

\begin{verbatim}
(defun |dbShowCons1| (conname)
  (let (cAlist)
    (setq cAlist (list (cons (|getConstructorForm| conname) t)))
    (\texttt{|dbShowCons1| nil cAlist ’names))))
\end{verbatim}
— defun dbShowCons1 —

(defun dbShowCons1 (htPage cAlist key)
  (let ((|$conformsAreDomains| item conlist kinds a kind
         proplist page u fn y flist result)
        (declare (special|$conformsAreDomains||$exposedOnlyIfTrue|)))
    (setq conlist
      (remdup
       (dolist (x cAlist result)
        (push
         (if|$exposedOnlyIfTrue|
          (isExposedConstructor|opOf|(car x)))
          (car x))
         result))))
  (cond
   ((and (consp conlist) (eq (qcdr conlist) nil))
    (conPage)
    ((and htPage (htpProperty htPage '|domname|))
     (car conlist)
     (|opOf|(car conlist))))
   (t
    (setq conlist (loop for x in conlist collect (|opOf|x)))
    (setq kinds
      (apply #'|union|
        (loop for x in conlist collect (|dbConstructorKind|x))))
    (setq kind
      (if (and (consp kinds) (eq (qcdr kinds) nil))
       (qcar kinds)
       '|constructor|)
    (setq proplist (when htPage (|htCopyProplist| htPage)))
    (setq page
      (|htInitPageNoScroll| proplist
       (|dbConsHeading| htPage conlist key kind)))
    (if (setq u (|htpProperty| page '|specialMessage|))
      (apply (car u) (cdr u)))
    (|htSayStandard| "\beginscroll ")
    (|httpSetProperty| page '|cAlist| cAlist)
    (setq|$conformsAreDomains| (|htpProperty| page '|domname|))
    (cond
      ((eq key '|names|) (|bcNameConTable| conlist))
      ...)
defun dbConsExposureMessage

(htSay p??)
($atLeastOneUnexposed p??)

— defun dbConsExposureMessage —

(defun |dbConsExposureMessage| ()
  (declare (special $atLeastOneUnexposed))
  (when $atLeastOneUnexposed
    (|htSay| \"\newline{}-------------\newline{}{\em *} = unexposed\\})

——

defun dbShowConsKindsFilter

(htpSetProperty p1254)
[dbShowCons p1387]
--- defun dbShowConsKindsFilter ---

(defun |dbShowConsKindsFilter| (htPage args)
  (|htpSetProperty| htPage '|cAlist| (second args))
  (|dbShowCons| htPage (|htpProperty| htPage '|exclusion|)))

---

defun dbShowConsDoc

--- defun dbShowConsDoc ---

(defun |dbShowConsDoc| (htPage conlist)
  (labels ((fn (cAlist x)
      (let ((index 0))
        (loop while (not (equal (caaar cAlist) x))
          do (setq index (1+ index))
          (setq cAlist (cdr cAlist))
          (unless cAlist (|systemError|)))
         index))
    (let (index cAlist)
      (cond ((null (cdr conlist))
        (|dbShowConsDoc1| htPage
          (|getConstructorForm| (|opOf| (car conlist))) nil))
      (t
        (setq cAlist (|htpProperty| htPage '|cAlist|))
        (loop for x in (remdup conlist) do
          (|dbShowConsDoc1| htPage
            (|getConstructorForm| x) (fn cAlist x)))))))
defun dbShowConsDoc1

(member p1048)
(htpProperty p1254)
(getl p1050)
(displayDomainOp p??)
(isExposedConstructor p820)
(getConstructorDocumentation p1394)
(getConstructorSignature p??)
(getdatabase p1010)
(sublis p??)
(sublisFormal p??)
(htSaySaturn p??)
(displayDomainOp p??)
(htSaySaturn p??)
($TriangleVariableList p??)
($Primitives p??)

— defun dbShowConsDoc1 —

(defun |dbShowConsDoc1| (htPage conform indexOrNil)
(let (conargs conname lt1 exposeFlag doc signature sig)
(declare (special |$TriangleVariableList| |$Primitives|))
(setq conname (car conform))
(setq conargs (cdr conform))
(cond
  ((member conname |$Primitives|)
    (setq conname (|htpProperty| htPage '|conname|))
    (setq lt1 (getl conname '|documentation|))
    (cond ((eq (caar lt1) '|constructor|) (caar lt1))
           ((eq (caadar lt1) 'nil) (caadar lt1))
           (setq doc (car (cdadar lt1))))
    (setq sig '((category domain) (%SetCategory% %SetCategory%))
      (displayDomainOp| htPage |"constructor"
      conform conname sig t doc indexOrNil |dbSelectCon| nil nil))
  (t
    (setq exposeFlag (isExposedConstructor conname))
    (setq doc (list (getConstructorDocumentation conname))
      (setq signature (getConstructorSignature conname))
    (setq sig
      (if (eq (getdatabase conname 'constructorkind) '|category|)
        (sublislis conargs |$TriangleVariableList| signature)
        (sublisFormal conargs signature))
      (htSaySaturn "\\begin{description}"
        (displayDomainOp| htPage |"constructor" conform conname sig t doc indexOrNil |dbSelectCon| (null exposeFlag) nil)
      (htSaySaturn "\\end{description}"))))
))

defun getConstructorDocumentation

[lassoc p?]
[getdatabase p1010]
[qcar p?]
[qcaar p?]
[qcadar p?]

— defun getConstructorDocumentation —

(defun getConstructorDocumentation (conname)
  (let (IT1)
    (setq IT1 (lassoc '|constructor| (getdatabase conname 'documentation)))
    (or
      (and (consp IT1) (consp (qcar IT1)) (null (qcaar IT1)) (consp (qcadar IT1))
       (qcadar IT1))
    ""))

defun dbSelectCon

[conPage p1343]
[opOf p?]
[htpProperty p1254]

— defun dbSelectCon —

(defun dbSelectCon (htPage which index)
  (declare (ignore which))
  (conPage (opOf (car (elt (htpProperty htPage '|cAlist|) index)))))

defun dbShowConditions

[htpProperty p1254]
[opOf p?]
[splitConTable p?]
[pluralize p?]
(defun |dbShowConditions| (htPage cAlist kind)
  (let (conform conname article whichever lt1 consNoPred consPred singular plural)
    (setq conform (|htpProperty| htPage '|conform|))
    (setq conname (|opOf| conform))
    (setq article (|htpProperty| htPage '|article|))
    (setq whichever (|htpProperty| htPage '|whichever|))
    (setq lt1 (\implitConTable\ cAlist))
    (setq consNoPred (car lt1))
    (setq consPred (cdr lt1))
    (setq singular (list kind " is")
    (setq plural (list (\pluralize (\stringimage kind)) " are"))
    (|dbSayItems| (\#\ consNoPred) singular plural " unconditional")
    (|htSaySaturn| "\\")
    (|bcConPredTable| consNoPred conname)
    (|htSayHrule|)
    (|dbSayItems| (\#\ consPred) singular plural " conditional")
    (|htSaySaturn| "\\")
    (|bcConPredTable| consPred conname))))

----

defun dbConsHeading

[|htpProperty| p1254]
[length p??]
[remdup p??]
[\stringimage p??]
[form2HtString p??]
[\capitalize p??]
[\pluralize p??]
[member p1048]
[\\nequal p??]
[\$\exposedOnlyIfTrue p??]

----
(defun |dbConsHeading| (htPage conlist view kind)
  (let (thing place count rank modifier exposureWord firstWord prefix
        placepart connective heading)
    (declare (special |$exposedOnlyIfTrue|))
    (setq thing (or (and htPage (|htpProperty| htPage '|thing|)) "constructor"))
    (setq place
      (when htPage
        (or (|htpProperty| htPage 'domname) (|htpProperty| htPage 'conform))))
    (setq count (|#| (remdup conlist)))
    (cond
      ((string= thing "benefactor")
       (list (stringimage count) " Constructors Used by "
         (|form2HtString| place nil t)))
      (t
       (setq modifier
         (cond
          ((string= thing "argument")
           (setq rank (and htPage (|htpProperty| htPage 'rank)))
           (list " Possible " rank " "))
          ((eq kind '|constructor|)
           (list " "))
          (t
           (cons " " (|capitalize| (stringimage kind)))))
       (setq exposureWord (when |$exposedOnlyIfTrue| '(" Exposed "))
       (setq prefix
         (cond
          ((eql count 1)
           (cons (stringimage count)
             (append modifier (list (|capitalize| thing))))))
          (t
           (setq firstWord (if (eql count 0) "No " (stringimage count)))
           (cons firstWord
             (append exposureWord
              (append modifier
               (list (|capitalize| (|pluralize| thing))))))))
       (setq placepart
         (when place (list " of \{\em " (|form2HtString| place nil t) ""))
       (setq heading (append prefix placepart))
       (setq connective
         (if (|member| view '(|abbrs| |files| |kinds|)) " as " " with ")
         (cond
          ((and (nequal count 0)
            (|member| view '(|abbrs| |files| |parameters| |conditions|)))
           (setq heading
             (append heading
               (list " viewed" connective "\{\em " (stringimage view) "}")))))
       heading))))
  )
defun dbShowConstructorLines

[getConstructorForm p??]  
[intern p??]  
[dbName p??]  
[dbShowCons1 p1389]  
[listSort p??]  
[function p??]  
[glesseqp p??]

— defun dbShowConstructorLines —

(defun |dbShowConstructorLines| (lines)
 (let (cAlist)
  (setq cAlist
   (loop for line in lines
      collect (cons (|getConstructorForm| (|intern| (|dbName| line))) t)))
   (|dbShowCons1| nil (|listSort| (|function| glesseqp) cAlist) '|names|)))

——

defun bcUnixTable

[htSay p??]  
[htBeginTable p??]  
[htSaySaturn p??]  
[namestring p1040]  
[findfile p??]  
[stringimage p??]  
[htMakePage p1263]  
[htEndTable p??]

— defun bcUnixTable —

(defun |bcUnixTable| (u)
 (let (firstTime filename)
  (|htSay|  
  (\newline")
  (|htBeginTable|)
  (setq firstTime t)
  (loop for x in u do
     (if firstTime (setq firstTime nil) (|htSaySaturn| "&"))
     (|htSay|  
     (setq filename (namestring ($findfile (stringimage x) "SPAD")))
     (|htMakePage|)
     (list
      (list '|text|  "\unixcommand\{" (pathname-name x)  

```
Special Code for Union, Mapping, and Record


defun dbSpecialDescription

(let (conform heading page)
 (declare (special $conformsAreDomains))
 (setq conform (getConstructorForm conname))
 (setq heading
 (list "Description of Domain \{sf " (form2HtString conform) "}")
 (setq page (htInitPage heading nil))
 (htpSetProperty page 'conname conname)
 (setq $conformsAreDomains nil)
 (dbShowConsDoc1 page conform nil)
 (htShowPage)))


defun dbSpecialOperations

(htInitPage p1262)
 (getConstructorForm p??)
 (dbSpecialExpandIfNecessary p1400)
 (get1 p1050)
 (form2HtString p??)
 (htpSetProperty p1254)
 (dbShowOp1 p??)
--- defun dbSpecialOperations ---

(defun |dbSpecialOperations| (conname)
  (let (page conform opAlist fromHeading)
    (setq page (|htInitPage| nil nil))
    (setq conform (|getConstructorForm| conname))
    (setq opAlist
      (|dbSpecialExpandIfNecessary| conform
       (cdr (getl conname '|documentation|))))
    (setq fromHeading (list " from domain {\sf " (|form2HtString| conform) "}")
    (|htpSetProperty| page '|fromHeading| fromHeading)
    (|htpSetProperty| page '|conform| conform)
    (|htpSetProperty| page '|opAlist| opAlist)
    (|htpSetProperty| page '|noUsage| t)
    (|htpSetProperty| page '|condition?| '|no!|)
    (|dbShowOp1| page opAlist "operation" '|names|)))

---

defun dbSpecialExports

[getConstructorForm p??]
[htInitPage p1262]
[form2HtString p??]
[dbSpecialExpandIfNecessary p1400]
[getl p1050]
[kePageDisplay p1354]
[htShowPage p1263]

--- defun dbSpecialExports ---

(defun |dbSpecialExports| (conname)
  (let (conform page opAlist)
    (setq conform (|getConstructorForm| conname))
    (setq page
      (|htInitPage| (list "Exports of {\sf " (|form2HtString| conform) "}")
    (setq opAlist
      (|dbSpecialExpandIfNecessary| conform
       (cdr (getl conname '|documentation|))))
    (|kePageDisplay| page "operation" opAlist)
    (|htShowPage|)))

_____
defun dbSpecialExpandIfNecessary

(defun dbSpecialExpandIfNecessary (conform opAlist)
  (if (and (consp opAlist) (consp (qcar opAlist)) (consp (qcdar opAlist))
       (consp (qcadar opAlist)) (cdr (qcdr (qcadar opAlist))))
   opAlist
   (dolist (item opAlist)
     (dolist (pair (cdr item))
       (rplacd pair (list t conform t (second pair)))))
   opAlist)

— initvars —

(defvar message1 (concatenate 'string
"{\sf Record(a:A,b:B)} is used to create the class of pairs of objects 
"made up of a value of type {\em A} selected by the symbol {\em a} and " 
a value of type {\em B} selected by the symbol {\em b}. 
"In general, the {\sf Record} constructor can take any number of arguments 
"and thus can be used to create aggregates of heterogeneous components of " 
"arbitrary size selectable by name. " 
"{\sf Record} is a primitive domain of Axiom which cannot be " 
"defined in the Axiom language."))

— postvars —

(eval-when (eval load)
 (put '|Record| '|documentation|
  (subst message1 'message
   '((|constructor| (nil message))
     (= (((|Boolean|) $ $) 
      "$\spad{r = s} tests for equality of two records \spad{r} and \spad{s}"))
     (|coerce| (((|OutputForm|) $) 
       "$\spad{\text{coerce(r)}} returns an representation of \spad{r} as an output form"))
    ((List (|Any|))))))
CONSTRUCTOR PAGE MENU

|elt| ((A $ "a")
,concatenate 'string
"\spad{coerce(u)}, where \spad{u} is the list \spad{[x,y]} for \spad{x} "
of type \spad{A} and \spad{y} of type \spad{B}, returns the record "
\spad{[a:x,b:y]}"))

|setelt| ((A $ "a" A)
,concatenate 'string
"\spad{r . a} returns the value stored in record \spad{r} under "
"selector \spad{a}."))

|setelt| ((B $ "b")
,concatenate 'string
"\spad{r . b} returns the value stored in record \spad{r} "
"under selector \spad{b}."))

|setelt| ((A $ "a" A)
,concatenate 'string
"\spad{r . a := x} destructively replaces the value stored in "
"record \spad{r} under selector \spad{a} by the value of \spad{x}. "
"Error: if \spad{r} has not been previously assigned a value."))

|setelt| ((B $ "b" B)
,concatenate 'string
"\spad{r . b := y} destructively replaces the value stored in "
"record \spad{r} under selector \spad{b} by the value of \spad{y}. "
"Error: if \spad{r} has not been previously assigned a value."))

:test #equal))

— initvars —

(defvar message2 (concatenate 'string
"\{\sf Union(A,B)\} denotes the class of objects which are which are either "
"members of domain \{A\} or of domain \{B\}. The \{Union\} "
"constructor can take any number of arguments. "
"For an alternate form of \{Union\} with "tags", see "
\downlink{Union(a:A,b:B)}{DomainUnion}. \{Union\} is a primitive "
"domain of Axiom which cannot be defined in the Axiom language."))

— postvars —

(eval-when (eval load)
(put '|UntaggedUnion| '|documentation|
(subst message2 'message
'(''(|constructor| (nil message))
 (= ((((|Boolean|) $ $))
 ,concatenate 'string
"\{\sf Union(A,B)\} denotes the class of objects which are which are either "
"members of domain \{A\} or of domain \{B\}. The \{Union\} "
"constructor can take any number of arguments. "
"For an alternate form of \{Union\} with "tags", see "
\downlink{Union(a:A,b:B)}{DomainUnion}. \{Union\} is a primitive "
"domain of Axiom which cannot be defined in the Axiom language."))

)
"\spad{u = v} tests if two objects of the union are equal, "
"that is, u and v are hold objects of same branch which are equal."
))})
  ((|case| (((|Boolean|) $ "A")
    , (concatenate 'string
      "\spad{u case A} tests if \spad{u} is of the type \spad{A} "
      "branch of the union."))
    , (concatenate 'string
      "\spad{coerce(u)} returns \spad{x} of type \spad{A} if "
      "\spad{x} is of the \spad{A} branch of the union."))
  )
  ((B $)
    , (concatenate 'string
      "\spad{coerce(u)} returns \spad{x} of type \spad{B} if "
      "\spad{x} is of the \spad{B} branch of the union."))
  )
  )
  )
  )
  )
  )
  )

— initvars —

(defvar message3 (concatenate 'string
  "{{\sf Union(a:A,b:B)}} denotes the class of objects which are either "
  "members of domain {{\em A}} or of domain {{\em B}}. "
  "The symbols {{\em a}} and {{\em b}} are called "tags" and are used to "
  "identify the two "branches" of the union. "
  "The {{\sf Union}} constructor can take any number of arguments and has an "
  "alternate form without {{\em tags}} "
  "(see {{\downlink{Union(A,B)}{UntaggedUnion}}}). "
  "This tagged {{\sf Union}} type is necessary, for example, to disambiguate "
  "two branches of a union where {{\em A}} and {{\em B}} denote the same type. "
  "{{\sf Union}} is a primitive domain of Axiom which cannot be "
  "defined in the Axiom language."))


75.6. CONSTRUCTOR PAGE MENU

-- postvars --

(eval-when (eval load)
(put 'Union 'documentation|
 (subst message3 'message
 '(((|constructor| (NIL MESSAGE)))
 (= (((|Boolean|) $ $)
 ,(concatenate 'string
 "\spad{u = v} tests if two objects of the union are equal, that "
 "is, \spad{u} and \spad{v} are objects of same branch which are equal."))
 (|case| (((|Boolean|) $ "A")
 "\spad{u case a} tests if \spad{u} is of branch \spad{a} of the union.")
 (((|Boolean|) $ "B")
 "\spad{u case b} tests if \spad{u} is of branch \spad{b} of the union."))
 (|coerce| ((A $)
 ,(concatenate 'string
 "\spad{coerce(u)} returns \spad{x} of type \spad{A} if "
 "\spad{x} is of branch \spad{a} of the union. "
 "Error: if \spad{u} is of branch \spad{b} of the union."))
 ((B $)
 ,(concatenate 'string
 "\spad{coerce(u)} returns \spad{x} of type \spad{B} if "
 "\spad{x} is of branch \spad{b} branch of the union. "
 "Error: if \spad{u} is of the \spad{a} branch of the union."))
 (($ A)
 ,(concatenate 'string
 "\spad{coerce(x)}, where \spad{x} has type \spad{A}, returns "
 "\spad{x} as a union type."))
 (($ B)
 ,(concatenate 'string
 "\spad{coerce(y)}, where \spad{y} has type \spad{B}, returns "
 "\spad{y} as a union type."))))
 :test #'equal))

---

-- initvars --

(defvar message4 (concatenate 'string
 "{\sf Mapping(T,S,...)} denotes the class of objects which are mappings from "
 "a source domain {{\em S,...}} into a target domain {{\em T}}. The "
 "{\sf Mapping} constructor can take any number of arguments."
 "All but the first argument is regarded as part of a source tuple for the "
 "mapping. For example, {{\sf Mapping(T,A,B)}} denotes the class of mappings "
 "from {{\em (A,B)}} into {{\em T}}. "
)
"\{sf Mapping\} is a primitive domain of Axiom which cannot be defined in "
"the Axiom language.\}"

— postvars —

(eval-when (eval load)
 (put '|Mapping| '|documentation|
  (subst message4 'message
   '((constructor| (NIL MESSAGE))
     (= (((Boolean) $ $)
         "\spad{u = v} tests if mapping objects are equal."))))
  :test #'equal))

— initvars —

(defvar message5 (concatenate 'string
  "\{em Enumeration(a1, a2 ,..., aN)\} creates an object which is exactly one "
  "of the N symbols \{em a1\}, \{em a2\}, ..., or \{em aN\}, N > 0. "
  " The \{em Enumeration\} can constructor can take any number of symbols as "
  "arguments.\}"

— postvars —

(eval-when (eval load)
 (put '|Enumeration| '|documentation|
  (subst message5 'message
   '((constructor| (nil message))
     (= (((Boolean) $ $)
         "\spad{e = f} tests for equality of two enumerations \spad{e} "
         "and \spad{f}\})))
     (= (((Boolean) $ $)
         "\spad{e ^= f} tests that two enumerations \spad{e} and "
         "\spad{f} are not equal\}))
     (coerce| (((OutputForm) $)
         "\spad{\{ coerced \} \{OutputForm\} $)"
     ,(concatenate 'string
     "\spad{e = f} tests for equality of two enumerations \spad{e} "
     "and \spad{f}\})))
     (= (((Boolean) $ $)
         "\spad{e ^= f} tests that two enumerations \spad{e} and "
         "\spad{f} are not equal\})))
     (coerce| (((OutputForm) $)
         "\spad{\{ coerced \} \{OutputForm\} $)"
     ,(concatenate 'string
     "\spad{e = f} tests for equality of two enumerations \spad{e} "
     "and \spad{f}\})))
     (= (((Boolean) $ $)
         "\spad{e ^= f} tests that two enumerations \spad{e} and "
         "\spad{f} are not equal\})))
     (coerce| (((OutputForm) $)
\spad{coerce(e)} returns a representation of enumeration \\spad{r} as an output form\\)
\((\ ($ \ (\{\Symbol\}))
  ,\concatenate \ 'string
\(\spad{coerce(s)}\) converts a symbol \spad{\{s\}} into an "
"enumeration which has \spad{\{s\}} as a member symbol"))))
:test #\equal))

---

defun mkConArgSublis

[\pname \ p\1045]
[\maxindex \ p??]
[\digitp \ p\1045]
[\digits2Names \ p\1405]

---

defun mkConArgSublis —

(defun |mkConArgSublis| (args)
  (loop for arg in args
    when
      (and
        (setq s (pname arg))
        (some #'identity
          (loop for i from 0 to (maxindex s)
            collect (digitp (elt s i)))))
      collect (cons arg (intern (\digits2Names\ (pname arg)))))))

---

This is necessary since arguments of conforms CANNOT have digits in TechExplorer. Since Saturn is gone we can remove it.

defun digits2Names

[\digit-char-p \ p??]
[\concat \ p\1047]

---

defun digits2Names —

(defun |digits2Names| (s)
  (let (str c n segment)
    (setq str "")
    (for i from 0 to (maxindex s) do

---
(setq c (elt s i))
(setq segment
  (cond
    ((setq n (digit-char-p c))
     (elt
      ('"Zero" "One" "Two" "Three" "Four" "Five" "Six" "Seven" "Eight" "Nine")
       n))
    (t c)))
  (concat str segment))
str))

defun lefts

[hascategory-hash p??]
— defun lefts —

(defun |lefts| (u)
  (let (keys)
    (setq keys (hkeys *hascategory-hash*))
    (loop for x in keys when (equal (cdr x) u) collect x)))

——

Build Library Database (libdb.text,...)

defun dbMkForm

— defun dbMkForm —

(defun |dbMkForm| (x)
  (or (and (atom x) (cons x nil)) x))

——

defun libConstructorSig

[getdatabase p1010]
[take p??]
--- defun libConstructorSig ---

(defun libConstructorSig (arg)
  (labels ((fn (x)
                (cond
                  ((atom x) x)
                  ((and (consp x) (eq (qcar x) '|Join|) (consp (qcdr x)))
                   (list '|Join| (fn (qcadr x)) '|etc|))
                  ((and (consp x) (eq (qcar x) 'category))
                   '|etc|)
                  (t
                   (loop for y in x collect (fn y))))
            (g (x u i)
              "does x appear in any but i-th element of u?"
              (some #'identity
                (loop for y in u for j from 1
                    when (not (= i j))
                    collect (contained x y))))
            (let (conname argl formals keys sig sigpart)
              (declare (special |$TriangleVariableList|))
              (setq conname (car arg))
              (setq argl (cdr arg))
              (setq sig (cdar (getdatabase conname 'constructormodemap)))
              (setq formals (take (#\ argl) |$FormalMapVariableList|))
              (setq sig (sublislis formals |$TriangleVariableList| sig))
              (setq keys
                (loop for f in formals for i from 1
                  collect (g f sig i)))
              (setq sig
                (fn (sublislis argl |$FormalMapVariableList| sig)))
              (setq sig (cons (car sig)
                  (loop for a in argl for s in (cdr sig) for k in keys
                    collect (if k (list #\ a s) s)))
                (unless (|ncParseFromString| sigpart)
                  (|sayBrightly| (list "Won't parse: " sigpart)))
              sigpart)))

---
Chapter 76

The Interpreter

--- Interpreter ---

(setq *print-array* nil)
(setq *print-circle* nil)
(setq *print-pretty* nil)

(in-package "BOOT")
\getchunk{initvars}

;;;; level 0 macros

\getchunk{defmacro bit-to-truth 0}
\getchunk{defmacro bvec-elt 0}
\getchunk{defmacro idChar? 0}
\getchunk{defmacro identp 0}
\getchunk{defmacro qsabsval 0}
\getchunk{defmacro qsadd1 0}
\getchunk{defmacro qsdifference 0}
\getchunk{defmacro qsmax 0}
\getchunk{defmacro qsemblp 0}
\getchunk{defmacro qssub1 0}
\getchunk{defmacro qstimes 0}
\getchunk{defmacro qssub 0}
\getchunk{defmacro spadConstant 0}

;;;; above level 0 macros
\getchunk{defmacro ancolsU8}
\getchunk{defmacro ancolsU16}
\getchunk{defmacro ancolsU32}
\getchunk{defmacro anrowsU8}
\getchunk{defmacro anrowsU16}
\getchunk{defmacro anrowsU32}
\getchunk{defmacro aref2U8}
\getchunk{defmacro aref2U16}
\getchunk{defmacro aref2U32}
\getchunk{defmacro assq}
\getchunk{defmacro bvec-setelt}
\getchunk{defmacro bvec-size}
\getchunk{defmacro cdaref2}
\getchunk{defmacro cdelt}
\getchunk{defmacro cdlen}
\getchunk{defmacro cdancols}
\getchunk{defmacro cdanrows}
\getchunk{defmacro cdsetaref2}
\getchunk{defmacro cdsetelt}
\getchunk{defmacro danrows}
\getchunk{defmacro dancols}
\getchunk{defmacro daref2}
\getchunk{defmacro delt}
\getchunk{defmacro DFAdd}
\getchunk{defmacro DFAcosh}
\getchunk{defmacro DFAsin}
\getchunk{defmacro DFAsinh}
\getchunk{defmacro DFAtan}
\getchunk{defmacro DFAtan2}
\getchunk{defmacro DFAtanh}
\getchunk{defmacro DFCos}
\getchunk{defmacro DFCosh}
\getchunk{defmacro DFDivide}
\getchunk{defmacro DFExp}
\getchunk{defmacro DFExpt}
\getchunk{defmacro DFIntegerDivide}
\getchunk{defmacro DFIntegerExpt}
\getchunk{defmacro DFIntegerMultiply}
\getchunk{defmacro DFLessThan}
\getchunk{defmacro DFlowserve}
\getchunk{defmacro DFLog}
\getchunk{defmacro DFLogE}
\getchunk{defmacro DFMax}
\getchunk{defmacro DFMin}
\getchunk{defmacro DFMinusp}
\getchunk{defmacro DFMultiply}
\getchunk{defmacro DFSin}
\getchunk{defmacro DFSinh}
\getchunk{defmacro DFSqrt}
\getchunk{defmacro DFSubtract}
\getchunk{defmacro DFTan}
\getchunk{defmacro DFTanh}
\getchunk{defmacro DFUnaryMinus}
\getchunk{defmacro DFZerop}
\getchunk{defmacro dlen}
\getchunk{defmacro dsetaref2}
\getchunk{defmacro dsetelt}
\getchunk{defmacro eltU8}
\getchunk{defmacro eltU16}
\getchunk{defmacro eltU32}
\getchunk{defmacro funfind}
\getchunk{defmacro hget}
\getchunk{defmacro make-cdouble-matrix}
\getchunk{defmacro make-cdouble-vector}
\getchunk{defmacro make-double-matrix}
\getchunk{defmacro make-double-matrix1}
\getchunk{defmacro make-double-vector}
\getchunk{defmacro make-double-vector1}
\getchunk{defmacro makeMatrixU8}
\getchunk{defmacro makeMatrix1U8}
\getchunk{defmacro makeMatrixU16}
\getchunk{defmacro makeMatrix1U16}
\getchunk{defmacro makeMatrixU32}
\getchunk{defmacro makeMatrix1U32}
\getchunk{defmacro qsDot26432}
\getchunk{defmacro qsDot2Mod6432}
\getchunk{defmacro qsMod6432}
\getchunk{defmacro qsMulAdd6432}
\getchunk{defmacro qsMulAddMod6432}
\getchunk{defmacro qsMul6432}
\getchunk{defmacro qsMulMod32}
\getchunk{defmacro qvlenU8}
\getchunk{defmacro qvlenU16}
\getchunk{defmacro qvlenU32}
\getchunk{defmacro Rest}
\getchunk{defmacro startsId?}
\getchunk{defmacro setAref2U8}
\getchunk{defmacro setAref2U16}
\getchunk{defmacro setAref2U32}
\getchunk{defmacro seteltU8}
\getchunk{defmacro seteltU16}
\getchunk{defmacro seteltU32}
\getchunk{defmacro trapNumericErrors}
\getchunk{defmacro truth-to-bit}
\getchunk{defmacro while}
\getchunk{defmacro whileWithResult}

;;; layer 0 (all common lisp)
CHAPTER 76. THE INTERPRETER

\getchunk{defun acot 0}
\getchunk{defun acoth 0}
\getchunk{defun acsc 0}
\getchunk{defun acsch 0}
\getchunk{defun asec 0}
\getchunk{defun asech 0}
\getchunk{defun axiomVersion 0}
\getchunk{defun BooleanEquality 0}
\getchunk{defun bvec-and 0}
\getchunk{defun bvec-concat 0}
\getchunk{defun bvec-copy 0}
\getchunk{defun bvec-equal 0}
\getchunk{defun bvec-greater 0}
\getchunk{defun bvec-make-full 0}
\getchunk{defun bvec-nand 0}
\getchunk{defun bvec-nor 0}
\getchunk{defun bvec-not 0}
\getchunk{defun bvec-or 0}
\getchunk{defun bvec-xor 0}
\getchunk{defun cleanupLine 0}
\getchunk{defun clearMacroTable 0}
\getchunk{defun concat 0}
\getchunk{defun cot 0}
\getchunk{defun coth 0}
\getchunk{defun createCurrentInterpreterFrame 0}
\getchunk{defun credits 0}
\getchunk{defun csc 0}
\getchunk{defun csch 0}
\getchunk{defun Delay 0}
\getchunk{defun desiredMsg 0}
\getchunk{defun DirToString 0}
\getchunk{defun divide2 0}
\getchunk{defun dqAppend 0}
\getchunk{defun dqToList 0}
\getchunk{defun dqUnit 0}
\getchunk{defun emptyInterpreterFrame 0}
\getchunk{defun fin 0}
\getchunk{defun findFrameInRing 0}
\getchunk{defun flatten 0}
\getchunk{defun fnameExists? 0}
\getchunk{defun fnameName 0}
\getchunk{defun fnameReadable? 0}
\getchunk{defun fnameType 0}
\getchunk{defun frameExposureData 0}
\getchunk{defun frameHiFiAccess 0}
\getchunk{defun keyword 0}
\getchunk{defun keyword? 0}

\getchunk{defun lfcomment 0}
\getchunk{defun lferror 0}
\getchunk{defun lffloat 0}
\getchunk{defun lfid 0}
\getchunk{defun lfinteger 0}
\getchunk{defun lfnegcomment 0}
\getchunk{defun lfrinteger 0}
\getchunk{defun lfspaces 0}
\getchunk{defun lfstring 0}
\getchunk{defun lnCreate 0}
\getchunk{defun lnExtraBlanks 0}
\getchunk{defun lnFileName? 0}
\getchunk{defun lnGlobalNum 0}
\getchunk{defun lnImmediate? 0}
\getchunk{defun lnLocalNum 0}
\getchunk{defun lnPlaceOfOrigin 0}
\getchunk{defun lnSetGlobalNum 0}
\getchunk{defun lnString 0}

\getchunk{defun mac0Define 0}
\getchunk{defun mac0InfiniteExpansion,name 0}
\getchunk{defun make-absolute-filename 0}
\getchunk{defun makeByteWordVec2 0}
\getchunk{defun makeInitialModemapFrame 0}
\getchunk{defun manexp 0}
\getchunk{defun member 0}
\getchunk{defun monitor-add 0}
\getchunk{defun monitor-apropos 0}
\getchunk{defun monitor-autoload 0}
\getchunk{defun monitor-checkpoint 0}
\getchunk{defun monitor-decr 0}
\getchunk{defun monitor-delete 0}
\getchunk{defun monitor-dirname 0}
\getchunk{defun monitor-disable 0}
\getchunk{defun monitor-enable 0}
\getchunk{defun monitor-end 0}
\getchunk{defun monitor-exposedp 0}
\getchunk{defun monitor-file 0}
\getchunk{defun monitor-help 0}
\getchunk{defun monitor-incr 0}
\getchunk{defun monitor-info 0}
\getchunk{defun monitor-inittable 0}
\getchunk{defun monitor-libname 0}
\getchunk{defun monitor-nrlib 0}
\getchunk{defun monitor-parse 0}
\getchunk{defun monitor-percent 0}
\getchunk{defun monitor-readinterp 0}
\getchunk{defun monitor-report 0}
\getchunk{defun monitor-reset 0}
\getchunk{defun monitor-restore 0}
\getchunk{defun monitor-results 0}
\getchunk{defun monitor-spadfile 0}
\getchunk{defun monitor-tested 0}
\getchunk{defun monitor-untested 0}
\getchunk{defun monitor-write 0}
\getchunk{defun ncError 0}
\getchunk{defun ncloopEscaped 0}
\getchunk{defun ncloopPrefix? 0}
\getchunk{defun ncloopPrintLines 0}
\getchunk{defun nonBlank 0}
\getchunk{defun npAnyNo 0}
\getchunk{defun npboot 0}
\getchunk{defun npEqPeek 0}
\getchunk{defun npEqPeek 0}
\getchunk{defun npPush 0}
\getchunk{defun npPop1 0}
\getchunk{defun npPop2 0}
\getchunk{defun npPop3 0}
\getchunk{defun npPush 0}
\getchunk{defun opTran 0}
\getchunk{defun pfAndLeft 0}
\getchunk{defun pfAndRight 0}
\getchunk{defun pfAppend 0}
\getchunk{defun pfApplicationArg 0}
\getchunk{defun pfApplicationOp 0}
\getchunk{defun pfAssignLhsItems 0}
\getchunk{defun pf0AssignLhsItems 0}
\getchunk{defun pfAssignRhs 0}
\getchunk{defun pfBreakFrom 0}
\getchunk{defun pfCoercetoExpr 0}
\getchunk{defun pfCoercetoType 0}
\getchunk{defun pfCollectBody 0}
\getchunk{defun pfCollectIterators 0}
\getchunk{defun pfDefinitionLhsItems 0}
\getchunk{defun pfDefinitionRhs 0}
\getchunk{defun pfDoBody 0}
\getchunk{defun pfExitCond 0}
\getchunk{defun pfExitExpr 0}
\getchunk{defun pfFirst 0}
\getchunk{defun pfFreeItems 0}
\getchunk{defun pfForInLhs 0}
\getchunk{defun pfForInWhole 0}
\getchunk{defun pfFromdomDomain 0}
\getchunk{defun pfFromdomWhat 0}
\getchunk{defun pfIfCond 0}
CHAPTER 76. THE INTERPRETER

\getchunk{defun pfIfElse 0}
\getchunk{defun pfIfThen 0}
\getchunk{defun pfLambdaArgs 0}
\getchunk{defun pfLambdaBody 0}
\getchunk{defun pfLambdaRets 0}
\getchunk{defun pfLiteral? 0}
\getchunk{defun pfLocalItems 0}
\getchunk{defun pfLoopIterators 0}
\getchunk{defun pfMacroLhs 0}
\getchunk{defun pfMacroRhs 0}
\getchunk{defun pfMLambdaArgs 0}
\getchunk{defun pfMLambdaBody 0}
\getchunk{defun pfNotArg 0}
\getchunk{defun pfNovalueExpr 0}
\getchunk{defun pfOrLeft 0}
\getchunk{defun pfOrRight 0}
\getchunk{defun pfParts 0}
\getchunk{defun pfPile 0}
\getchunk{defun pfPretendExpr 0}
\getchunk{defun pfPretendType 0}
\getchunk{defun pfRestrictExpr 0}
\getchunk{defun pfRestrictType 0}
\getchunk{defun pfReturnExpr 0}
\getchunk{defun pfRuleLhsItems 0}
\getchunk{defun pfRuleRhs 0}
\getchunk{defun pfSecond 0}
\getchunk{defun pfSequenceArgs 0}
\getchunk{defun pfSuchthatCond 0}
\getchunk{defun pfTaggedExpr 0}
\getchunk{defun pfTaggedTag 0}
\getchunk{defun pfTree 0}
\getchunk{defun pfTypedId 0}
\getchunk{defun pfTypedType 0}
\getchunk{defun pfTupleParts 0}
\getchunk{defun pfWhereContext 0}
\getchunk{defun pfWhereExpr 0}
\getchunk{defun pfWhileCond 0}
\getchunk{defun pmDontQuote? 0}
\getchunk{defun poCharPosn 0}
\getchunk{defun poGetLineObject 0}
\getchunk{defun poNoPos? 0}
\getchunk{defun poNoPosition 0}
\getchunk{defun poNoPosition? 0}
\getchunk{defun printAsTeX 0}
\getchunk{defun pname 0}
\getchunk{defun qenum 0}
\getchunk{defun qquotient 0}
\getchunk{defun qremainder 0}
\getchunk{defun quotient2 0}
(defun random 0)
(defun rdigit? 0)
(defun reclaim 0)
(defun remainder2 0)
(defun remLine 0)
(defun rep 0)
(defun resetStackLimits 0)
(defun sameUnionBranch 0)
(defun satisfiesUserLevel 0)
(defun scanCloser? 0)
(defun sec 0)
(defun sech 0)
(defun setCurrentLine 0)
(defun setMsgPrefix 0)
(defun setMsgText 0)
(defun set-restart-hook 0)
(defun showMsgPos? 0)
(defun StreamNull 0)
(defun stripLisp 0)
(defun stripSpaces 0)
(defun theid 0)
(defun thefname 0)
(defun theorigin 0)
(defun tokPart 0)
(defun To 0)
(defun Top? 0)
(defun trademark 0)
(defun zeroOneTran 0)
;;; above level 0

(defun abbQuery)
(defun abbreviations)
(defun abbreviationsSpad2Cmd)
(defun addBinding)
(defun addBindingInteractive)
(defun addInputLibrary)
(defun addNewInterpreterFrame)
(defun addoperations)
(defun addTraceItem)
(defun algCoerceInteractive)
(defun allConstructors)
(defun allOperations)
(defun alqlGetOrigin)
(defun alqlGetParams)
(defun alqlGetKindString)
CHAPTER 76. THE INTERPRETER

(defun alreadyOpened?)
(defun apropos)
(defun assertCond)
(defun augmentHasArgs)
(defun augmentTraceNames)

(defun basicLookup)
(defun basicLookupCheckDefaults)
(defun basicStringize)
(defun bcComplexLimit)
(defun bcComplexLimitGen)
(defun bcCreateVariableString)
(defun bcDefiniteIntegrate)
(defun bcDefiniteIntegrateGen)
(defun bcDifferentiate)
(defun bcDifferentiateGen)
(defun bcDraw)
(defun bcDrawIt)
(defun bcDrawIt2)
(defun bcDraw2Dfun)
(defun bcDraw2DfunGen)
(defun bcDraw2Dpar)
(defun bcDraw2DparGen)
(defun bcDraw2DSolve)
(defun bcDraw2DSolveGen)
(defun bcDraw3Dfun)
(defun bcDraw3DfunGen)
(defun bcDraw3Dpar)
(defun bcDraw3DparGen)
(defun bcDraw3Dpar1)
(defun bcDraw3Dpar1Gen)
(defun bcError)
(defun bcFindString)
(defun bcFinish)
(defun bcGen)
(defun bcGenEquations)
(defun bcGenExplicitMatrix)
(defun bcHt)
(defun bchtMakeButton)
(defun bcIndefiniteIntegrate)
(defun bcIndefiniteIntegrateGen)
(defun bcInputEquations)
(defun bcInputEquationsEnd)
(defun bcInputExplicitMatrix)
(defun bcInputMatrixByFormula)
(defun bcInputMatrixByFormulaGen)
(defun bcInputSolveInfo)
(defun bcIssueHt)
(defun bcLaurentSeries)
(defun bcLaurentSeriesGen)
(defun bcLimit)
(defun bcLinearExtractMatrix)
(defun bcLinearMatrixGen)
(defun bcLinearSolve)
(defun bcLinearSolveEqns)
(defun bcLinearSolveEqns1)
(defun bcLinearSolveEqnsGen)
(defun bcLinearSolveMatrix)
(defun bcLinearSolveMatrix1)
(defun bcLinearSolveMatrixHomo)
(defun bcLinearSolveMatrixInhomo)
(defun bcLinearSolveMatrixInhomoGen)
(defun bcLinearSolveMatrix)
(defun bcMatrixGen)
(defun bcMakeEquations)
(defun bcMakeLinearEquations)
(defun bcMakeUnknowns)
(defun bcMkFunction)
(defun bcNotReady)
(defun bcOptional)
(defun bcProduct)
(defun bcProductGen)
(defun bcPuiseuxGen)
(defun bcPuiseuxSeries)
(defun bcPuiseuxSeriesGen)
(defun bcReadMatrix)
(defun bcRealLimit)
(defun bcRealLimitGen)
(defun bcRealLimitGen1)
(defun bcSadFaces)
(defun bcSeries)
(defun bcSeriesByFormula)
(defun bcSeriesByFormulaGen)
(defun bcSeriesExpansion)
(defun bcSeriesExpansionGen)
(defun bcSeriesGen)
(defun bcSolve)
(defun bcSolveEquations)
(defun bcSolveEquationsNumerically)
(defun bcSolveNumerically1)
(defun bcSolveSingle)
(defun bcString2HyString)
(defun bcString2HyString2)
(defun bcString2WordList)
(defun bcSystemSolveEqns1)
(defun bcSum)
(defun bcSumGen)
(defun bcSystemSolve)
(defun bcTaylorSeries)
(defun bcTaylorSeriesGen)
(defun bcUnixTable)
\getchunk{defun bcVectorGen}
\getchunk{defun bcvspace}
\getchunk{defun bcwords2liststring}
\getchunk{defun beforeAfter}
\getchunk{defun bracketString}
\getchunk{defun break}
\getchunk{defun breaklet}
\getchunk{defun brightprint}
\getchunk{defun brightprint-0}
\getchunk{defun browse}
\getchunk{defun browseOpen}
\getchunk{defun buttonNames}

\getchunk{defun cacheKeyedMsg}
\getchunk{defun canFuncall?}
\getchunk{defun categoryOpen}
\getchunk{defun changeHistListLen}
\getchunk{defun changeToNamedInterpreterFrame}
\getchunk{defun charDigitVal}
\getchunk{defun checkCondition}
\getchunk{defun chkAllNonNegativeInteger}
\getchunk{defun chkDirectory}
\getchunk{defun chkNamedList}
\getchunk{defun chkNonNegativeInteger}
\getchunk{defun chkOutputFileName}
\getchunk{defun chkPosInteger}
\getchunk{defun chkRange}
\getchunk{defun cleanline}
\getchunk{defun clear}
\getchunk{defun clearCmdAll}
\getchunk{defun clearCmdCompletely}
\getchunk{defun clearCmdExcept}
\getchunk{defun clearCmdParts}
\getchunk{defun clearCmdSortedCaches}
\getchunk{defun clearFrame}
\getchunk{defun clearParserMacro}
\getchunk{defun clearSpad2Cmd}
\getchunk{defun close}
\getchunk{defun closeInterpreterFrame}
\getchunk{defun cmpnote}
\getchunk{defun coerceSpadArgs2E}
\getchunk{defun coerceSpadFunValue2E}
\getchunk{defun coercerTraceArgs2E}
\getchunk{defun coerceTraceFunValue2E}
\getchunk{defun commandAmbiguityError}
\getchunk{defun commandError}
\getchunk{defun commandErrorIfAmbiguous}
\getchunk{defun commandErrorMessage}
\getchunk{defun commandsForUserLevel}
\getchunk{defun commandUserLevelError}
\getchunk{defun compareposns}
\getchunk{defun compileBoot}
\getchunk{defun compiledLookup}
\getchunk{defun compiledLookupCheck}
\getchunk{defun computeDomainVariableAlist}
\getchunk{defun condErrorMsg}
\getchunk{defun conOpPage}
\getchunk{defun conOpPage1}
\getchunk{defun conPage}
\getchunk{defun conPageChoose}
\getchunk{defun conPageConEntry}
\getchunk{defun conPageFastPath}
\getchunk{defun constoken}
\getchunk{defun constructSubst}
\getchunk{defun containsVars}
\getchunk{defun containsVars1}
\getchunk{defun copyright}
\getchunk{defun countCache}
\getchunk{defun DaaseName}
\getchunk{defun dbAddChain}
\getchunk{defun dbAddChainDomain}
\getchunk{defun dbAddDocTable}
\getchunk{defun dbCompositeWithMap}
\getchunk{defun dbConsExposureMessage}
\getchunk{defun dbConsHeading}
\getchunk{defun dbConstructorDoc}
\getchunk{defun dbConstructorDoc,hn}
\getchunk{defun dbConstructorDoc,gn}
\getchunk{defun dbDocTable}
\getchunk{defun dbExtractUnderlyingDomain}
\getchunk{defun dbGetDocTable}
\getchunk{defun dbGetDocTable,gn}
\getchunk{defun dbGetDocTable,hn}
\getchunk{defun dbMkForm}
\getchunk{defun dbNonEmptyPattern}
\getchunk{defun dbSearchOrder}
\getchunk{defun dbSelectCon}
\getchunk{defun dbShowConditions}
\getchunk{defun dbShowCons}
\getchunk{defun dbShowCons1}
\getchunk{defun dbShowConsDoc}
\getchunk{defun dbShowConsDoc1}
\getchunk{defun dbShowConsKindsFilter}
\getchunk{defun dbShowConstructorLines}
\getchunk{defun dbSpecialDescription}
\getchunk{defun dbSpecialExpandIfNecessary}
\getchunk{defun dbSpecialExports}
\getchunk{defun dbSpecialOperations}
\getchunk{defun dbSubConform}
\texttt{\defun decideHowMuch}
\texttt{\defun defaultTargetFE}
\texttt{\defun defiostream}
\texttt{\defun deldatabase}
\texttt{\defun deleteFile}
\texttt{\defun describe}
\texttt{\defun describeFortPersistence}
\texttt{\defun describeInputLibraryArgs}
\texttt{\defun describeOutputLibraryArgs}
\texttt{\defun describeSetFortDir}
\texttt{\defun describeSetFortTmpDir}
\texttt{\defun describeSetFunctionsCache}
\texttt{\defun describeSetLinkerArgs}
\texttt{\defun describeSetNagHost}
\texttt{\defun describeSetOutputAlgebra}
\texttt{\defun describeSetOutputFormula}
\texttt{\defun describeSetOutputFortran}
\texttt{\defun describeSetOutputHtml}
\texttt{\defun describeSetOutputMathml}
\texttt{\defun describeSetOutputOpenMath}
\texttt{\defun describeSetOutputTex}
\texttt{\defun describeSetStreamsCalculate}
\texttt{\defun describeSpad2Cmd}
\texttt{\defun dewritify}
\texttt{\defun dewritify,dewritifyInner}
\texttt{\defun diffAlist}
\texttt{\defun digit?}
\texttt{\defun digitp}
\texttt{\defun digits2Names}
\texttt{\defun disableHist}
\texttt{\defun display}
\texttt{\defun displayCondition}
\texttt{\defun displayExposedConstructors}
\texttt{\defun displayExposedGroups}
\texttt{\defun displayFrameNames}
\texttt{\defun displayHiddenConstructors}
\texttt{\defun displayMacro}
\texttt{\defun displayMacros}
\texttt{\defun displayMode}
\texttt{\defun displayModemap}
\texttt{\defun displayOperations}
\texttt{\defun displayOperationsFromLispLib}
\texttt{\defun displayParserMacro}
\texttt{\defun displayProperties}
\texttt{\defun displayProperties,sayFunctionDeps}
\texttt{\defun displaySetOptionInformation}
\texttt{\defun displaySetVariableSettings}
\texttt{\defun displaySpad2Cmd}
\texttt{\defun displayType}
\texttt{\defun displayValue}
\getchunk{defun displayWorkspaceNames}
\getchunk{defun doDoitButton}
\getchunk{defun domainDescendantsOf}
\getchunk{defun domainToGenvar}
\getchunk{defun domArg}
\getchunk{defun domArg2}
\getchunk{defun doSystemCommand}
\getchunk{defun downlink}
\getchunk{defun downlinkSaturn}
\getchunk{defun dqConcat}
\getchunk{defun dropInputLibrary}
\getchunk{defun dumbTokenize}
\getchunk{defun edit}
\getchunk{defun editFile}
\getchunk{defun editSpad2Cmd}
\getchunk{defun Else?}
\getchunk{defun Elseif?}
\getchunk{defun enPile}
\getchunk{defun eofp}
\getchunk{defun eqpileTree}
\getchunk{defun enMsgCompare}
\getchunk{defun enMsgSep}
\getchunk{defun enMsgSort}
\getchunk{defun evalCategory}
\getchunk{defun evalDomain}
\getchunk{defun evaluateSignature}
\getchunk{defun evaluateType}
\getchunk{defun evaluateType1}
\getchunk{defun executeInterpreterCommand}
\getchunk{defun ExecuteInterpSystemCommand}
\getchunk{defun executeQuietCommand}
\getchunk{defun explainLinear}
\getchunk{defun fetchKeyedMsg}
\getchunk{defun fetchOutput}
\getchunk{defun fillerSpaces}
\getchunk{defun filterAndFormatConstructors}
\getchunk{defun filterListOfStrings}
\getchunk{defun filterListOfStringsWithFn}
\getchunk{defun finalExactRequest}
\getchunk{defun firstTokPosn}
\getchunk{defun fixObjectForPrinting}
\getchunk{defun flattenOperationAlist}
\getchunk{defun float2Sex}
\getchunk{defun fnameDirectory}
\getchunk{defun fnameMake}
\getchunk{defun fnameNew}
\getchunk{defun fnameWritable?}
\getchunk{defun frame}
\getchunk{defun frameEnvironment}
\getchunk{defun frameSpad2Cmd}
\getchunk{defun funfind, LAM}

\getchunk{defun gatherGlossLines}
\getchunk{defun genDomainTraceName}
\getchunk{defun gensymInt}
\getchunk{defun getAliasIfTracedMapParameter}
\getchunk{defun getAndEvalConstructorArgument}
\getchunk{defun getAndSay}
\getchunk{defun getBpiNameIfTracedMap}
\getchunk{defun getBrowseDatabase}
\getchunk{defun getConstructorDocumentation}
\getchunk{defun getdatabase}
\getchunk{defun getDependentsOfConstructor}
\getchunk{defun getDirectoryList}
\getchunk{defun getFirstWord}
\getchunk{defun getKeyedMsg}
\getchunk{defun getMapSig}
\getchunk{defun getMapSubNames}
\getchunk{defun getMsgCatAttr}
\getchunk{defun getMsgFTTag?}
\getchunk{defun getMsgInfoFromKey}
\getchunk{defun getMsgLitSym}
\getchunk{defun getMsgPos}
\getchunk{defun getMsgPos2}
\getchunk{defun getMsgToWhere}
\getchunk{defun getOplistForConstructorForm}
\getchunk{defun getOplistWithUniqueSignatures}
\getchunk{defun getOption}
\getchunk{defun getPosStL}
\getchunk{defun getPreviousMapSubNames}
\getchunk{defun getProplist}
\getchunk{defun getRefvU8}
\getchunk{defun getRefvU16}
\getchunk{defun getRefvU32}
\getchunk{defun getStFromMsg}
\getchunk{defun getSystemCommandLine}
\getchunk{defun getTraceOption}
\getchunk{defun getTraceOption, hn}
\getchunk{defun getTraceOptions}
\getchunk{defun getUsersOfConstructor}
\getchunk{defun getWorkspaceNames}

\getchunk{defun handleNoParseCommands}
\getchunk{defun handleParsedSystemCommands}
\getchunk{defun handleTokenizeSystemCommands}
\getchunk{defun hasAtt}
\getchunk{defun hasAttSig}
\getchunk{defun hasCatExpression}
(defun hasCate)
(defun hasCateSpecial)
(defun hasCateSpecialNew)
(defun hasCate1)
(defun hasCaty)
(defun hasCaty1)
(defun hashable)
(defun hasOption)
(defun hasPair)
(defun hasSig)
(defun hasSigAnd)
(defun hasSigOr)
(defun help)
(defun helpSpad2Cmd)
(defun histFileErase)
(defun histFileName)
(defun histInputFileName)
(defun history)
(defun historySpad2Cmd)
(defun hkeys)
(defun hput)
(defun htAddHeading)
(defun htAllOrNum)
(defun htBcLinks)
(defun htBcLispLinks)
(defun htBcRadioButtons)
(defun htCacheAddChoice)
(defun htCacheOne)
(defun htCacheSet)
(defun htCheckList)
(defun htCheck)
(defun htDoneButton)
(defun htDoNothing)
(defun htEscapeString)
(defun htFunctionSetLiteral)
(defun htGlossPage)
(defun htGlossSearch)
(defun htGloss)
(defun htGreekSearch)
(defun htInitPage)
(defun htInputStrings)
(defun htKill)
(defun htLispLinks)
(defun htLispMemoLinks)
(defun htMakeButton)
(defun htMakeDoitButton)
(defun htMakeDoneButton)
(defun htMakeErrorPage)
(defun htMakeInputList)
(defun htMakeLabel)
CHAPTER 76. THE INTERPRETER

\getchunk{defun htMakePage}
\getchunk{defun htMakePage1}
\getchunk{defun htMakePathKey, fn}
\getchunk{defun htMakePathKey}
\getchunk{defun htMakeTemplates, substLabel}
\getchunk{defun htMakeTemplates}
\getchunk{defun htMarkTree}
\getchunk{defun htMkName}
\getchunk{defun htpAddInputAreaProp}
\getchunk{defun htpAddToPageDescription}
\getchunk{defun htpButtonValue}
\getchunk{defun htpDestroyPage}
\getchunk{defun htpDomainConditions}
\getchunk{defun htpDomainPvarSubstList}
\getchunk{defun htpDomainVariableAlist}
\getchunk{defun htpInputAreaAlist}
\getchunk{defun htpLabelDefault}
\getchunk{defun htpLabelErrorMsg}
\getchunk{defun htpLabelFilteredInputString}
\getchunk{defun htpLabelInputString}
\getchunk{defun htpLabelType}
\getchunk{defun htpName}
\getchunk{defun htpPageDescription}
\getchunk{defun htpProperty}
\getchunk{defun htpPropertyList}
\getchunk{defun htpProcessBcButtons}
\getchunk{defun htpProcessBcStrings}
\getchunk{defun htpProcessDoitButton}
\getchunk{defun htpProcessDomainConditions}
\getchunk{defun htpProcessDoneButton}
\getchunk{defun htpProcessToggleButtons}
\getchunk{defun htpSetDomainConditions}
\getchunk{defun htpSetDomainPvarSubstList}
\getchunk{defun htpSetDomainVariableAlist}
\getchunk{defun htpSetInputAreaAlist}
\getchunk{defun htpSetLabelErrorMsg}
\getchunk{defun htpSetLabelInputString}
\getchunk{defun htpSetName}
\getchunk{defun htpSetPageDescription}
\getchunk{defun htpSetProperty}
\getchunk{defun htpSetRadioButtonAlist}
\getchunk{defun htpSetCache}
\getchunk{defun htpSetExpose}
\getchunk{defun htpSetFunCommandContinue}
\getchunk{defun htSetFunCommand}
\getchunk{defun htSetHistory}
\getchunk{defun htSetInputLibrary}
\getchunk{defun htSetInteger}
\getchunk{defun htSetLinkerArgs}
\getchunk{defun htSetLiterals}
\getchunk{defun htSetLiteral}
\getchunk{defun htSetNotAvailable}
\getchunk{defun htSetOutputCharacters}
\getchunk{defun htSetOutputLibrary}
\getchunk{defun htSetSystemVariableKind}
\getchunk{defun htSetSystemVariable}
\getchunk{defun htSetVars}
\getchunk{defun htSetVarDoneButton}
\getchunk{defun htShowCount}
\getchunk{defun htShowFunctionPageContinued}
\getchunk{defun htShowFunctionPage}
\getchunk{defun htShowIntegerPage}
\getchunk{defun htShowLiteralsPage}
\getchunk{defun htShowPage}
\getchunk{defun htShowPageNoScroll}
\getchunk{defun htShowSetPage}
\getchunk{defun htShowSetTreeValue}
\getchunk{defun htShowSetTree}
\getchunk{defun htStringPad}
\getchunk{defun htsv}
\getchunk{defun htSystemVariables,displayOptions}
\getchunk{defun htSystemVariables,fn}
\getchunk{defun htSystemVariables,functionTail}
\getchunk{defun htSystemVariables,gn}
\getchunk{defun htSystemVariables}
\getchunk{defun htTextSearch}
\getchunk{defun htTutorialSearch}
\getchunk{defun If?}
\getchunk{defun ifCond}
\getchunk{defun iht}
\getchunk{defun importFromFrame}
\getchunk{defun incAppend}
\getchunk{defun incAppend1}
\getchunk{defun incBiteOff}
\getchunk{defun incClassify}
\getchunk{defun incCommandTail}
\getchunk{defun incConsoleInput}
\getchunk{defun incFileInput}
\getchunk{defun incFileName}
\getchunk{defun incigen}
\getchunk{defun incigen1}
\getchunk{defun incIfname}
\getchunk{defun incLine}
CHAPTER 76. THE INTERPRETER

\getchunk{defun incLine1}
\getchunk{defun inclmsgCannotRead}
\getchunk{defun inclmsgFileCycle}
\getchunk{defun inclmsgPrematureFin}
\getchunk{defun include}
\getchunk{defun include1}
\getchunk{defun inclmsgConActive}
\getchunk{defun inclmsgConStill}
\getchunk{defun inclmsgIfSyntax}
\getchunk{defun inclmsgNoSuchFile}
\getchunk{defun inclmsgSay}
\getchunk{defun incNConsoles}
\getchunk{defun incRenumber}
\getchunk{defun incRenumberItem}
\getchunk{defun incRenumberLine}
\getchunk{defun incRgen}
\getchunk{defun incRgen1}
\getchunk{defun incStream}
\getchunk{defun incString}
\getchunk{defun incZip}
\getchunk{defun incZip1}
\getchunk{defun init-boot/spad-reader}
\getchunk{defun initHist}
\getchunk{defun initHistList}
\getchunk{defun initial-getdatabase}
\getchunk{defun initializeInterpreterFrameRing}
\getchunk{defun initializeSetVariables}
\getchunk{defun initImPr}
\getchunk{defun initroot}
\getchunk{defun initToWhere}
\getchunk{defun insertAlist}
\getchunk{defun insertpile}
\getchunk{defun InterpExecuteSpadSystemCommand}
\getchunk{defun interpFunctionDepAlists}
\getchunk{defun interpOpen}
\getchunk{defun interpret}
\getchunk{defun interpret1}
\getchunk{defun interpret2}
\getchunk{defun interpretTopLevel}
\getchunk{defun interpretPform}
\getchunk{defun intloop}
\getchunk{defun intloopEchoParse}
\getchunk{defun intloopInclude}
\getchunk{defun intloopInclude0}
\getchunk{defun intnplisp}
\getchunk{defun intntplisp}
\getchunk{defun intloopProcess}
\getchunk{defun intloopProcessString}
\getchunk{defun intloopReadConsole}
\getchunk{defun intloopSpadProcess}
\getchunk{defun intloopSpadProcess, interp}
\getchunk{defun intProcessSynonyms}
\getchunk{defun ioclear}
\getchunk{defun iostat}
\getchunk{defun isDomainOrPackage}
\getchunk{defun isDomainValuedVariable}
\getchunk{defun isEqualOrSubDomain}
\getchunk{defun isExposedConstructor}
\getchunk{defun isgenvar}
\getchunk{defun isEqualOrSubDomain}
\getchunk{defun isExposedConstructor}
\getchunk{defun isgenvar}
\getchunk{defun isPartialMode}
\getchunk{defun isListOfIdentifiers}
\getchunk{defun isListOfIdentifiersOrStrings}
\getchunk{defun isPatternVar}
\getchunk{defun isSharpVar}
\getchunk{defun isSharpVarWithNum}
\getchunk{defun isSubForRedundantMapName}
\getchunk{defun isSystemDirectory}
\getchunk{defun isTraceGensym}
\getchunk{defun isUncompiledMap}
\getchunk{defun justifyMyType}
\getchunk{defun kArgPage}
\getchunk{defun ArgumentCheck}
\getchunk{defun kcaPage}
\getchunk{defun kcaPage1}
\getchunk{defun kccPage}
\getchunk{defun kcdPage}
\getchunk{defun kcdoPage}
\getchunk{defun kCheckArgumentNumbers}
\getchunk{defun kcnPage}
\getchunk{defun kcPage}
\getchunk{defun kcPage1}
\getchunk{defun kDomainName}
\getchunk{defun kdPageInfo}
\getchunk{defun KeepPart?}
\getchunk{defun kePage}
\getchunk{defun kePageDisplay}
\getchunk{defun kePageOpAlist}
\getchunk{defun kiPage}
\getchunk{defun kisValidType}
\getchunk{defun koaPageFilterByName}
\getchunk{defun koPage}
\getchunk{defun koPageAux}
\getchunk{defun koPageAux1}
\getchunk{defun koPageFromKkPage}
\getchunk{defun koPageInputAreaUnchanged?}
\getchunk{defun koPage}
\getchunk{defun kcuPage}
\getchunk{defun kTestPred}
\getchunk{defun lassocSub}
\getchunk{defun lastTokPosn}
\getchunk{defun leader?}
\getchunk{defun leaveScratchpad}
\getchunk{defun lefts}
\getchunk{defun letPrint}
\getchunk{defun letPrint2}
\getchunk{defun letPrint3}
\getchunk{defun lfkey}
\getchunk{defun libConstructorSig}
\getchunk{defun library}
\getchunk{defun line?}
\getchunk{defun linearFinalRequest}
\getchunk{defun lineoftoks}
\getchunk{defun linkGen}
\getchunk{defun listConstructorAbbreviations}
\getchunk{defun listDecideHowMuch}
\getchunk{defun listOfStrings2String}
\getchunk{defun listOutputter}
\getchunk{defun lnFileName}
\getchunk{defun load}
\getchunk{defun loadFunctor}
\getchunk{defun loadLib}
\getchunk{defun loadLibNoUpdate}
\getchunk{defun localdatabase}
\getchunk{defun localnrlib}
\getchunk{defun lookupInDomainVector}
\getchunk{defun loopIters2Sex}
\getchunk{defun lotsof}
\getchunk{defun ltrace}
\getchunk{defun macroApplication}
\getchunk{defun macExpand}
\getchunk{defun macId}
\getchunk{defun macLambda}
\getchunk{defun macLambda,mac}
\getchunk{defun macLambdaParameterHandling}
\getchunk{defun macMacro}
\getchunk{defun macSubstituteId}
\getchunk{defun macSubstituteOuter}
\getchunk{defun macroExpanded}
\getchunk{defun macroWhere}
\getchunk{defun macWhere,mac}
\getchunk{defun macOExpandBody}
\getchunk{defun macroOGet}
\getchunk{defun macroOGetName}
\getchunk{defun macOInfiniteExpansion}
\getchunk{defun macOMLambdaApply}
\getchunk{defun mac0SubstituteOuter}
\getchunk{defun make-appendstream}
\getchunk{defun make-databases}
\getchunk{defun makeFullNnamestring}
\getchunk{defun makeHistFileNome}
\getchunk{defun makeInputFilename}
\getchunk{defun make-instream}
\getchunk{defun makeLeaderMsg}
\getchunk{defun makeMsgFromLine}
\getchunk{defun makeOrdinal}
\getchunk{defun make-outstream}
\getchunk{defun makePathname}
\getchunk{defun makeSpadCommand}
\getchunk{defun makeStream}
\getchunk{defun mapLetPrint}
\getchunk{defun mapStringize}
\getchunk{defun mergePathnames}
\getchunk{defun messageprint}
\getchunk{defun messageprint-1}
\getchunk{defun messageprint-2}
\getchunk{defun mkConArgSublis}
\getchunk{defun mkConform}
\getchunk{defun mkCurryFun}
\getchunk{defun mkDomPvar}
\getchunk{defun mkDomTypeForm}
\getchunk{defun mkEvalable}
\getchunk{defun mkEvalableMapping}
\getchunk{defun mkEvalableRecord}
\getchunk{defun mkEvalableUnion}
\getchunk{defun mkLineList}
\getchunk{defun mkprompt}
\getchunk{defun mkSetTitle}
\getchunk{defun mkUnixPattern}
\getchunk{defun msgCreate}
\getchunk{defun msgImPr?}
\getchunk{defun msgNoRep?}
\getchunk{defun msgOutputter}
\getchunk{defun msgText}
\getchunk{defun myWritable?}
\getchunk{defun namestring}
\getchunk{defun ncAlist}
\getchunk{defun ncBug}
\getchunk{defun ncConversationPhase}
\getchunk{defun ncConversationPhase,wrapup}
\getchunk{defun ncEltQ}
\getchunk{defun ncHardError}
\getchunk{defun ncIntLoop}
\getchunk{defun ncloopCommand}
\getchunk{defun ncloopDQlines}
\getchunk{defun ncloopIncFileName}
\getchunk{defun ncloopInclude}
\getchunk{defun ncloopInclude0}
\getchunk{defun ncloopInclude1}
\getchunk{defun ncloopParse}
\getchunk{defun ncParseFromString}
\getchunk{defun ncPutQ}
\getchunk{defun ncSoftError}
\getchunk{defun ncTag}
\getchunk{defun ncTopLevel}
\getchunk{defun newHelpSpad2Cmd}
\getchunk{defun next}
\getchunk{defun next1}
\getchunk{defun nextInterpreterFrame}
\getchunk{defun nextline}
\getchunk{defun next-lines-clear}
\getchunk{defun next-lines-show}
\getchunk{defun npAdd}
\getchunk{defun npADD}
\getchunk{defun npAmpersand}
\getchunk{defun npAmpersandFrom}
\getchunk{defun npAndOr}
\getchunk{defun npAngleBared}
\getchunk{defun npApplication}
\getchunk{defun npApplication2}
\getchunk{defun npArith}
\getchunk{defun npAssign}
\getchunk{defun npAssignment}
\getchunk{defun npAssignVariable}
\getchunk{defun npAtom1}
\getchunk{defun npAtom2}
\getchunk{defun npBacksetElse}
\getchunk{defun npBackTrack}
\getchunk{defun npBDefinition}
\getchunk{defun npBracedFileDefinition}
\getchunk{defun npBraced}
\getchunk{defun npBracketed}
\getchunk{defun npBracketed}
\getchunk{defun npBreak}
\getchunk{defun npBy}
\getchunk{defun npCategory}
\getchunk{defun npCategoryL}
\getchunk{defun npCoerceTo}
\getchunk{defun npColon}
\getchunk{defun npColonQuery}
\getchunk{defun npComma}
\getchunk{defun npCommaBackSet}
\getchunk{defun npCompMissing}
\getchunk{defun npConditional}
\getchunk{defun npConditionalStatement}
\texttt{\textbf{getchunk}}\{\texttt{defun npConstTok}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDDInfKey}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDecl}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDef}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefaultDecl}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefaultItem}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefaultItemlist}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefaultValue}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefinition}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefinitionItem}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefinitionlist}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefinitionOrStatement}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefn}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDefTail}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDiscrim}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDisjand}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDollar}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npDotted}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npElse}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npEncAp}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npEnclosed}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npEqKey}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npExit}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npExpress}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npExpress1}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npExport}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npFirstTok}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npFix}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npForIn}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npFree}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npFromdom}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npFromdom1}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npGives}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npId}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npImport}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npInfGeneric}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npInfixUp}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npInfixOperator}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npInfKey}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npInline}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npInterval}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npItem}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npItem1}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npIterate}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npIterator}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npIterators}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npLambda}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npLeftAssoc}\}
\texttt{\textbf{getchunk}}\{\texttt{defun npLet}\}
getchunk{defun npQuiver}
getchunk{defun npRecoverTrap}
getchunk{defun npRestore}
getchunk{defun npRestrict}
getchunk{defun npReturn}
getchunk{defun npRightAssoc}
getchunk{defun npRule}
getchunk{defun npSCategory}
getchunk{defun npSDefaultItem}
getchunk{defun npSegment}
getchunk{defun npSelector}
getchunk{defun npSemiBackSet}
getchunk{defun npSemiListing}
getchunk{defun npSigDecl}
getchunk{defun npSigItem}
getchunk{defun npSigItemlist}
getchunk{defun npSignature}
getchunk{defun npSignatureDefinee}
getchunk{defun npSingleRule}
getchunk{defun npSLocalItem}
getchunk{defun npSQualifiedList}
getchunk{defun npStatement}
getchunk{defun npSuch}
getchunk{defun npSuchThat}
getchunk{defun npSum}
getchunk{defun npSynonym}
getchunk{defun npSymbolVariable}
getchunk{defun npSynthetic}
getchunk{defun npSystem}
getchunk{defun npState}
getchunk{defun npTagged}
getchunk{defun npTerm}
getchunk{defun npTrap}
getchunk{defun npTrapForm}
getchunk{defun npTuple}
getchunk{defun npType}
getchunk{defun npTypedForm}
getchunk{defun npTypedForm1}
getchunk{defun npTypeStyle}
getchunk{defun npTypified}
getchunk{defun npTyping}
getchunk{defun npTypeVariable}
getchunk{defun npTypeVariableList}
getchunk{defun npVariable}
getchunk{defun npVariableList}
getchunk{defun npVariableName}
getchunk{defun npVoid}
getchunk{defun npWConditional}
getchunk{defun npWhile}
getchunk{defun npWith}
\getchunk{defun npZeroOrMore}
\getchunk{defun NRTEvalDomain}
\getchunk{defun ofCategory}
\getchunk{defun oldCompLookup}
\getchunk{defun oldHistFileName}
\getchunk{defun oldParseString}
\getchunk{defun om-bindTCP}
\getchunk{defun om-closeConn}
\getchunk{defun om-closeDev}
\getchunk{defun om-connectTCP}
\getchunk{defun om-getApp}
\getchunk{defun om-getAtp}
\getchunk{defun om-getAttr}
\getchunk{defun om-getBind}
\getchunk{defun om-getBVar}
\getchunk{defun om-getConnInDev}
\getchunk{defun om-getConnOutDev}
\getchunk{defun om-getEndApp}
\getchunk{defun om-getEndAtp}
\getchunk{defun om-getEndAttr}
\getchunk{defun om-getEndBind}
\getchunk{defun om-getEndBVar}
\getchunk{defun om-getEndError}
\getchunk{defun om-getError}
\getchunk{defun om-getFloat}
\getchunk{defun om-getInt}
\getchunk{defun om-getObject}
\getchunk{defun om-getString}
\getchunk{defun om-getSymbol}
\getchunk{defun om-getType}
\getchunk{defun om-getVar}
\getchunk{defun om-listCDs}
\getchunk{defun om-listSymbols}
\getchunk{defun om-makeConn}
\getchunk{defun om-openFileDev}
\getchunk{defun om-openStringDev}
\getchunk{defun om-putApp}
\getchunk{defun om-putAtp}
\getchunk{defun om-putAttr}
\getchunk{defun om-putBind}
\getchunk{defun om-putBVar}
\getchunk{defun om-putByteArray}
\getchunk{defun om-putEndApp}
\getchunk{defun om-putEndAtp}
\getchunk{defun om-putEndAttr}
\getchunk{defun om-putEndBind}
\getchunk{defun om-putEndBVar}
\getchunk{defun om-putEndError}
\defun om-putEndObject
\defun om-putError
\defun om-putFloat
\defun om-putInt
\defun om-putObject
\defun om-putObject
\defun om-putSymbol
\defun om-putVar
\defun om-Read
\defun om-setDevEncoding
\defun om-stringPtrToString
\defun om-stringToStringPtr
\defun om-supportsCD
\defun om-supportsSymbol
\defun openOutputLibrary
\defun opensever
\defun operationOpen
\defun optionError
\defun optionUserLevelError
\defun orderBySlotNumber
\defun originsInSlotNumber
\defun parseAndEval
\defun parseAndEval1
\defun parseAndInterpret
\defun parseFromString
\defun parseNoMacroFromString
\defun parseSystemCmd
\defun parseWord
\defun pathname
\defun pathnameDirectory
\defun pathnameName
\defun pathnameType
\defun pathnameTypeId
\defun patternVarsOf
\defun patternVarsOf1
\defun pcounters
\defun pfAbSynOp
\defun pfAbSynOp?
\defun pfAdd
\defun pfAnd
\defun pfAnd?
\defun pfApplication
\defun pfApplication?
\defun pfApplication2Sex
\defun pfAssign
\defun pfAssign?
\defun pfAttribute
\defun pfBrace
\defun pfBraceBar
CHAPTER 76. THE INTERPRETER

\getchunk{defun pfBracket}
\getchunk{defun pfBracketBar}
\getchunk{defun pfBreak}
\getchunk{defun pfBreak?}
\getchunk{defun pfCharPosn}
\getchunk{defun pfCheckArg}
\getchunk{defun pfCheckMacroOut}
\getchunk{defun pfCheckId}
\getchunk{defun pfCheckItOut}
\getchunk{defun pfCoerceto}
\getchunk{defun pfCoerceto?}
\getchunk{defun pfCollect}
\getchunk{defun pfCollect?}
\getchunk{defun pfCollect1?}
\getchunk{defun pfCollectArgTran}
\getchunk{defun pfCollectVariable1}
\getchunk{defun pfCollect2Sex}
\getchunk{defun pfCopyWithPos}
\getchunk{defun pfDefinition}
\getchunk{defun pfDefinition?}
\getchunk{defun pfDefinition2Sex}
\getchunk{defun pfDo}
\getchunk{defun pfDo?}
\getchunk{defun pfDocument}
\getchunk{defun pfEnSequence}
\getchunk{defun pfExit}
\getchunk{defun pfExit?}
\getchunk{defun pfExport}
\getchunk{defun pfExpression}
\getchunk{defun pfFileName}
\getchunk{defun pfFix}
\getchunk{defun pfFlattenApp}
\getchunk{defun pfFree}
\getchunk{defun pfFree?}
\getchunk{defun pfForin}
\getchunk{defun pfForin?}
\getchunk{defun pfFromDom}
\getchunk{defun pfFromdom}
\getchunk{defun pfGlobalLinePosn}
\getchunk{defun pfHide}
\getchunk{defun pfId}
\getchunk{defun pfId?}
\getchunk{defun pfIdPos}
\getchunk{defun pfIdSymbol}
\getchunk{defun pfIf}
\getchunk{defun pfIf?}
\getchunk{defun pfIfThenOnly}
\getchunk{defun pfImport}
\getchunk{defun pfInline}
\getchunk{defun pfInfApplication}
\getchunk{defun pfIterate}
\getchunk{defun pfIterate?}
\getchunk{defun pfLam}
\getchunk{defun pfLambda}
\getchunk{defun pfLambdaTran}
\getchunk{defun pfLambda?}
\getchunk{defun pfLambda2Sex}
\getchunk{defun pfLeaf}
\getchunk{defun pfLeaf?}
\getchunk{defun pfLeafPosition}
\getchunk{defun pfLeafToken}
\getchunk{defun pfLhsRule2Sex}
\getchunk{defun pfLinePosn}
\getchunk{defun pfList0f}
\getchunk{defun pfLiteralClass}
\getchunk{defun pfLiteralString}
\getchunk{defun pfLiteral2Sex}
\getchunk{defun pfLocal}
\getchunk{defun pfLocal?}
\getchunk{defun pfLoop}
\getchunk{defun pfLoop1}
\getchunk{defun pfLoop?}
\getchunk{defun pfLp}
\getchunk{defun pfMacro}
\getchunk{defun pfMacro?}
\getchunk{defun pfMapParts}
\getchunk{defun pfMLambda}
\getchunk{defun pfMLambda?}
\getchunk{defun pfname}
\getchunk{defun pfNoPosition}
\getchunk{defun pfNoPosition?}
\getchunk{defun pfNot?}
\getchunk{defun pfNothing}
\getchunk{defun pfNothing?}
\getchunk{defun pfNovalue}
\getchunk{defun pfNovalue?}
\getchunk{defun pfOp2Sex}
\getchunk{defun pfOr}
\getchunk{defun pfOr?}
\getchunk{defun pfParen}
\getchunk{defun pfPretend}
\getchunk{defun pfPretend?}
\getchunk{defun pfPushBody}
\getchunk{defun pfPushMacroBody}
\getchunk{defun pfQualType}
\getchunk{defun pfRestrict}
\getchunk{defun pfRestrict?}
\getchunk{defun pfRetractTo}
\getchunk{defun pfReturn}
CHAPTER 76. THE INTERPRETER

\getchunk{defun pfReturn?}
\getchunk{defun pfReturnNoName}
\getchunk{defun pfReturnTyped}
\getchunk{defun pfRhsRule2Sex}
\getchunk{defun pfRule}
\getchunk{defun pfRule?}
\getchunk{defun pfRule2Sex}
\getchunk{defun pfSequence}
\getchunk{defun pfSequence?}
\getchunk{defun pfSequenceToList}
\getchunk{defun pfSequence2Sex}
\getchunk{defun pfSequence2Sex0}
\getchunk{defun pfSexpr}
\getchunk{defun pfSexpr,strip}
\getchunk{defun pfSourcePosition}
\getchunk{defun pfSourceStok}
\getchunk{defun pfSpread}
\getchunk{defun pfSuch}
\getchunk{defun pfSuchthat}
\getchunk{defun pfSuchthat?}
\getchunk{defun pfSuchThat2Sex}
\getchunk{defun pfSymb}
\getchunk{defun pfSymbol}
\getchunk{defun pfSymbol?}
\getchunk{defun pfSymbolSymbol}
\getchunk{defun pfTagged}
\getchunk{defun pfTagged?}
\getchunk{defun pfTaggedToTyped}
\getchunk{defun pfTaggedToTyped1}
\getchunk{defun pfTransformArg}
\getchunk{defun pfTuple}
\getchunk{defun pfTupleListOf}
\getchunk{defun pfTweakIf}
\getchunk{defun pfTyped}
\getchunk{defun pfTyped?}
\getchunk{defun pfTyping}
\getchunk{defun pfTuple?}
\getchunk{defun pfUnSequence}
\getchunk{defun pfWDec}
\getchunk{defun pfWDeclare}
\getchunk{defun pfWhere}
\getchunk{defun pfWhere?}
\getchunk{defun pfWhile}
\getchunk{defun pfWhile?}
\getchunk{defun pfWith}
\getchunk{defun pfWrong}
\getchunk{defun pfWrong?}
\getchunk{defun pf0ApplicationArgs}
\getchunk{defun pf0DefinitionLhsItems}
\getchunk{defun pf0FlattenSyntacticTuple}
\getchunk{defun pf0ForinLhs}
\getchunk{defun pf0FreeItems}
\getchunk{defun pf0LambdaArgs}
\getchunk{defun pf0LocalItems}
\getchunk{defun pf0LoopIterators}
\getchunk{defun pf0MLambdaArgs}
\getchunk{defun pf0SequenceArgs}
\getchunk{defun pf0TupleParts}
\getchunk{defun pf0WhereContext}
\getchunk{defun pf2Sex}
\getchunk{defun pf2Sex1}
\getchunk{defun phMacro}
\getchunk{defun phParse}
\getchunk{defun phInterpret}
\getchunk{defun phIntReportMsgs}
\getchunk{defun pileCforest}
\getchunk{defun pileColumn}
\getchunk{defun pileCtree}
\getchunk{defun pileForest}
\getchunk{defun pileForest1}
\getchunk{defun pileForests}
\getchunk{defun pilePlusComment}
\getchunk{defun pilePlusComments}
\getchunk{defun pileTree}
\getchunk{defun poFileName}
\getchunk{defun poGlobalLinePosn}
\getchunk{defun poLinePosn}
\getchunk{defun poPosImmediate?}
\getchunk{defun porigin}
\getchunk{defun posend}
\getchunk{defun posPointers}
\getchunk{defun ppos}
\getchunk{defun pquit}
\getchunk{defun pquitSpad2Cmd}
\getchunk{defun previousInterpreterFrame}
\getchunk{defun printLabelledList}
\getchunk{defun printStatisticsSummary}
\getchunk{defun printStorage}
\getchunk{defun printSynonyms}
\getchunk{defun printTypeAndTime}
\getchunk{defun printTypeAndTimeNormal}
\getchunk{defun printTypeAndTimeSaturn}
\getchunk{defun probeName}
\getchunk{defun processChPosesForOneLine}
\getchunk{defun processInteractive}
\getchunk{defun processInteractive1}
\getchunk{defun processKeyedError}
\getchunk{defun processMsgList}
\getchunk{defun protectedEVAL}
\getchunk{defun processSynonymLine}
\getchunk{defun processSynonymLine, removeKeyFromLine}
\getchunk{defun processSynonyms}
\getchunk{defun prTraceNames}
\getchunk{defun prTraceNames, fn}
\getchunk{defun pspacers}
\getchunk{defun ptimers}
\getchunk{defun put}
\getchunk{defun putFTText}
\getchunk{defun punctuation?}
\getchunk{defun putDatabaseStuff}
\getchunk{defun putHist}
\getchunk{defun pvarCondList1}
\getchunk{defun pvarCondList}
\getchunk{defun pvarPredTran}
\getchunk{defun pvarsOfPattern}
\getchunk{defun queryClients}
\getchunk{defun queueUpErrors}
\getchunk{defun quit}
\getchunk{defun quitSpad2Cmd}
\getchunk{defun quoteString}
\getchunk{defun rassocSub}
\getchunk{defun rdefistream}
\getchunk{defun rdefoutstream}
\getchunk{defun read}
\getchunk{defun /read}
\getchunk{defun readHiFi}
\getchunk{defun readSpadProfileIfThere}
\getchunk{defun readSpad2Cmd}
\getchunk{defun recordAndPrint}
\getchunk{defun recordFrame}
\getchunk{defun recordNewValue}
\getchunk{defun recordNewValue0}
\getchunk{defun recordOldValue}
\getchunk{defun recordOldValue0}
\getchunk{defun reduceAlistForDomain}
\getchunk{defun redundant}
\getchunk{defun regress}
\getchunk{defun remFile}
\getchunk{defun remover}
\getchunk{defun removeOption}
\getchunk{defun removeTracedMapSigs}
\getchunk{defun removeUndoLines}
\getchunk{defun renamePatternVariables1}
\getchunk{defun renamePatternVariables}
\getchunk{defun replaceFile}
\getchunk{defun replacePercentByDollar, fn}
\getchunk{defun replacePercentByDollar}
\getchunk{defun replaceSharps}
\getchunk{defun reportAO}
\getchunk{defun reportCategory}
\getchunk{defun reportinstantiations}
\getchunk{defun reportOperations}
\getchunk{defun reportOpsFromLisplib}
\getchunk{defun reportOpsFromLisplib0}
\getchunk{defun reportOpsFromLisplib1}
\getchunk{defun reportOpsFromUnitDirectly}
\getchunk{defun reportOpsFromUnitDirectly0}
\getchunk{defun reportOpsFromUnitDirectly1}
\getchunk{defun reportSpadTrace}
\getchunk{defun reportUndo}
\getchunk{defun reportWhatOptions}
\getchunk{defun reroot}
\getchunk{defun resetCounters}
\getchunk{defun resethashables}
\getchunk{defun resetInCoreHist}
\getchunk{defun resetSpacers}
\getchunk{defun resetTimers}
\getchunk{defun resetWorkspaceVariables}
\getchunk{defun restart}
\getchunk{defun restart0}
\getchunk{defun restoreHistory}
\getchunk{defun retract}
\getchunk{defun rread}
\getchunk{defun ruleLhsTran}
\getchunk{defun rulePredicateTran}
\getchunk{defun runspad}
\getchunk{defun rwrite}

\getchunk{defun safeWritify}
\getchunk{defun sameMsg?}
\getchunk{defun satisfiesRegularExpressions}
\getchunk{defun saveHistory}
\getchunk{defun saveMapSig}
\getchunk{defun savesystem}
\getchunk{defun saveDependentsHashTable}
\getchunk{defun saveUsersHashTable}
\getchunk{defun sayAllCacheCounts}
\getchunk{defun sayBrightly1}
\getchunk{defun sayCacheCount}
\getchunk{defun sayExample}
\getchunk{defun sayKeyedMsg}
\getchunk{defun sayKeyedMsgLocal}
\getchunk{defun sayMSG}
\getchunk{defun sayMSG2File}
\getchunk{defun sayShowWarning}
\getchunk{defun scanCheckRadix}
\getchunk{defun scanComment}
\getchunk{defun scanDictCons}
\getchunk{defun scanError}
\getchunk{defun scanEsc}
\getchunk{defun scanEscape}
\getchunk{defun scanExponent}
\getchunk{defun scanIgnoreLine}
\getchunk{defun scanInsert}
\getchunk{defun scanKeyTr}
\getchunk{defun scanNegComment}
\getchunk{defun scanNumber}
\getchunk{defun ScanOrPairVec}
\getchunk{defun ScanOrPairVec, ScanOrInner}
\getchunk{defun scanPossFloat}
\getchunk{defun scanPunct}
\getchunk{defun scanPunCons}
\getchunk{defun scanS}
\getchunk{defun scanSpace}
\getchunk{defun scanString}
\getchunk{defun scanKeyTableCons}
\getchunk{defun scanToken}
\getchunk{defun scanTransform}
\getchunk{defun scanW}
\getchunk{defun scanWord}
\getchunk{defun search}
\getchunk{defun searchCurrentEnv}
\getchunk{defun searchTailEnv}
\getchunk{defun segmentKeyedMsg}
\getchunk{defun selectOption}
\getchunk{defun selectOptionLC}
\getchunk{defun separatePiles}
\getchunk{defun serverReadLine}
\getchunk{defun set}
\getchunk{defun set1}
\getchunk{defun setdatabase}
\getchunk{defun setExpose}
\getchunk{defun setExposeAdd}
\getchunk{defun setExposeAddConstr}
\getchunk{defun setExposeAddGroup}
\getchunk{defun setExposeDrop}
\getchunk{defun setExposeDropConstr}
\getchunk{defun setExposeDropGroup}
\getchunk{defun setFortDir}
\getchunk{defun setFortPers}
\getchunk{defun setFortTmpDir}
\getchunk{defun setFunctionsCache}
\getchunk{defun setHistoryCore}
\getchunk{defun setInputLibrary}
\getchunk{defun setIOindex}
\getchunk{defun setLinkerArgs}
\getchunk{defun setMsgCatlessAttr}
\getchunk{defun setMsgForcedAttr}
\texttt{\textit{getchunk\{defun setMsgForcedAttrList\}}}
\texttt{\textit{getchunk\{defun setMsgUnforcedAttr\}}}
\texttt{\textit{getchunk\{defun setMsgUnforcedAttrList\}}}
\texttt{\textit{getchunk\{defun setNagHost\}}}
\texttt{\textit{getchunk\{defun setOutputAlgebra\}}}
\texttt{\textit{getchunk\{defun setOutputCharacters\}}}
\texttt{\textit{getchunk\{defun setOutputFormula\}}}
\texttt{\textit{getchunk\{defun setOutputFortran\}}}
\texttt{\textit{getchunk\{defun setOutputLibrary\}}}
\texttt{\textit{getchunk\{defun setOutputHtml\}}}
\texttt{\textit{getchunk\{defun setOutputMathml\}}}
\texttt{\textit{getchunk\{defun setOutputOpenMath\}}}
\texttt{\textit{getchunk\{defun setOutputTex\}}}
\texttt{\textit{getchunk\{defun setStreamsCalculate\}}}
\texttt{\textit{getchunk\{defun setUpDefault\}}}
\texttt{\textit{getchunk\{defun shortenForPrinting\}}}
\texttt{\textit{getchunk\{defun show\}}}
\texttt{\textit{getchunk\{defun showdatabase\}}}
\texttt{\textit{getchunk\{defun showInOut\}}}
\texttt{\textit{getchunk\{defun showInput\}}}
\texttt{\textit{getchunk\{defun showSpad2Cmd\}}}
\texttt{\textit{getchunk\{defun shut\}}}
\texttt{\textit{getchunk\{defun size\}}}
\texttt{\textit{getchunk\{defun SkipEnd?\}}}
\texttt{\textit{getchunk\{defun SkipPart?\}}}
\texttt{\textit{getchunk\{defun Skipping?\}}}
\texttt{\textit{getchunk\{defun spad\}}}
\texttt{\textit{getchunk\{defun spadClosure?\}}}
\texttt{\textit{getchunk\{defun spad-error-loc\}}}
\texttt{\textit{getchunk\{defun SpadInterpretStream\}}}
\texttt{\textit{getchunk\{defun spad-long-error\}}}
\texttt{\textit{getchunk\{defun spadReply\}}}
\texttt{\textit{getchunk\{defun spadReply,printName\}}}
\texttt{\textit{getchunk\{defun spadrread\}}}
\texttt{\textit{getchunk\{defun spadrwrite\}}}
\texttt{\textit{getchunk\{defun spadrwrite0\}}}
\texttt{\textit{getchunk\{defun spad-save\}}}
\texttt{\textit{getchunk\{defun spad-short-error\}}}
\texttt{\textit{getchunk\{defun spadStartUpMsgs\}}}
\texttt{\textit{getchunk\{defun spad-syntax-error\}}}
\texttt{\textit{getchunk\{defun spadTrace\}}}
\texttt{\textit{getchunk\{defun spadTraceAlias\}}}
\texttt{\textit{getchunk\{defun spadTrace,\texttt{g}\}}}
\texttt{\textit{getchunk\{defun spadTrace,isTraceable\}}}
\texttt{\textit{getchunk\{defun spadUntrace\}}}
\texttt{\textit{getchunk\{defun spad2BootCoerce\}}}
\texttt{\textit{getchunk\{defun specialChar\}}}
\texttt{\textit{getchunk\{defun spletI\}}}
\texttt{\textit{getchunk\{defun spletII\}}}
\texttt{\textit{getchunk\{defun splitIntoOptionBlocks\}}}

(defun write-browsedb)
(defun write-categorydb)
(defun writeHiFi)
(defun writeHistModesAndValues)
(defun writeInputLines)
(defun write-interpdb)
(defun write-operationdb)
(defun write-warmdata)
(defun writify)
(defun writifyComplain)
(defun writify-writifyInner)

(defun x1CannotRead)
(defun x1CmdBug)
(defun x1ConActive)
(defun x1Console)
(defun x1ConStill)
(defun x1FileCycle)
(defun x1IfBug)
(defun x1IfSyntax)
(defun x1Msg)
(defun x1NoSuchFile)
(defun x1OK)
(defun x1OK1)
(defun x1PrematureEOF)
(defun x1PrematureFin)
(defun x1Say)
(defun x1Skip)
(defun x1SkippingFin)

(defun yesanswer)

(defun zsystemdevelopment)
(defun zsystemdevelopment1)
(defun zsystemDevelopmentSpad2Cmd)

(postvars)
Chapter 77

The Global Variables

77.1 Star Global Variables

<table>
<thead>
<tr>
<th>NAME</th>
<th>SET</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>eof*</td>
<td>ncTopLevel</td>
<td></td>
</tr>
<tr>
<td>features*</td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td>package*</td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td>standard-input*</td>
<td></td>
<td>ncIntLoop</td>
</tr>
<tr>
<td>standard-output*</td>
<td></td>
<td>ncIntLoop</td>
</tr>
<tr>
<td>top-level-hook*</td>
<td></td>
<td>set-restart-hook</td>
</tr>
</tbody>
</table>

*eof*

The *eof* variable is set to NIL in ncTopLevel.

*features*

The *features* variable from common lisp is tested for the presence of the :unix keyword. Apparently this controls the use of Saturn, a previous Axiom frontend. The Saturn frontend was never released as open source and so this test and the associated variables are probably not used.

*package*

The *package* variable, from common lisp, is set in restart to the BOOT package where the interpreter lives.
*standard-input*

The *standard-input* common lisp variable is used to set the curinstream variable in ncIntLoop.

This variable is an argument to serverReadLine in the intloopReadConsole function.

*standard-output*

The *standard-output* common lisp variable is used to set the curoutstream variable in ncIntLoop.

*top-level-hook*

The *top-level-hook* common lisp variable contains the name of a function to invoke when an image is started. In our case it is called restart. This is the entry point to the Axiom interpreter.
### 77.2 Dollar Global Variables

<table>
<thead>
<tr>
<th>NAME</th>
<th>SET</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$boot</td>
<td>nctoplevel</td>
<td>runspad</td>
</tr>
<tr>
<td>coercionFailure</td>
<td>nctoplevel</td>
<td></td>
</tr>
<tr>
<td>curinstream</td>
<td>ncinloop</td>
<td></td>
</tr>
<tr>
<td>curoutstream</td>
<td>ncinloop</td>
<td></td>
</tr>
<tr>
<td>$currentLine</td>
<td>restart</td>
<td>removeUndoLines</td>
</tr>
<tr>
<td>$daiymode</td>
<td></td>
<td>intloopReadConsole</td>
</tr>
<tr>
<td>$displayStartMsgs</td>
<td></td>
<td>restart</td>
</tr>
<tr>
<td>$e</td>
<td>nctoplevel</td>
<td></td>
</tr>
<tr>
<td>$eMsgToss</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$fn</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$frameRecord</td>
<td>initvars</td>
<td>clearFrame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undoSteps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recordFrame</td>
</tr>
<tr>
<td>$HiFiAccess</td>
<td>initHist</td>
<td>historySpad2Cmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setHistoryCore</td>
</tr>
<tr>
<td>$HistList</td>
<td>initHist</td>
<td></td>
</tr>
<tr>
<td>$HistListAct</td>
<td>initHist</td>
<td></td>
</tr>
<tr>
<td>$HistListLen</td>
<td>initHistList</td>
<td></td>
</tr>
<tr>
<td>$HistRecord</td>
<td>initHistList</td>
<td></td>
</tr>
<tr>
<td>$historyDirectory</td>
<td></td>
<td>makeHistFileName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>makeHistFileName</td>
</tr>
<tr>
<td>$historyFileType</td>
<td>initvars</td>
<td>histInputFileName</td>
</tr>
<tr>
<td>$InteractiveFrame</td>
<td>restart</td>
<td>nctoplevel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recordFrame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undoSteps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reportUndo</td>
</tr>
<tr>
<td>$internalHistoryTable</td>
<td>initvars</td>
<td></td>
</tr>
<tr>
<td>$interpreterFrameName</td>
<td>initializeInterpreterFrameRing</td>
<td></td>
</tr>
<tr>
<td>$interpreterFrameRing</td>
<td>initializeInterpreterFrameRing</td>
<td></td>
</tr>
<tr>
<td>$intRestart</td>
<td></td>
<td>intloop</td>
</tr>
<tr>
<td>$intTopLevel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$IOindex</td>
<td>restart</td>
<td>historySpad2Cmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undoCount</td>
</tr>
<tr>
<td>$genValue</td>
<td></td>
<td>i-toplev</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-analy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-syscmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-spec1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-spec2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-map</td>
</tr>
<tr>
<td>$lastPos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$libQuiet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$msgDatabaseName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ncMsgList</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$newcompErrorCount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$newspad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$nones</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$okToExecuteMachineCode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$oldHistoryFileName</td>
<td>initvars</td>
<td>oldHistFileName</td>
</tr>
<tr>
<td>$options</td>
<td></td>
<td>historySpad2Cmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$boot
The $boot variable is set to NIL in ncTopLevel.

c coerceFailure
The coerceFailure symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

$currentLine
The $currentLine line is set to NIL in restart. It is used in removeUndoLines in the undo mechanism.

$displayStartMsgs
The $displayStartMsgs variable is used in restart but is not set so this is likely a bug.

$e
The $e variable is set to the value of $InteractiveFrame which is set in restart to the value of the call to the makeInitialModemapFrame function. This function simply returns a copy of the variable $InitialModemapFrame.
Thus $e is a copy of the variable $InitialModemapFrame.
This variable is used in the undo mechanism.

$erMsgToss
The $erMsgToss variable is set to NIL in SpadInterpretStream.

$fn
The $fn variable is set in SpadInterpretStream. It is set to the second argument which is a list. It appears that this list has the same structure as an argument to the LispVM rdefi ostream function.

$frameRecord

$frameRecord = [delta1, delta2, ...] where delta(i) contains changes in the “backwards” direction. Each delta(i) has the form ((var . proplist)... where proplist denotes an ordinary proplist. For example, an entry of the form ((x (value) (mode (Integer)))...) indicates that to undo 1 step, x’s value is cleared and its mode should be set to (Integer).
A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a system command is given which alters the environment.

recordFrame('system) is called before a command is executed, and recordFrame('normal) is called after (see processInteractive1). If no changes are found for former, no special entry is given.

This is part of the undo mechanism.

$HiFiAccess

The $HiFiAccess is set by initHist to T. It is a flag used by the history mechanism to record whether the history function is currently on. It can be reset by using the axiom command

)history off

It appears that the name means “History File Access”.

The $HiFiAccess variable is used by historySpad2Cmd to check whether history is turned on. T means it is, NIL means it is not.

$HistList

The $HistList variable is set by initHistList to an initial value of NIL elements. The last element of the list is smashed to point to the first element to make the list circular. This is a circular list of length $HistListLen.

$HistListAct

The $HistListAct variable is set by initHistList to 0. This variable holds the actual number of elements in the history list. This is the number of “undoable” steps.

$HistListLen

The $HistListLen variable is set by initHistList to 20. This is the length of a circular list maintained in the variable $HistList.

$HistRecord

The $HistRecord variable is set by initHistList to NIL. $HistRecord collects the input line, all variable bindings and the output of a step, before it is written to the file named by the function histFileName.
$\text{historyFileType}$

The $\text{historyFileType}$ is set at load time by a call to initvars to a value of “axh”. It appears that this is intended to be used as a filetype extension. It is part of the history mechanism. It is used in makeHistFileName as part of the history file name.

$\text{internalHistoryTable}$

The $\text{internalHistoryTable}$ variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism.

$\text{interpreterFrameName}$

The $\text{interpreterFrameName}$ variable, set in initializeInterpreterFrameRing to the constant initial to indicate that this is the initial (default) frame.

Frames are structures that capture all of the variables defined in a session. There can be multiple frames and the user can freely switch between them. Frames are kept in a ring data structure so you can move around the ring.

$\text{interpreterFrameRing}$

The $\text{interpreterFrameRing}$ is set to a pair whose car is set to the result of emptyInterpreterFrame

$\text{InteractiveFrame}$

The $\text{InteractiveFrame}$ is set in the restart function to the value of the call to the makeInitialModemapFrame function. This function simply returns a copy of the variable $\text{InitialModemapFrame}$

$\text{intRestart}$

The $\text{intRestart}$ variable is used in intloop but has no value. This is probably a bug. While the variable’s value is unchanged the system will continually reenter the SpadInterpretStream function.

$\text{intTopLevel}$

The $\text{intTopLevel}$ is a catch tag. Throwing to this tags which is caught in the intloop will restart the SpadInterpretStream function.
$IOindex

The $IOindex index variable is set to 1 in restart. This variable is used in the historySpad2Cmd function in the history mechanism. It is set in the removeUndoLines function in the undo mechanism.

This is used in the undo mechanism in function undoCount to compute the number of undos. You can’t undo more actions then have already happened.

$lastPos

The $lastPos variable is set in SpadInterpretStream to the value of the $nopos variable. Since $nopos appears to have no value this is likely a bug.

$libQuiet

The $libQuiet variable is set to the third argument of the SpadInterpretStream function. This is passed from intloop with the value of T. This variable appears to be intended to control the printing of library loading messages which would need to be suppressed if input was coming from a file.

$msgDatabaseName

The $msgDatabaseName is set to NIL in reroot.

$ncMsgList

The $ncMsgList is set to NIL in SpadInterpretStream.

$newcompErrorCount

The $newcompErrorCount is set to 0 in SpadInterpretStream.

$newspad

The $newspad is set to T in ncTopLevel.

$nupos

The $nupos variable is used in SpadInterpretStream but does not appear to have a value and is likely a bug.
$oldHistoryFileName

The $oldHistoryFileName is set at load time by a call to initvars to a value of “last”. It is part of the history mechanism. It is used in the function oldHistFileName and restoreHistory.

$okToExecuteMachineCode

The $okToExecuteMachineCode is set to T in SpadInterpretStream.

$options

The $options variable is tested by the history function. If it is NIL then output the message

    You have not used the correct syntax for the history command.
    Issue )help history for more information.

The $options variable is tested in the historySpad2Cmd function. It appears to record the options that were given to a spad command on the input line. The function selectOptionLC appears to take a list off options to scan.
This variable is not yet set and is probably a bug.

$previousBindings

The $previousBindings is a copy of the CAAR $InteractiveFrame. This is used to compute the delta(i)s stored in $frameRecord. This is part of the undo mechanism.

$PrintCompilerMessageIfTrue

The $PrintCompilerMessageIfTrue variable is set to NIL in spad.

$reportundo

The $reportundo variable is used in diffAlist. It was not normally bound but has been set to T in initvars. If the variable is set to T then we call reportUndo.
It is part of the undo mechanism.

$spad

The $spad variable is set to T in ncTopLevel.
$SpadServer

If an open server is not requested then this variable to T. It has no value before this time (and is thus a bug).

$SpadServerName

The $SpadServerName is passed to the openServer function, if the function exists.

$systemCommandFunction

The $systemCommandFunction is set in SpadInterpretStream to point to the function Inter-
pExecuteSpadSystemCommand.

top_level

The top_level symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

$quitTag

The $quitTag is used as a variable in a catch block. It appears that it can be thrown somewhere below ncTopLevel.

$useInternalHistoryTable

The $useInternalHistoryTable variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism.

$undoFlag

The $undoFlag is used in recordFrame to decide whether to do undo recording. It is initially set to T in initvars. This is part of the undo mechanism.
Bibliography

[1] Daly, Timothy, "The Axiom Literate Documentation"
   http://axiom.axiom-developer.org/axiom-website/documentation.html

[2] Daly, Timothy, “The Axiom Wiki Website"
   http://axiom.axiom-developer.org


Chapter 78

Index
Index

*allOperations*, 1000
  used by allOperations, 1033
  used by localnrlib, 1016
defvar, 1000

*allconstructors*, 1000
  used by allConstructors, 1032
  used by browseOpen, 1005
  used by interpOpen, 1004
  used by localnrlib, 1016
  used by make-databases, 1019
  used by resethashtables, 1001
defvar, 1000

*ancestors-hash*
  used by write-interpdb, 1028

*browse-stream*, 999
  used by browseOpen, 1005
  used by getdatabase, 1010
  used by resethashtables, 1000
defvar, 999

*browse-stream-stamp*, 999
  used by browseOpen, 1005
defvar, 999

*build-version*
  used by axiomVersion, 476
  used by spadStartUpMsgs, 17

*category-stream*, 999
  used by categoryOpen, 1006
  used by getdatabase, 1010
  used by resethashtables, 1000
defvar, 999

*category-stream-stamp*, 1000
  used by categoryOpen, 1006
  used by getdatabase, 1010
  used by resethashtables, 1000
defvar, 1000

*defaultdomain-list*, 996
  used by getdatabase, 1010
defvar, 996

*eof*, 22
  used by ncTopLevel, 23
  used by serverReadLine, 43
defvar, 22

*hasCategory-hash*, 997
  used by categoryOpen, 1006
  used by getdatabase, 1010
  used by write-categorydb, 1031
defvar, 997

*hascategory-hash*
  used by getdatabase, 1010
  used by resethashtables, 1001

*index-filename*
  used by localdatabase, 1014

*interp-stream*, 998
  used by getdatabase, 1010
  used by interpOpen, 1004
  used by resethashtables, 1000
defvar, 998

*interp-stream-stamp*, 998
  used by interpOpen, 1004
  used by resethashtables, 1001
defvar, 998

*miss*, 997
  used by getdatabase, 1010
defvar, 997

*monitor-domains*, 1163
  used by monitor-readinterp, 1176
  used by monitor-report, 1177
  used by monitor-spadfile, 1178
defvar, 1163

*monitor-nrlibs*, 1163
  used by monitor-dirname, 1174
defvar, 1163

*monitor-table*, 1164
  used by monitor-add, 1166
  used by monitor-apropos, 1179
used by monitor-checkpoint, 1171
used by monitor-decr, 1168
used by monitor-delete, 1166
used by monitor-disable, 1167
used by monitor-enable, 1166
used by monitor-end, 1165
used by monitor-incr, 1168
used by monitor-info, 1169
used by monitor-inittable, 1164
used by monitor-nrlib, 1175
used by monitor-percent, 1179
used by monitor-reset, 1167
used by monitor-results, 1165
used by monitor-untested, 1169
defvar, 1164
*monitor-table*)
  used by monitor-tested, 1170
*msghash*, 327
  used by cacheKeyedMsg, 329
  used by fetchKeyedMsg, 328
defvar, 327
*operation-hash*, 997
  used by addoperations, 1007
  used by allOperations, 1033
  used by getdatabase, 1010
  used by make-databases, 1019
  used by operationOpen, 1007
  used by resethashtables, 1001
  used by write-operationdb, 1031
defvar, 997
*operation-stream*, 998
  used by getdatabase, 1010
  used by operationOpen, 1007
  used by resethashtables, 1000
defvar, 998
*operation-stream-stamp*, 999
  used by operationOpen, 1007
  used by resethashtables, 1000
defvar, 999
*print-package*
  used by monitor-checkpoint, 1171
*print-pretty*
  used by write-browsedb, 1029
  used by write-categorydb, 1030
  used by write-interpdb, 1028
*sourcefiles*
  used by make-databases, 1019
  used by resethashtables, 1000
  used by write-browsedb, 1029
*standard-output*
  used by init-boot/spad-reader, 968
*workspace*, 22
  used by assertCond, 94
defvar, 22
*yearweek*
  used by axiomVersion, 476
  used by spadStartUpMsgs, 17
/D,1
called by compileBoot, 907
called by zsystemdevelopment1, 954
/breakcondition
  used by break, 906
/comp
called by zsystemdevelopment1, 954
/countlist
  used by pcounters, 860
  used by resetCounters, 858
  used by resetWorkspaceVariables, 654
/editfile, 515
  used by /read, 644
  used by editSpad2Cmd, 545
  used by readSpad2Cmd, 643
  used by readSpadProfileIfThere, 961
  used by resetWorkspaceVariables, 654
defvar, 515
/pretty
  used by resetWorkspaceVariables, 654
/read, 644
called by readSpad2Cmd, 642
calls /rf[9], 644
calls /rq[9], 644
uses /editfile, 644
defun, 644
/rd[9]
called by /read, 644
/rd[9]
called by /read, 644
/sourcefiles
  used by resetWorkspaceVariables, 654
/spacelist
  used by pspacers, 859
  used by resetSpacers, 858
used by resetWorkspaceVariables, 654
	/timerlist
		used by ptimers, 859
		used by resetTimers, 858
		used by resetWorkspaceVariables, 654
	/trace, 0
	
called by trace1, 848
	/tracenames
		used by /tracerreply, 894
		used by ?t, 903
		used by getBplNameIfTracedMap, 890
		used by getMapSubNames, 869
		used by prTraceNames, 898
		used by spadReply, 895
		used by spadTrace, 877
		used by spadUntrace, 896
		used by traceReply, 900
		used by untraceDomainConstructor, 883
		used by untrace, 862
	/tracerreply, 894
	
calls devaluate, 894
	
calls exit, 894
	
calls isDomainOrPackage, 894
	
calls qcar, 894
	
calls seq, 894
	
defun, 894
	
defun, 894
	/untrace, 0
	
called by untraceDomainConstructor, keepTraced?, 882
	
called by untrace, 862
	/untrace, 2
	
called by untraceMapSubNames, 873
	/version
	
called by zsystemdevelopment1, 954
	/wsname
	
called by zsystemdevelopment1, 954
	?t, 902
	
called by trace1, 848
	
calls bright, 903
	
calls devaluate, 903
	
calls get, 902
	
calls isDomainOrPackage, 903
	
calls isDomain, 903
	
calls isgenvar, 902
	
calls qcar, 903

calls qcdr, 903

calls rassocSub, 903

calls reportSpadTrace, 903

calls sayBrightly, 903

calls sayMSG, 903

calls take, 903

takes /tracenames, 903

takes $InteractiveFrame, 903

takes $mapSubNameAlist, 903

defun, 902

$FINDFILE

called by $editSpad2Cmd, 544

$erase

called by $addNewInterpreterFrame, 561

called by $closeInterpreterFrame, 562

$copy

called by $restoreHistory, 597

$filep

called by $setOutputAlgebra, 763

called by $setOutputFormula, 790

called by $setOutputFortran, 770

called by $setOutputHtml, 781

called by $setOutputMathml, 776

called by $setOutputOpenMath, 786

called by $setOutputTex, 797

$nagMessages, 758

defvar, 758

$replace

called by $setHist, 581

$reportBottomUpFlag, 747

defvar, 747

$systemCommandFunction

called by $intProcess, 62

called by $ncCommand, 478

$BreakMode, 661

takes letPrint2, 887

takes letPrint3, 888

defvar, 661

$CallInterp

takes serverReadLine, 43

$CatOfCatDatabase

takes clearCmdCompletely, 502

$categoryFrame

takes loadLibNoUpdate, 1037

takes loadLib, 1036

takes getProplst, 964
used by reportOpsFromUnitDirectly, 822
used by systemCommand, 448
$CloseClient
  used by close, 511
$Coerce
  local ref unifyStructVar, 429
  used by processInteractive, 48
$CommandSynonymAlist, 478
  used by npProcessSynonym, 473
  used by printSynonyms, 474
  used by processSynonyms, 32
  used by resetWorkspaceVariables, 654
  used by synonymSpad2Cmd, 832
defvar, 478
$ComplexInteger
  local ref hasCateSpecialNew, 436
$ConstructorCache
  used by clearCmdSortedCaches, 501
  used by localdatabase, 1014
  used by traceDomainConstructor, 881
$CreateFrameAnswer
  used by serverReadLine, 43
$CreateFrame
  used by serverReadLine, 43
$DomOfCatDatabase
  used by clearCmdCompletely, 502
$EchoLines
  used by intloopEchoParse, 67
$EmptyEnvironment
  used by describeSpad2Cmd, 529
  used by displaySpad2Cmd, 536
$EmptyMode
  local ref evaluateType1, 918
  local ref evaluateType, 916
  local ref hasCate, 433
  local ref isPartialMode, 420
  local ref mkEvalable, 913
  local ref retract, 1064
  used by bcComplexLimit, 1224
  used by bcDefiniteIntegrate, 1191
  used by bcDifferentiate, 1195
  used by bcDraw2DSolve, 1202
  used by bcDraw2Dfun, 1198
  used by bcDraw2Dpar, 1200
  used by bcDraw3Dfun, 1204
  used by bcDraw3Dpar1, 1208
  used by bcDraw3Dpar, 1206
  used by bcIndefiniteIntegrate, 1189
  used by bcInputEquations, 1232
  used by bcInputExplicitMatrix, 1186
  used by bcLaurentSeries, 1216
  used by bcLimit, 1220
  used by bcLinearSolveEqns, 1228
  used by bcLinearSolveMatrix1, 1237
  used by bcLinearSolveMatrixInhomo, 1238
  used by bcLinearSolve, 1227
  used by bcProduct, 1194
  used by bcPuiseuxSeries, 1217
  used by bcRealLimit, 1221
  used by bcSeriesExpansion, 1211
  used by bcSeries, 1210
  used by bcSolve, 1226
  used by bcSum, 1193
  used by bcTaylorSeries, 1214
  used by displayValue, 459
  used by interpret2, 53
  used by recordAndPrint, 55
$EndOfOutput
  used by serverReadLine, 43
$EndServerSession, 42
  used by serverReadLine, 43
defvar, 42
$EndSession
  used by serverReadLine, 43
$FormalMapVariableList
  local ref domArg, 422
  local ref mkDomPvar, 433
  local ref replaceSharps, 958
  used by dbConstructorDoc, 1381
  used by dbGetDocTable, 1384
  used by dbSubConform, 1387
  used by displayOperationsFromLisplib, 821
  used by localrilib, 1016
  used by reportOpsFromLisplib, 818
$FunctionalExpression
  local ref defaultTargetFE, 437
$HTCompanionWindowID, 50
  used by recordAndPrint, 54
defvar, 50
$HiFiAccess, 733
  used by createCurrentInterpreterFrame, 557
  used by disableHist, 603
used by emptyInterpreterFrame, 556
used by historySpad2Cmd, 583
used by initHist, 581
used by putHist, 591
used by restoreHistory, 597
used by saveHistory, 596
used by setHistoryCore, 585
used by undoFromFile, 594
used by undoInCore, 593
used by updateFromCurrentInterpreterFrame, 558
used by updateHist, 590
used by writeInputLines, 587
defvar, 733
$HistListAct
used by changeHistListLen, 589
used by createCurrentInterpreterFrame, 557
used by emptyInterpreterFrame, 556
used by initHistList, 581
used by resetInCoreHist, 588
used by undoInCore, 593
used by updateFromCurrentInterpreterFrame, 558
used by updateInCoreHist, 590
$HistListLen
used by changeHistListLen, 589
used by createCurrentInterpreterFrame, 557
used by emptyInterpreterFrame, 556
used by initHistList, 581
used by resetInCoreHist, 588
used by undoInCore, 593
used by updateFromCurrentInterpreterFrame, 558
used by updateInCoreHist, 590
$HistList
used by changeHistListLen, 589
used by createCurrentInterpreterFrame, 557
used by emptyInterpreterFrame, 556
used by initHistList, 581
used by resetInCoreHist, 588
used by undoInCore, 593
used by updateFromCurrentInterpreterFrame, 558
used by updateInCoreHist, 590
$HistRecord
used by createCurrentInterpreterFrame, 557
used by emptyInterpreterFrame, 556
used by initHistList, 581
used by recordNewValue0, 591
used by updateFromCurrentInterpreterFrame, 558
used by updateHist, 590
used by writeHiFi, 602
$IOindex, 10
$IOindex
used by createCurrentInterpreterFrame, 557
used by historySpad2Cmd, 583
used by mkprompt, 40
used by removeUndoLines, 934
used by resetWorkspaceVariables, 654
used by restart, 16
used by setHistoryCore, 585
used by setIOindex, 599
used by undoCount, 929
used by undoInCore, 593
used by undoSteps, 930
used by updateFromCurrentInterpreterFrame, 558
used by updateHist, 589
used by writeHiFi, 602
used by writeInputLines, 587
defvar, 10
$InitialCommandSynonymAlist, 476
$InitialModemapFrame, 7
$Integer
local ref defaultTargetFE, 437
local ref hasCateSpecialNew, 436
local ref hasCateSpecial, 435
local ref mkEvalable, 913
$InteractiveFrame
local ref isDomainValuedVariable, 959
used by ?, 903
used by augmentTraceNames, 872
used by clearCmdAll, 503
used by clearCmdParts, 505
used by createCurrentInterpreterFrame, 557
used by getAliasIfTracedMapParameter, 890
INDEX

used by getBpiNameIfTracedMap, 890
used by getMapSig, 853
used by getMapSubNames, 869
used by getWorkspaceNames, 455
used by interpFunctionDepAlists, 465
used by isInterpOnlyMap, 872
used by isUncompiledMap, 871
used by ncTopLevel, 23
used by parseAndInterpret, 46
used by processInteractive1, 50
used by recordFrame, 923
used by reportundo, 928
used by restart, 16
used by restoreHistory, 597
used by showSpad2Cmd, 815
used by undoChanges, 594
used by undoFromFile, 594
used by undoInCore, 593
used by undoSteps, 930
used by undo, 922
used by updateFromCurrentInterpreterFrame, 558
used by writeHistModesAndValues, 603
$InteractiveMode, 22
  local def unparseInputForm, 1054
  local ref loadLibNoUpdate, 1037
  local ref loadLib, 1036
used by addBinding, 963
used by ncTopLevel, 23
used by parseAndInterpret, 46
used by readSpad2Cmd, 642
used by zsystemDevelopmentSpad2Cmd, 953
used by zsystemdevelopment1, 954
defvar, 22
$JoinOfCatDatabase
  used by clearCmdCompletely, 502
$JoinOfDomDatabase
  used by clearCmdCompletely, 502
$KillLispSystem
  used by serverReadLine, 43
$LispCommand
  used by serverReadLine, 43
$MenuServer
  used by executeQuietCommand, 46
  used by serverReadLine, 43
$NeedToSignalSessionManager, 42
  used by intloopSpadProcess, 63
  used by serverReadLine, 43
defvar, 42
$NonNullStream, 611
  used by dewritify,dewritifyInner, 612
  used by writify,writifyInner, 607
defvar, 611
$NonSmanSession
  used by serverReadLine, 43
$NullStream, 611
  used by dewritify,dewritifyInner, 612
  used by writify,writifyInner, 607
defvar, 611
$OutputForm
  used by coerceSpadArgs2E, 865
  used by coerceSpadFunValue2E, 868
  used by coerceTraceArgs2E, 864
  used by coerceTraceFunValue2E, 867
$PatternVariableList
  used by kDomainName, 1370
$Primitives
  used by conOpPage1, 1376
  used by dbShowConsDoc1, 1393
$PrintCompilerMessageIfTrue
  used by spad, 18
$ProcessInteractiveValue, 50
  used by isValidType, 1373
  used by processInteractive1, 50
  used by processInteractive, 48
  used by topLevelInterpEval, 1372
defvar, 50
$QueryClients
  used by queryClients, 510
$QuickLet
  used by breaklet, 905
  used by tracelet, 904
$QuietCommand, 45
  used by executeQuietCommand, 46
  used by pf2Sex1, 302
  used by pf2Sex, 299
  used by recordAndPrint, 54
defvar, 45
$QuietSpadCommand
  used by serverReadLine, 43
$QuotientField
INDEX

local ref hasCateSpecial, 435
$RTspecialCharacters, 978
  used by setOutputCharacters, 767
defvar, 978
$RationalNumber
  local ref defaultTargetFE, 438
  local ref hasCateSpecialNew, 436
$SessionManager
  used by close, 511
  used by queryClients, 510
  used by serverReadLine, 43
$SpadCommand
  used by serverReadLine, 43
$SpadServerName, 10
  used by restart, 15
defvar, 10
$SpadServer, 10
  used by close, 511
  used by restart, 16
  used by serverReadLine, 43
  used by spad-save, 990
defvar, 10
$StreamFrame
  used by processInteractive, 48
$Subst
  local def ofCategory, 419
  local ref hasCate, 433
  local ref unifyStructVar, 429
$SwitchFrames
  used by serverReadLine, 43
defvar, 43
Symbol
  local ref defaultTargetFE, 438
$ThrowAwayMode
  used by interpret2, 53
$TriangleVariableList
  local ref replaceSharps, 958
  used by dbShowConsDoc1, 1393
  used by libConstructorSig, 1407
$UserAbbreviationsAlist
  used by resetWorkspaceVariables, 654
$UserLevel, 807
  used by getDirectoryList, 984
  used by readSpad2Cmd, 642
  used by satisfiesUserLevel, 451
  used by set1, 809
  used by synonymsForUserLevel, 833
  used by userLevelErrorMessage, 451
  used by whatCommands, 942
defvar, 807
$Void
  used by clearCmdSortedCaches, 501
  used by recordAndPrint, 55
defvar, 760
$activePageList, 1249
defvar, 1249
$algebraFormat, 761
  used by setOutputAlgebra, 763
defvar, 761
$algebraOutputFile, 762
  used by setOutputAlgebra, 763
defvar, 762
$algebraOutputStream, 762
  used by recordAndPrint, 55
  used by sayMSG, 331
  used by setOutputAlgebra, 763
defvar, 762
$analyzingMapList
  used by processInteractive, 48
$ans
  used by nplisp, 472
$args
  used by dbConstructorDoc,hn, 1381
$atLeastOneUnexposed
  used by dbConsExposureMessage, 1391
$attrCats, 364
  used by whichCat, 365
defvar, 364
$attributeDb
  used by clearCmdCompletely, 502
$bcParseOnly, 1249
  used by bcInputEquations, 1232
  used by bcInputExplicitMatrix, 1186
  used by bcInputMatrixByFormula, 1184
defvar, 1249
$boot, 23
  used by iclear, 972
  used by iostat, 970
  used by ncTopLevel, 23
  used by parseAndInterpret, 46
  used by resetWorkspaceVariables, 654
defvar, 23
INDEX

$cacheAlist, 707
defvar, 707
$cacheMessages, 326
defvar, 326
$clearExcept, 499
  usedby clearSpad2Cmd, 500
defvar, 499
$clearOptions, 499
  usedby clearCmdExcept, 505
  usedby clearCmdParts, 505
  usedby clearSpad2Cmd, 500
defvar, 499
$coerceIntByMapCounter
  usedby resetWorkspaceVariables, 654
$collectOutput
  usedby printStatisticsSummary, 56
  usedby printStorage, 56
  usedby printTypeAndTimeNormal, 57
  usedby recordAndPrint, 55
$commentedOps
  usedby reportOpsFromUnitDirectly, 822
$compErrorMessageStack
  usedby processInteractive, 48
$compileDontDefineFunctions, 712
defvar, 712
$compileMapFlag
  usedby resetWorkspaceVariables, 654
$compileRecurrence, 712
defvar, 712
$compilingLoop
  usedby processInteractive, 48
$compilingMap
  usedby processInteractive, 48
$conArgstrings
  usedby conPage, 1343
$conformsAreDomains
  usedby kePage, 1352
$conformsAreDomains
  usedby dbShowCons1, 1390
  usedby dbSpecialDescription, 1398
  usedby kiPage, 1351
$conform
  usedby conPageConEntry, 1344
  usedby dbGetDocTable.gn, 1384
  usedby dbGetDocTable.hm, 1384
  usedby dbGetDocTable, 1385
$conname
  usedby conPageConEntry, 1344
$constructorList
  usedby make-databases, 1019
$constructors, 899
  usedby addTraceItem, 902
  usedby traceReply, 899
defvar, 899
$curpage, 1249
defvar, 1249
$current-directory, 5
  usedby getDirectoryList, 984
  usedby reroot, 39
  usedby restart, 16
defvar, 5
$current-line
  usedby iostat, 970
  usedby spad-short-error, 970
$currentCarrier
  usedby intloopSpadProcess, 63
$currentFrameNum, 41
  usedby close, 511
  usedby serverReadLine, 43
defvar, 41
$currentLine
  usedby ExecuteInterpSystemCommand, 31
  usedby clearCmdAll, 504
  usedby getSystemCommandLine, 834
  used by intnplisp, 34
  used by removeUndoLines, 934
  used by restart, 16
  used by setCurrentLine, 40
  used by unAbbreviateKeyword, 469
  used by updateHist, 590
  used by writeHiFi, 602
$dalymode, 663
  used by intloopReadConsole, 29
defvar, 663
$declaredMode
  used by processInteractive, 48
$defaultFortVar
  used by processInteractive, 48
$defaultFortranType, 717
defvar, 717
$defaultMsgDatabaseName, 6
used by fetchKeyedMsg, 328
used by reroot, 39
defvar, 6
$defaultPackageNamesHT
used by kcPage, 1358
$defaultSpecialCharacters, 975
defvar, 975
$depTb
local ref saveDependentsHashTable, 1022
$dependeeAlist
used by displayProperties, sayFunctionDeps, 458
used by displayProperties, 462
used by interpFunctionDepAlists, 465
$dependeeClosureAlist
used by resetWorkspaceVariables, 654
$dependentAlist
used by displayProperties, sayFunctionDeps
458
used by displayProperties, 462
used by interpFunctionDepAlists, 465
$describeOptions, 528
used by describeSpad2Cmd, 529
defvar, 528
$directory-list, 6
used by getDirectoryList, 984
used by reroot, 39
defvar, 6
$displayDroppedMap, 738
defvar, 738
$displayMsgNumber, 746
used by sayKeyedMsgLocal, 330
defvar, 746
$displayOptions, 535
used by displaySpad2Cmd, 536
defvar, 535
$displaySetValue, 748
used by set1, 809
defvar, 748
$displayStartMsgs, 749
used by restart, 16
defvar, 749
$doNotAddEmptyModeIfTrue
used by domainToGenvar, 861
used by reportOperations, 816
used by transTraceItem, 863
$docTableHash
used by dbDocTable, 1382
$docTable
used by dbAddDocTable, 1383
used by dbDocTable, 1382
$doc
used by conPageConEntry, 1344
$domPvar, 47
local def hasAtt, 424
local def hasCate1, 427
local def hasCaty1, 432
local def hasSig, 423
local ref domArg2, 422
local ref hasCaty, 421
local ref unifyStructVar, 429
used by processInteractive, 48
defvar, 47
$domainTraceNameAssoc
used by genDomainTraceName, 862
$domains
used by addTraceItem, 902
used by traceReply, 899
$domain
used by dbSearchOrder, 1356
used by kTestPred, 1386
$dotdot
used by opTran, 323
$echoLineStack
used by resetWorkspaceVariables, 654
$envHashTable,
defvar, 963
$env
local ref isDomainValuedVariable, 959
used by interpret, 52
used by reportOperations, 816
used by resetWorkspaceVariables, 654
used by showSpad2Cmd, 815
$erMsgToss
used by SpadInterpretStream, 27
used by initImPr, 367
used by phIntReportMsgs, 65
used by showMsgPos?, 358
$erase
local ref saveDependentsHashTable, 1022
local ref saveUsersHashTable, 1023
INDEX

used by deleteFile, 1043
used by reportOpsFromLispLib1, 817
used by reportOpsFromUnitDirectly1, 826
$evalDomain
  local ref evalDomain, 913
$eval
  used by interpret1, 53
  used by interpret, 52
  used by reportOperations, 816
$existingFiles
  used by clearCmdCompletely, 503
  used by resetWorkspaceVariables, 654
$expandSegments
  local def evaluateType, 916
$exposed?
  used by conPageConEntry, 1344
$exposedOnlyIfTrue
  used by dbConsHeading, 1395
  used by dbShowCons1, 1390
  used by dbShowCons, 1388
$e
  local ref isDomainValuedVariable, 959
  used by clearCmdParts, 505
  used by describeSpad2Cmd, 529
  used by displaySpad2Cmd, 536
  used by interpFunctionDepAlists, 465
  used by ncTopLevel, 23
  used by parseAndInterpret, 46
  used by processInteractive1, 50
  used by recordAndPrint, 54
  used by resetWorkspaceVariables, 654
  used by restoreHistory, 597
  used by showSpad2Cmd, 815
  used by systemSpadCommand, 448
  used by whatSpad2Cmd, 940
$filep
  used by setOutputAlgebra, 763
  used by setOutputFormula, 790
  used by setOutputFortran, 770
  used by setOutputHtml, 781
  used by setOutputMathml, 777
  used by setOutputOpenMath, 786
  used by setOutputTex, 797
$findfile
  used by readSpad2Cmd, 642
$floatok
  used by lineoftoks, 111
  used by scanKeyTr, 120
  used by scanNumber, 132
  used by scanSpace, 129
  used by scanString, 130
  used by scanWord, 126
$fn
  used by SpadInterpretStream, 27
  used by toFile?, 363
$forceDatabaseUpdate
  local ref loadLib, 1036
  local ref updateDatabase, 1017
  used by localDatabase, 1014
$formulaFormat, 789
  used by setOutputFormula, 791
defvar, 789
$formulaOutputFile, 789
defvar, 789
$formulaOutputStream
  used by setOutputFormula, 790
defvar, 790
$fortenStream
  used by setOutputStream, 790
defvar, 790
$fortIndef, 715
defvar, 715
$fortInts2Floats, 714
defvar, 714
$fortLength, 716
  used by setOutputFortran, 770
defvar, 716
$fortPersistence, 756
  used by describeFortPersistence, 757
defvar, 756
$fortVar
  used by processInteractive, 48
$fortenArrayStartingIndex, 722
defvar, 722
$fortenDirectory, 726
  used by describeSetFortDir, 727
defvar, 726
$fortenFormat, 769
  used by setOutputFortran, 770
defvar, 769
$fortenLibraries, 728
  used by describeSetLinkerArgs, 729
used by setLinkerArgs, 728
defvar, 728
$fortranOptimizationLevel, 721
defvar, 721
$fortranOutputFile, 769
used by setOutputFortran, 770
defvar, 769
$fortranOutputStream
used by setOutputFortran, 770
$fortranPrecision, 718
defvar, 718
$fortranSegment, 720
defvar, 720
$fortranTmpDir, 723
used by describeSetFortTmpDir, 725
used by setFortTmpDir, 724
defvar, 723
$fractionDisplayType, 773
defvar, 773
$frameAlist, 41
used by serverReadLine, 43
defvar, 41
$frameMessages, 741
used by clearCmdAll, 504
used by displayProperties, 462
used by updateFromCurrentInterpreterFrame, local ref basicLookup, 1076
558
defvar, 741
$frameNumber, 41
used by serverReadLine, 43
defvar, 41
$frameRecord, 921
used by clearCmdAll, 503
used by clearFrame, 929
used by recordFrame, 923
used by undoSteps, 930
defvar, 921
$freeVars
used by processInteractive, 48
$fromSpadTrace
used by spadTrace, 877
$fullScreenSysVars, 730
defvar, 730
$functionTable, 502
used by clearCmdCompletely, 503
used by resetWorkspaceVariables, 654
defvar, 502
$funnyBacks, 1297
used by unescapeStringsInForm, 61
defvar, 1297
$funnyQuote, 1297
used by unescapeStringsInForm, 61
defvar, 1297
$f
used by lineoftoks, 111
used by nextline, 113
$genValue, 51
used by interpret1, 53
used by interpret, 52
used by reportOperations, 816
defvar, 51
$giveExposureWarning, 739
defvar, 739
$globalExposureGroupAlist, 670
local ref isExposedConstructor, 820
used by setExposeAddGroup, 699
used by setExposeDropGroup, 703
defvar, 670
$hashOp0
local ref basicLookup, 1076
$hashOp1
$hashOpApply
local ref basicLookup, 1076
$hashOpSet
local ref basicLookup, 1076
$hashSeg
local ref basicLookup, 1076
$highlightAllowed, 742
defvar, 742
$historyDirectory, 579
used by histInputFileName, 580
defvar, 579
$historyDisplayWidth, 731, 1324
defvar, 731, 1324
$historyFileType, 579
defvar, 579
$hope
local def hasCate, 433
local def ofCategory, 419
local def unifyStructVar, 429
$htLineList, 1249
INDEX

defvar, 1249
$defvar, 1249
$htmlFormat, 780
used by setOutputHtml, 781
defvar, 780
$htmlOutputFile, 780
used by setOutputHtml, 781
defvar, 780
$htmlOutputStream
used by setOutputHtml, 781
$imPrGuys, 358
defvar, 358
$imPrTagGuys, 367
used by initImPr, 367
defvar, 367
$inRetract
used by processInteractive, 48
$inclAssertions
used by SpadInterpretStream, 27
used by assertCond, 94
used by ifCond, 87
$includeUnexposed?
used by getBrowseDatabase, 1090
$infovec
used by dbAddChainDomain, 1386
used by dbSearchOrder, 1356
$inputPromptType, 746
used by mkprompt, 40
defvar, 746
$inputStream
used by npFirstTok, 143
used by npListAndRecover, 189
used by npMoveTo, 191
used by npNext, 145
used by npParse, 141
used by npRestore, 152
used by npState, 212
$insideApplication
used by pf2Sex, 299
used by pfApplication2Sex, 306
used by pfDefinition2Sex, 315
$insideRule
used by pf2Sex1, 302
used by pf2Sex, 299
used by pfApplication2Sex, 306
used by pfLhsRule2Sex, 318
used by pfLiteral2Sex, 304
used by pfOp2Sex, 308
used by pfRhsRule2Sex, 319
$insideSEQ
used by pf2Sex1, 302
used by pf2Sex, 299
used by pfSequence2Sex, 310
$instantCanCoerceCount
used by processInteractive, 48
$instantCoerceCount
used by processInteractive, 48
$instantMmCondCount
used by processInteractive, 48
$instantRecord
used by processInteractive, 48
$intCoerceFailure, 30
used by InterpExecuteSpadSystemCommand, 30
used by intloopSpadProcess, 63
defvar, 30
$intRestart, 24
used by intloop, 24
defvar, 24
$intSpadReader, 30
used by InterpExecuteSpadSystemCommand, 30
used by intloopSpadProcess, 63
defvar, 30
$intTopLevel, 24
used by intloop, 24
defvar, 24
$internalHistoryTable
used by clearCmdAll, 503
used by createCurrentInterpreterFrame, 557
used by readHiFi, 601
used by restoreHistory, 597
used by saveHistory, 596
used by setHistoryCore, 585
used by updateFromCurrentInterpreterFrame, 558
used by writeHiFi, 602
$interpOnly, 47
used by processInteractive, 48
defvar, 47
$interpreterFrameName
used by clearCmdAll, 504
used by closeInterpreterFrame, 562
used by createCurrentInterpreterFrame, 557  used by SpadInterpretStream, 27
used by displayExposedGroups, 705  used by getPosStL, 356
used by displayProperties, 462  $lastUntraced
used by histFileName, 580  used by getMapSubNames, 869
used by histInputFileName, 580  used by trace1, 848
used by initializeInterpreterFrameRing, 555  used by untraceMapSubNames, 873
used by mkprompt, 40  used by untrace, 862
used by setExposeAddConstr, 701  $letAssoc
used by setExposeAddGroup, 699  used by breaklet, 905
used by setExposeDropConstr, 704  used by letPrint2, 887
used by setExposeDropGroup, 703  used by letPrint3, 888
used by updateFromCurrentInterpreterFrame, 558  used by letPrint, 885
$interpreterFrameRing  used by spadTrace, 877
used by addNewInterpreterFrame, 561  used by spadUntrace, 896
used by changeToNamedInterpreterFrame, $libQuiet
  560  used by SpadInterpretStream, 27
used by closeInterpreterFrame, 562  $library-directory-list, 7
used by displayFrameNames, 563  used by getDirectoryList, 984
used by findFrameInRing, 559  used by reroor, 39
defvar, 7
used by frameNames, 556  $linearFormatScripts, 793
used by importFromFrame, 563  defvar, 793
used by initializeInterpreterFrameRing, 555  $linelength, 774
used by nextInterpreterFrame, 560  used by displayOperationsFromLisplib, 821
used by previousInterpreterFrame, 561  used by displaySetOptionInformation, 655
used by updateCurrentInterpreterFrame, 559  used by displaySetVariableSettings, 657
$interpreterTimedClasses  used by filterAndFormatConstructors, 944
used by printStorage, 56  used by justifyMyType, 60
used by printTypeAndTimeNormal, 57  used by msgOutputter, 353
used by printTypeAndTimeSaturn, 58  used by msgText, 60
used by processInteractive, 48  used by printSynonyms, 474
$interpreterTimedNames  used by reportOpsFromLisplib, 818
used by printStorage, 56  used by reportOpsFromUnitDirectly, 822
used by printTypeAndTimeNormal, 57  used by sayKeyedMsgLocal, 330
used by printTypeAndTimeSaturn, 58  used by setExposeAddConstr, 701
used by processInteractive, 48  used by setExposeAddGroup, 699
$kind  used by setExposeAdd, 698
used by conPageConEntry, 1344  used by setExposeDropConstr, 704
$lambdaType, 662  used by setExposeDropGroup, 702
defvar, 662  used by spadStartUpMsgs, 17
$lastLineInSEQ  used by traceReply, 899
used by processInteractive, 48  used by whatCommands, 942
$lastPos  used by workfilesSpad2Cmd, 950
defvar, 774
INDEX

$linepos
  used by lineoftoks, 112
  used by nextline, 113
  used by scanCheckRadix, 134
  used by scanError, 135
  used by scanS, 131
  used by scanToken, 114

$lines
  used by intloopEchoParse, 67
  used by intloopInclude0, 62
  used by nloopInclude0, 71

$ln
  used by lineoftoks, 112
  used by nextline, 113
  used by scanComment, 116
  used by scanError, 135
  used by scanEsc, 124
  used by scanExponent, 126
  used by scanNegComment, 117
  used by scanNumber, 132
  used by scanPossFloat, 121
  used by scanPunct, 118
  used by scanSpace, 129
  used by scanS, 131
  used by scanToken, 114
  used by scanW, 128
  used by spleI1, 122
  used by startsComment?, 115
  used by startsNegComment?, 117

$localExposureDataDefault, 696
  used by clearCmdCompletely, 503
  used by emptyInterpreterFrame, 556
  defvar, 696

$localExposureData, 697
  local ref isExposedConstructor, 820
  used by clearCmdCompletely, 502
  used by createCurrentInterpreterFrame, 558
  used by displayExposedConstructors, 705
  used by displayExposedGroups, 705
  used by displayHiddenConstructors, 706
  used by setExposeAddConstr, 701
  used by setExposeAddGroup, 699
  used by setExposeDropConstr, 704
  used by setExposeDropGroup, 703
  used by updateFromCurrentInterpreterFrame, 558
  defvar, 697

$localVars
  used by processInteractive, 48

$lookupDefaults
  local def oldCompLookup, 1079
  local ref basicLookupCheckDefaults, 1078
  used by clearCmdSortedCaches, 501

$LowerCaseConTb
  used by conPageFastPath, 1344

$macActive
  used by mac0ExpandBody, 225
  used by mac0MLambdaApply, 224
  used by macId, 227
  used by macroExpanded, 222

$mapList
  used by processInteractive, 48

$mapSubNameAlist
  used by ?t, 903
  used by isSubForRedundantMapName, 873
  used by saveMapSig, 853
  used by traceSpad2Cmd, 847
  used by untraceMapSubNames, 873
  used by untrace, 862

$margin, 774
  used by msgText, 60
  used by sayKeyedMsgLocal, 330
  defvar, 774

$mathTraceList
  used by coerceTraceArgs2E, 864
  used by coerceTraceFunValue2E, 867

$mathmlFormat, 775
  used by setOutputMathml, 777
  defvar, 775

$mathmlOutputFile, 775
  used by setOutputMathml, 777
  defvar, 775

$mathmlOutputStream
  used by setOutputMathml, 777
  defvar, 775

$maximumFortranExpressionLength, 720
  defvar, 720

$minivector
  used by processInteractive, 48

$minivectorCode
  used by processInteractive, 48

$minivectorNames, 47
  used by processInteractive, 48

$minivector
usedby processInteractive, 48
$mkTestFlag
  usedby recordAndPrint, 54
$mkTestInputStack
  usedby updateHist, 590
$mkTestOutputType
  usedby recordAndPrint, 54
$msgAlist, 326
  usedby resetWorkspaceVariables, 654
  usedby spadStartUpMsgs, 17
defvar, 326
$msgDatabaseName, 7
  usedby resetWorkspaceVariables, 654
  usedby reroot, 39
defvar, 7
$msgDatabase
  usedby resetWorkspaceVariables, 654
$msgdbNoBlanksAfterGroup, 328
defvar, 328
$msgdbNoBlanksBeforeGroup, 328
defvar, 328
$msgdbPrims, 327
  usedby fixObjectForPrinting, 456
defvar, 327
$msgdbPunct, 327
defvar, 327
$multiVarPredicateList
  usedby pfRule2Sex, 318
  usedby ruleLhsTran, 322
  usedby rulePredicateTran, 319
$nagEnforceDouble, 758
defvar, 758
$nagHost, 754
  usedby ncBug, 368
  usedby ncHardError, 352
  usedby ncSoftError, 351
defvar, 26
$newline, 1324
defvar, 1324
$newspad
  usedby ncTopLevel, 23
$noEvalTypeMsg, 919
  local ref throwEvalTypeMsg, 919
  usedby topLevelInterpEval, 1372
defvar, 919
$noParseCommands, 444
  usedby doSystemCommand, 446
defvar, 444
$noRepList
  usedby processMsgList, 369
defvar, 374
$noSubsumption, 962
defvar, 962
$nopos, 26
  usedby SpadInterpretStream, 27
  usedby npcSourcePosition, 238
  usedby poNoPosition, 414
defvar, 26
$npPParg
  usedby npPPff, 210
$npTokToNames, 204
  usedby npId, 205
defvar, 204
$numberOfEquations, 1226
defvar, 1226
$numericFailure, 1080
defvar, 1080
$n
  usedby lineoftoks, 112
  usedby nextline, 113
  usedby scanCheckRadix, 134
  usedby scanComment, 116

$newConlist
  usedby library, 1013
$newcompErrorCount, 26
  usedby SpadInterpretStream, 27
  usedby ncBug, 368
  usedby ncHardError, 352
  usedby ncSoftError, 351
  usedby pickleStream, 137
  usedby pickleStream, 137
defvar, 26
$ncConList, 25
  usedby SpadInterpretStream, 27
  usedby intloopSpadProcess, 63
  usedby ncConversationPhase, 66
  usedby processKeyedError, 353
  usedby processMsgList, 369
defvar, 25
used by `scanError`, 135
used by `scanEscape`, 135
used by `scanEsc`, 124
used by `scanExponent`, 126
used by `scanNegComment`, 117
used by `scanNumber`, 132
used by `scanPossFloat`, 121
used by `scanPunct`, 118
used by `scanSpace`, 129
used by `scanString`, 130
used by `scanS`, 131
used by `scanToken`, 114
used by `scanW`, 128
used by `spleI`, 123
used by `startsComment?`, 115
used by `startsNegComment?`, 117

$okToExecuteMachineCode
used by `SpadInterpretStream`, 27

$oldBreakMode, 1080
defvar, 1080

$oldHistoryFileName, 578
used by `oldHistFileName`, 580
used by `restoreHistory`, 597
defvar, 578

$oldline, 449
used by `commandErrorIfAmbiguous`, 470
used by `commandErrorMessage`, 450
defvar, 449

$opSysName
used by `spadStartUpMsgs`, 17

$openMathFormat, 784
used by `setOutputOpenMath`, 786
defvar, 784

$openMathOutputFile, 785
used by `setOutputOpenMath`, 786
defvar, 785

$openMathOutputStream
used by `setOutputOpenMath`, 786

$openServerIfTrue, 8
used by `restart`, 15
used by `spad-save`, 990
defvar, 8

$operationNameList
used by `resetWorkspaceVariables`, 654

$optionAlist, 846
used by `trace1`, 849
defvar, 846

$options
used by `abbreviationsSpad2Cmd`, 483
used by `clearSpad2Cmd`, 500
used by `close`, 511
used by `frameSpad2Cmd`, 566
used by `historySpad2Cmd`, 583
used by `history`, 582
used by `library`, 1013
used by `readSpad2Cmd`, 642
used by `reportOpsFromLispLib`, 818
used by `reportOpsFromUnitDirectly`, 822
used by `restoreHistory`, 597
used by `showSpad2Cmd`, 815
used by `systemCommand`, 448
used by `trace1`, 848
used by `undo`, 922
used by `workfilesSpad2Cmd`, 950
used by `zsystemdevelopment1`, 954

$op
used by `dbConstructorDoc.gn`, 1381
used by `dbConstructorDoc`, 1381
used by `dbGetDocTable`, 1385
used by `displayMacro`, 454
used by `displayType`, 460
used by `displayValue`, 459
used by `processInteractive`, 48

$outputLibraryName
used by `setOutputLibrary`, 664

$outputLines
used by `printTypeAndTimeNormal`, 57

$outputList
used by `processMsgList`, 369
used by `queueUpErrors`, 372

$outputMode
used by `recordAndPrint`, 54, 55

$packages
used by `addTraceItem`, 902
used by `traceReply`, 899

$pfMacros, 97
used by `clearMacroTable`, 504
used by `clearParserMacro`, 453
used by `displayParserMacro`, 464
used by `getParserMacroNames`, 453
used by `macroDefine`, 230
used by `macroGetName`, 226
used by mac0Get, 228  
used by mac0MLambdaApply, 224  
used by macApplication, 223  
used by macLambda,mac, 229  
used by macLambda, 229  
used by macWhere,mac, 228  
used by macWhere, 228  
defvar, 97  
$plainRTspecialCharacters, 978  
used by setOutputCharacters, 767  
defvar, 978  
$plainSpecialCharacters0, 976  
defvar, 976  
$plainSpecialCharacters1, 976  
defvar, 976  
$plainSpecialCharacters2, 977  
defvar, 977  
$plainSpecialCharacters3, 977  
defvar, 977  
$posActive  
used by mac0ExpandBody, 225  
used by mac0MLambdaApply, 224  
used by macId, 227  
used by macroExpanded, 222  
$preLength, 355  
used by getPreStL, 355  
used by makeLeaderMsg, 377  
used by makeMsgFromLine, 371  
used by processChPosesForOneLine, 376  
used by tabbing, 362  
defvar, 355  
$predicateList  
used by pfRule2Sex, 318  
used by pfSuchThat2Sex, 307  
used by ruleLhsTran, 322  
$predvec  
used by dbSearchOrder, 1356  
used by kTestPred, 1386  
$prettyprint, 806  
defvar, 806  
$prevCarrier  
used by intloopSpadProcess, 63  
$previousBindings, 921  
used by clearCmdAll, 503  
used by clearFrame, 929  
used by recordFrame, 923  
defvar, 921  
$printAnyIfTrue, 735  
used by recordAndPrint, 55  
defvar, 735  
$printFortranDecs, 716  
defvar, 716  
$printLoadMsgs, 736  
local ref loadLibNoUpdate, 1037  
local ref loadLib, 1036  
used by restart, 16  
defvar, 736  
$printMsgsToFile, 740  
used by sayKeyedMsgLocal, 330  
defvar, 740  
$printStatisticsSummaryIfTrue, 749  
used by recordAndPrint, 54  
defvar, 749  
$printStorageIfTrue  
used by recordAndPrint, 55  
$printTimeIfTrue, 751  
used by printTypeAndTimeNormal, 57  
used by printTypeAndTimeSaturn, 58  
used by recordAndPrint, 55  
defvar, 751  
$printTypeIfTrue, 752  
used by printTypeAndTimeNormal, 57  
used by printTypeAndTimeSaturn, 58  
used by recordAndPrint, 55  
defvar, 752  
$printVoidIfTrue, 753  
used by recordAndPrint, 55  
defvar, 753  
$promptMsg, 26  
used by SpadInterpretStream, 27  
defvar, 26  
$quadSymbol  
local ref evaluateType1, 918  
used by reportOperations, 816  
$quitCommandType, 800  
used by pquitSpad2Cmd, 634  
used by quitSpad2Cmd, 638  
defvar, 800  
$quitTag, 18  
used by runspad, 19  
defvar, 18  
$quotedOpList
used by \texttt{ioclear}, 972
used by \texttt{iostat}, 970
used by \texttt{ncTopLevel}, 23
used by \texttt{parseAndInterpret}, 46
\texttt{$\$\text{specialCharacterAlist}}}, 979
used by \texttt{setOutputCharacters}, 767
used by \texttt{specialChar}, 980
defvar, 979
\texttt{$\$\text{specialCharacters}}}, 979
used by \texttt{setOutputCharacters}, 767
used by \texttt{specialChar}, 980
defvar, 979
\texttt{$\$\text{stack}}}
used by \texttt{npListAndRecover}, 189
used by \texttt{npListofFun}, 221
used by \texttt{npList}, 155
used by \texttt{npParse}, 141
used by \texttt{npPop1}, 144
used by \texttt{npPop2}, 144
used by \texttt{npPop3}, 144
used by \texttt{npPushId}, 209
used by \texttt{npPush}, 143
used by \texttt{npRestore}, 152
used by \texttt{npState}, 212
used by \texttt{npZeroOrMore}, 177
\texttt{$\$\text{stepNo}}}
used by \texttt{intloopSpadProcess}, 63
\texttt{$\$\text{stok}}
used by \texttt{npConstTok}, 184
used by \texttt{npDDInfKey}, 208
used by \texttt{npDollar}, 184
used by \texttt{npEnclosed}, 211
used by \texttt{npEqKey}, 145
used by \texttt{npEqPeek}, 152
used by \texttt{npFirstTok}, 143
used by \texttt{npId}, 205
used by \texttt{npInfKey}, 208
used by \texttt{npInfixOperator}, 160
used by \texttt{npInfixOp}, 161
used by \texttt{npMissing}, 151
used by \texttt{npParenthesize}, 215
used by \texttt{npParse}, 141
used by \texttt{npPrefixColon}, 161
used by \texttt{npPushId}, 209
used by \texttt{npRecoverTrap}, 190
used by \texttt{npTrap}, 212
used by \texttt{sysSpecificErrorHere}, 192
\texttt{$\$\text{streamCount}}}, 801
used by \texttt{coerceSpadArgs2E}, 865
used by \texttt{coerceSpadFunValue2E}, 868
used by \texttt{describeStreamsCalculate}, 802
used by \texttt{setStreamsCalculate}, 802
defvar, 801
\texttt{$\$\text{streamsShowAll}}}, 803
defvar, 803
\texttt{$\$\text{syscommands}}}, 444
used by \texttt{newHelpSpad2Cmd}, 573
used by \texttt{systemCommand}, 448
used by \texttt{unAbbreviateKeyword}, 469
defvar, 444
\texttt{$\$\text{systemCommandFunction}}
used by \texttt{SpadInterpretStream}, 27
used by \texttt{intloopProcess}, 62
used by \texttt{ncloopCommand}, 478
\texttt{$\$\text{systemCommands}}}, 443
used by \texttt{synonymsForUserLevel}, 833
used by \texttt{systemCommand}, 448
used by \texttt{unAbbreviateKeyword}, 469
used by \texttt{whatCommands}, 942
defvar, 443
\texttt{$\$\text{systemType}}}, 1225
defvar, 1225
\texttt{$\$\text{sz}}
used by \texttt{lineoftoks}, 112
used by \texttt{nextline}, 112
used by \texttt{scanComment}, 116
used by \texttt{scanEsc}, 124
used by \texttt{scanExponent}, 126
used by \texttt{scanNegComment}, 117
used by \texttt{scanNumber}, 132
used by \texttt{scanPossFloat}, 121
used by \texttt{scanS}, 131
used by \texttt{scanW}, 128
used by \texttt{sple1I}, 123
used by \texttt{startsComment?}, 115
used by \texttt{startsNegComment?}, 117
\texttt{$\$\text{testingErrorPrefix}}}, 326
defvar, 326
\texttt{$\$\text{testingSystem}}}, 750
defvar, 750
\texttt{$\$\text{texFormatting}}}, 327
used by \texttt{sayKeyedMsg}, 329
INDEX

1481

defvar, 327
	(texFormat, 795
		used by setOutputTex, 797
defvar, 795
$texOutputFile, 796
	used by setOutputTex, 797
defvar, 796
$timedNameStack
	used by interpretTopLevel, 51
defvar, 363
	 StoWhereGuys, 363
defvar, 363
$tokenCommands, 475
	used by doSystemCommand, 446
defvar, 475
$topicHash
	used by write-warmdata, 1032
$traceletflag
	used by tracelet, 904
$ttok
	used by npEqKey, 145
	used by npEqPeek, 152
	used by npFirstTok, 143
	used by npId, 205
	used by npInfKey, 208
	used by npInfixOp, 161
	used by npParse, 141
	used by npPushId, 209
$underbar, 586

defvar, 586
$undoFlag, 921
	used by recordFrame, 923
defvar, 921
$useBFasDefault
	used by float2Sex, 305
$useEditorForShowOutput
	used by reportOpsFromLisplib0, 817
	used by reportOpsFromUnitDirectly0, 821
defvar, 794
$useFullScreenHelp, 732
	used by newHelpSpad2Cmd, 573
defvar, 732
$useInternalHistoryTable, 579
	used by clearCmdAll, 503
	used by initHist, 581
	used by readHiFi, 601
	used by restoreHistory, 597
	used by saveHistory, 596
	used by setHistoryCore, 585
	used by writeHiFi, 602
defvar, 579
$useIntrinsicFunctions, 719

defvar, 719
$usersTb
	local ref saveUsersHashTable, 1023
$variableNumberAlist
	used by clearCmdAll, 503
$whatOptions, 939
	used by reportWhatOptions, 941
	used by whatSpad2Cmd, 940
defvar, 939
$whereCacheList
	used by processInteractive, 48
$which
  used by dbGetDocTable, ln, 1384
  used by dbGetDocTable, 1385
$writifyComplained
  used by writifyComplain, 606
  used by writify, 610
$xdatabase
  used by clearCmdCompletely, 502

abbreviate
  called by traceReply, 899
abbreviation?
  called by abbreviationsSpad2Cmd, 483
abbreviations, 483
  calls abbreviationsSpad2Cmd, 483
defun, 483
abbreviations help page, 481
  manpage, 481
abbreviationsSpad2Cmd, 483
  called by abbreviations, 483
calls abbQuery, 483
calls deldatabase, 483
calls exit, 483
calls helpSpad2Cmd, 483
calls listConstructorAbbreviations, 483
calls mkUserConstructorAbbreviation, 483
calls opOf, 483
calls qcar, 483
calls qcdr, 483
calls sayKeyedMsg, 483
calls seq, 483
calls setdatabase, 483
calls size, 483
uses $options, 483
defun, 483
acot, 1104
defun, 1104
acoth, 1107
defun, 1107
acsc, 1105
defun, 1105
acsch, 1106
defun, 1106
addassoc
  called by saveMapSig, 853
called by traceReply, 899
addBinding, 963
calls addBindingInteractive, 963
calls getProplist, 963
calls hput, 963
uses $InteractiveMode, 963
uses $envHashTable, 963
defun, 963
addBindingInteractive, 967
called by addBinding, 963
calls assq, 967
defun, 967
addInputLibrary, 668
called by setInputLibrary, 667
calls dropInputLibrary, 668
uses input-libraries, 668
defun, 668
addNewInterpreterFrame, 561
called by frameSpad2Cmd, 566
called by serverReadLine, 42
calls $erase, 561
calls boot-equal, 561
calls emptyInterpreterFrame, 561
calls framename, 561
calls histFileName, 561
calls initHistList, 561
calls throwKeyedMsg, 561
calls updateCurrentInterpreterFrame, 561
calls updateFromCurrentInterpreterFrame, 561
uses $interpreterFrameRing, 561
defun, 561
addoperations, 1007
called by localnrlib, 1015
calls getdatabase, 1007
uses *operation-hash*, 1007
defun, 1007
addTraceItem, 902
called by traceReply, 899
INDEX

calls constructor?, 902
calls devaluate, 902
calls isDomainOrPackage, 902
calls isDomain, 902
uses $constructors, 902
uses $domains, 902
uses $packages, 902
defun, 902
aldorTrace
called by spadTrace, 876
algCoerceInteractive, 1066
defun, 1066
allConstructors, 1032
called by make-databases, 1019
called by write-browsedb, 1029
uses *allconstructors*, 1032
defun, 1032
allocate
called by init-memory-config, 33
allocate-contiguous-pages
called by init-memory-config, 33
allocate-relocatable-pages
called by init-memory-config, 33
allOperations, 1033
called by apropos, 945
uses *allOperations*, 1033
uses *operation-hash*, 1033
defun, 1033
alqlGetKindString, 1089
calls dbPart, 1089
calls substring, 1089
defun, 1089
alqlGetOrigin, 1088
calls charPosition, 1088
calls dbPart, 1088
calls substring, 1088
defun, 1088
alqlGetParams, 1089
calls charPosition, 1089
calls dbPart, 1089
calls substring, 1089
defun, 1089
alreadyOpened?, 363
called by msgOutputter, 353
calls msgImPr?, 363
defun, 363
ancestorsOf
called by kcaPage, 1362
called by originsInOrder, 1382
ancolsU16, 1060
defmacro, 1060
ancolsU32, 1061
defmacro, 1061
ancolsU8, 1058
defmacro, 1058
aurowsU16, 1059
defmacro, 1059
aurowsU32, 1061
defmacro, 1061
aurowsU8, 1058
defmacro, 1058
apropos, 945
called by whatSpad2Cmd, 940
calls allOperations, 945
calls downcase, 945
calls exit, 945
calls filterListOfStrings, 945
calls msort, 945
calls sayAsManyPerLineAsPossible, 945
calls sayKeyedMsg, 945
calls sayMessage, 945
calls seq, 945
defun, 945
aref2U16, 1059
defmacro, 1059
aref2U32, 1060
defmacro, 1060
aref2U8, 1057
defmacro, 1057
as-insert
called by spadTrace, 876
asec, 1105
defun, 1105
asech, 1107
defun, 1107
asharpConstructorName?
called by kcPage, 1358
assertCond, 94
called by incLude1, 80
calls MakeSymbol, 94
calls incCommandTail, 94
uses *whitespace*, 94
calls vecp, 1078
local ref $lookupDefaults, 1078
defun, 1078
basicMatch?
called by stringMatches?, 1090
basicStringize, 1261
defun, 1261
bcAbbTable
called by dbShowCons1, 1389
bcComplexLimit, 1223
calls htInitPage, 1223
calls htMakePage, 1223
calls htShowPage, 1224
uses $EmptyMode, 1224
defun, 1223
bcComplexLimitGen, 1225
calls bcFinish, 1225
calls htpButtonValue, 1225
calls htpLabelInputString, 1225
calls strconc, 1225
defun, 1225
bcConform
called by reportAO, 1348
called by reportCategory, 1347
bcConPredTable
called by dbShowConditions, 1395
called by kePage, 1352
called by reportCategory, 1347
bcConTable
called by dbShowCons1, 1390
bcCreateVariableString, 1233
called by bcMakeEquations, 1234
called by bcMakeLinearEquations, 1234
defun, 1233
bcDefiniteIntegrate, 1190
calls htInitPage, 1190
calls htMakePage, 1190
calls htShowPage, 1190
uses $EmptyMode, 1191
defun, 1190
bcDefiniteIntegrateGen, 1192
calls bcGen, 1192
calls htpButtonValue, 1192
calls htpLabelInputString, 1192
calls strconc, 1192
defun, 1192
bcDifferentiate, 1195
calls htInitPage, 1195
calls htMakeDoneButton, 1195
calls htMakePage, 1195
calls htShowPage, 1195
uses $EmptyMode, 1195
defun, 1195
bcDifferentiateGen, 1196
calls bcError, 1196
calls bcGen, 1196
calls bcString2WordList, 1196
calls bcwords2liststring, 1196
calls htpLabelInputString, 1196
calls length, 1196
calls strconc, 1196
defun, 1196
bcDraw, 1197
calls bcHt, 1197
calls htInitPage, 1197
calls htShowPage, 1197
defun, 1197
bcDraw2Dfun, 1198
calls htInitPage, 1198
calls htMakePage, 1198
calls htShowPage, 1198
uses $EmptyMode, 1198
defun, 1198
bcDraw2DfunGen, 1199
calls bcDrawIt2, 1199
calls bcFinish, 1199
calls htpLabelInputString, 1199
calls strconc, 1199
defun, 1199
bcDraw2Dpar, 1200
calls htInitPage, 1200
calls htMakePage, 1200
calls htShowPage, 1200
uses $EmptyMode, 1200
defun, 1200
bcDraw2DparGen, 1201
calls bcDrawIt2, 1201
calls bcFinish, 1201
calls htpLabelInputString, 1201
calls strconc, 1201
defun, 1201
bcDraw2DSolve, 1202
calls htInitPage, 1202
calls htMakeDoneButton, 1202
calls htMakePage, 1202
calls htShowPage, 1202
uses $EmptyMode, 1202
defun, 1202
bcDraw2DSolveGen, 1203
calls bcFinish, 1203
calls htpLabelInputString, 1203
calls strconc, 1203
defun, 1203
bcDraw3Dfun, 1204
calls htInitPage, 1204
calls htMakePage, 1204
calls htShowPage, 1204
uses $EmptyMode, 1204
defun, 1204
bcDraw3DfunGen, 1205
calls bcDrawIt2, 1205
calls bcFinish, 1205
calls htpLabelInputString, 1205
calls strconc, 1205
defun, 1205
bcDraw3Dpar, 1206
calls htInitPage, 1206
calls htMakePage, 1206
calls htShowPage, 1206
uses $EmptyMode, 1206
defun, 1206
bcDraw3Dpar1, 1208
calls htInitPage, 1208
calls htMakeDoneButton, 1208
calls htMakePage, 1208
calls htShowPage, 1208
uses $EmptyMode, 1208
defun, 1208
bcDraw3Dpar1Gen, 1209
calls bcDrawIt2, 1209
calls bcFinish, 1210
calls htpLabelInputString, 1209
calls strconc, 1210
defun, 1209
bcDraw3DparGen, 1207
calls bcDrawIt2, 1207
calls bcFinish, 1207
calls htpLabelInputString, 1207
calls strconc, 1207
defun, 1207
bcDrawIt, 1247
calls strconc, 1247
defun, 1247
bcDrawIt2, 1189
calledby bcDraw2DfunGen, 1199
calledby bcDraw2DparGen, 1201
calledby bcDraw3DfunGen, 1205
calledby bcDraw3Dpar1Gen, 1209
calledby bcDraw3DparGen, 1207
calls bcMatrixGen, 1189
defun, 1189
bcDrawIt2, 1243
calledby bcComplexLimitGen, 1225
calledby bcDraw2DSolveGen, 1203
calledby bcDraw2DfunGen, 1199
calledby bcDraw2DparGen, 1201
calledby bcDraw3DfunGen, 1205
calledby bcDraw3Dpar1Gen, 1210
calledby bcDraw3DparGen, 1207
calledby bcLinearMatrixGen, 1240
calledby bcLinearSolveEqnsGen, 1242
calledby bcRealLimitGen1, 1223
calledby bcRealLimitGen, 1222
calledby bcSeriesExpansionGen, 1212
calledby bcSeriesGen, 1219
calledby bcSolveEquations, 1236
defun, 1243
bcGen, 1244, 1245
calledby bcDefiniteIntegrateGen, 1192
calledby bcDifferentiateGen, 1196
calledby bcGenExplicitMatrix, 1187
calledby bcIndefiniteIntegrateGen, 1190
calledby bcInputMatrixByFormulaGen, 1185
calledby bcLinearMatrixGen, 1240
calledby bcProductGen, 1195
INDEX

called by bcSumGen, 1194
calls htInitPage, 1244, 1245
calls htMakePage, 1244, 1245
calls htShowPage, 1244, 1245
calls strconc, 1244, 1245
defun, 1244, 1245
bcGenEquations, 1242
called by bcLinearSolveEqnsGen, 1242
called by bcSolveEquations, 1236
calls bcwords2liststring, 1243
calls strconc, 1242
defun, 1242
bcGenExplicitMatrix, 1187
calls bcGen, 1187
calls bcMatrixGen, 1187
calls htpInputAreaAlist, 1187
calls htpProperty, 1187
calls htpSetProperty, 1187
defun, 1187
bcHt, 1260
called by bcDraw, 1197
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1186
called by bcvspace, 1246
called by kdPageInfo, 1345
defun, 1260
bcHtMakeButton, 1287
defun, 1287
bcIndefiniteIntegrate, 1189
calls htInitPage, 1189
calls htMakePage, 1189
calls htShowPage, 1189
uses $EmptyMode, 1189
defun, 1189
bcIndefiniteIntegrateGen, 1190
calls bcGen, 1190
calls htpLabelInputString, 1190
calls strconc, 1190
defun, 1190
bcInputEquations, 1231
called by bcLinearSolveEqns1, 1230
called by bcSolveSingle, 1229
called by bcSystemSolveEqns1, 1230
calls bcHt, 1231
calls bcMakeEquations, 1231
calls bcMakeLinearEquations, 1231
calls bcMakeUnknowns, 1231
calls htInitPage, 1231
calls htMakeDoneButton, 1231
calls htMakePage, 1231
calls htProperty, 1231
calls htSay, 1231
calls htShowPage, 1231
calls htpPropertyList, 1231
calls htpSetProperty, 1231
calls objValUnwrap, 1231
calls parse-integer, 1231
calls strconc, 1231
calls stringimage, 1231
uses $EmptyMode, 1232
uses $bcParseOnly, 1232
defun, 1231
bcInputEquationsEnd, 1235
calls systemError, 1235
defun, 1235
bcInputExplicitMatrix, 1185
called by bcHt, 1186
called by bcInputEquations, 1186
called by bcInputDoneButton, 1186
called by bcInputPage, 1186
called by bcInputSetProperty, 1186
called by bcInputShowPage, 1186
called by htpLabelInputString, 1185
called by htpLabelSpadValue, 1185
called by htpPropertyList, 1186
called by htpSetProperty, 1186
called by length, 1185
called nreverse0, 1186
called objValUnwrap, 1185
called parse-integer, 1185
called strconc, 1186
called stringimage, 1186
uses $EmptyMode, 1186
uses $bcParseOnly, 1186
defun, 1185
bcInputMatrixByFormula, 1183
called by bcHt, 1183
called by bcInputDoneButton, 1183
called by bcInputPage, 1183
called by bcInputShowPage, 1183
called by htpLabelInputString, 1183
called by htpLabelSpadValue, 1183
called by htpSetProperty, 1183
calls objValUnwrap, 1183
calls parse-integer, 1183
uses $bcParseOnly, 1184
defun, 1183
bcInputMatrixByFormulaGen, 1185
calls bcGen, 1185
calls htpLabelInputString, 1185
calls htpProperty, 1185
calls strconc, 1185
calls stringimage, 1185
defun, 1185
bcInputSolveInfo, 1230
calls htInitPage, 1230
calls htMakePage, 1230
calls htShowPage, 1230
calls htpInputAreaList, 1230
calls htpPropertyList, 1230
calls htpSetProperty, 1230
defun, 1230
bcIssueHt, 1261
defun, 1261
bcLaurentSeries, 1215
calls htInitPage, 1215
calls htMakePage, 1215
calls htShowPage, 1215
uses $EmptyMode, 1216
defun, 1215
bcLaurentSeriesGen, 1219
calls bcSeriesGen, 1219
defun, 1219
bcLimit, 1220
calls htInitPage, 1220
calls htMakePage, 1220
calls htShowPage, 1220
uses $EmptyMode, 1220
defun, 1220
bcLinearExtractMatrix, 1238
calls htpInputAreaAlist, 1238
defun, 1238
bcLinearMatrixGen, 1240
calledby bcLinearMatrixHomo, 1240
calledby bcLinearMatrixInhomoGen, 1239
calls bcFinish, 1240
calls bcGen, 1240
calls bcMatrixGen, 1240
calls bcMkFunction, 1240
calls bcVectorGen, 1240
calls htpInputAreaGen, 1240
calls strconc, 1240
defun, 1240
bcLinearSolve, 1227
calls htInitPage, 1227
calls htMakePage, 1227
calls htShowPage, 1227
uses $EmptyMode, 1227
defun, 1227
bcLinearSolveEqns, 1228
calls htInitPage, 1228
calls htMakeDoneButton, 1228
calls htMakePage, 1228
calls htShowPage, 1228
uses $EmptyMode, 1228
defun, 1228
bcLinearSolveEqns1, 1230
calls bcInputEquations, 1230
calls htpSetProperty, 1230
defun, 1230
bcLinearSolveEqnsGen, 1242
calls bcFinish, 1242
calls bcGenEquations, 1242
calls bcString2WordList, 1242
calls bcwords2liststring, 1242
calls htpInputAreaAlist, 1242
calls htpLabelInputString, 1242
defun, 1242
bcLinearSolveMatrix, 1237
calls bcReadMatrix, 1237
defun, 1237
bcLinearSolveMatrix1, 1237
calls htInitPage, 1237
calls htMakePage, 1237
calls htShowPage, 1237
uses $EmptyMode, 1237
defun, 1237
bcLinearSolveMatrixHomo, 1240
calls bcLinearMatrixGen, 1240
defun, 1240
bcLinearSolveMatrixInhomo, 1238
calledby bcLinearMatrixHomo, 1240
calledby bcLinearMatrixInhomoGen, 1239
calls htpPropertyList, 1238
calls htpProperty, 1238
calls htpSetProperty, 1238
calls strconc, 1238
calls stringimage, 1238
uses $EmptyMode, 1238
defun, 1238
bcLinearSolveMatrixInhomoGen, 1239
calls bcLinearMatrixGen, 1239
defun, 1239
bcMakeEquations, 1234
calledby bcInputEquations, 1231
calls bcCreateVariableString, 1234
calls reverse0, 1234
calls strconc, 1234
defun, 1234
bcMakeLinearEquations, 1234
calledby bcInputEquations, 1231
calls bcCreateVariableString, 1234
calls reverse0, 1234
calls strconc, 1234
defun, 1234
bcMakeUnknowns, 1233
calledby bcInputEquations, 1231
defun, 1233
bcMatrix, 1182
calls bcReadMatrix, 1182
defun, 1182
bcMatrixGen, 1188
calledby bcDrawIt2, 1189
calledby bcGenExplicitMatrix, 1187
calledby bcLinearMatrixGen, 1240
calls bcwords2liststring, 1188
calls htpProperty, 1188
calls lassoc, 1188
calls strconc, 1188
calls stringimage, 1188
calls systemError, 1188
defun, 1188
bcMkFunction, 1243
calledby bcLinearMatrixGen, 1240
defun, 1243
bcNameConTable
calledby dbShowCons1, 1389
bcNotReady, 1247
calledby bcSeriesByFormulaGen, 1215
calls htpInitPage, 1247
calls htpMakePage, 1247
calls htpShowPage, 1247
defun, 1247
bcOptional, 1245
defun, 1245
bcPred
calledby reportCategory, 1347
bcProduct, 1194
calledby htpLabelInputString, 1195
defun, 1195
bcPuiseuxSeries, 1217
calledby htpLabelInputString, 1217
defun, 1217
bcPuiseuxSeriesGen, 1219
calls bcSeriesGen, 1219
defun, 1219
bcQueryInteger
calledby finalExactRequest, 1241
calledby linearFinalRequest, 1241
bcReadMatrix, 1182
calledby bcLinearSolveMatrix, 1237
calledby bcMatrix, 1182
calls htpInitPage, 1182
calls htpMakePage, 1182
calls htpShowPage, 1182
calls htpSetProperty, 1182
defun, 1182
bcRealLimit, 1221
calls htpInitPage, 1221
calls htpMakePage, 1221
calls htpShowPage, 1221
defun, 1221
bcRealLimitGen, 1222
calls bcFinish, 1222
calls htInitPage, 1222
calls htMakePage, 1222
calls htShowPage, 1222
calls htpButtonValue, 1222
calls htpLabelInputString, 1222
calls htpSetProperty, 1222
defun, 1222
bcRealLimitGen1, 1223
calls bcFinish, 1223
calls htpProperty, 1223
calls strconc, 1223
defun, 1223
bcSadFaces, 1269
defun, 1269
bcSeries, 1210
calls htInitPage, 1210
calls htMakePage, 1210
calls htShowPage, 1210
uses $EmptyMode, 1210
defun, 1210
bcSeriesByFormula, 1212
calls htInitPage, 1212
calls htMakePage, 1212
calls htShowPage, 1213
defun, 1212
bcSeriesByFormulaGen, 1215
calls bcNotReady, 1215
defun, 1215
bcSeriesExpansion, 1211
calls htInitPage, 1211
calls htMakeDoneButton, 1211
calls htMakePage, 1211
calls htShowPage, 1211
uses $EmptyMode, 1211
defun, 1211
bcSeriesExpansionGen, 1212
calls bcFinish, 1212
calls htpLabelInputString, 1212
calls strconc, 1212
defun, 1212
bcSeriesGen, 1219
calledby bcLaurentSeriesGen, 1219
calledby bcPuiseuxSeriesGen, 1219
calledby bcTaylorSeriesGen, 1219
calls bcFinish, 1219
calls htpLabelInputString, 1219
calls strconc, 1219
defun, 1219
bcSolve, 1226
calls htInitPage, 1226
calls htMakePage, 1226
calls htShowPage, 1226
uses $EmptyMode, 1226
defun, 1226
bcSolveEquations, 1236
calledby bcSolveNumerically1, 1236
calls bcFinish, 1236
calls bcGenEquations, 1236
calls bcString2WordList, 1236
calls bcwords2liststring, 1236
calls htpButtonValue, 1236
calls htpLabelInputString, 1236
calls htpProperty, 1236
calls member, 1236
calls strconc, 1236
defun, 1236
bcSolveEquationsNumerically, 1235
calls htInitPage, 1235
calls htMakeDoneButton, 1235
calls htMakePage, 1235
calls htShowPage, 1235
calls htpPropertyList, 1235
defun, 1235
bcSolveNumerically1, 1236
calls bcSolveEquations, 1236
defun, 1236
bcSolveSingle, 1229
calls bcInputEquations, 1229
calls htpSetProperty, 1229
defun, 1229
bcString2HyString, 1244
defun, 1244
bcString2HyString2, 1244
defun, 1244
bcString2WordList, 1246
calledby bcDifferentiateGen, 1196
calledby bcLinearSolveEquationsGen, 1242
calledby bcSolveEquations, 1236
defun, 1246
bcSum, 1192
calls htInitPage, 1192
calls htMakePage, 1193
calls htShowPage, 1193
uses $EmptyMode, 1193
defun, 1192
bcSumGen, 1194
calls bcGen, 1194
calls htpLabelInputString, 1194
calls strconc, 1194
defun, 1194
bcSystemSolve, 1228
calls htInitPage, 1228
calls htMakeDoneButton, 1228
calls htMakePage, 1228
calls htShowPage, 1229
defun, 1228
bcSystemSolveEqns1, 1229
calls bcInputEquations, 1230
calls htpSetProperty, 1229
defun, 1229
bcTaylorSeries, 1213
calls htInitPage, 1213
calls htMakePage, 1213
calls htShowPage, 1213
uses $EmptyMode, 1214
defun, 1213
bcTaylorSeriesGen, 1219
calls bcSeriesGen, 1219
defun, 1219
bcUnixTable, 1397
called by dbShowCons1, 1389
calls findfile, 1397
calls htBeginTable, 1397
calls htEndTable, 1397
calls htMakePage, 1397
calls htSaySaturn, 1397
calls htSay, 1397
calls namestring, 1397
calls stringimage, 1397
defun, 1397
bcVectorGen, 1246
called by bcLinearMatrixGen, 1240
calls bcwords2liststring, 1246
defun, 1246
bcvspace, 1246
calls bcHt, 1246
defun, 1246
bcwords2liststring, 1246
called by bcDifferentiateGen, 1196
called by bcGenEquations, 1243
called by bcLinearSolveEqnsGen, 1242
called by bcMatrixGen, 1188
called by bcSolveEquations, 1236
called by bcVectorGen, 1246
calls strconc, 1246
defun, 1246
beforeAfter, 1272
defun, 1272
bit-to-truth, 1071
defmacro, 1071
blankList
called by filterAndFormatConstructors, 944
called by printLabelledList, 474
called by whatCommands, 942
blocks, 386
syntax, 386
Boolean
called by hashable, 1075
BooleanEquality, 1070
defun, 1070
boot help page, 487
manpage, 487
boot-equal
called by addNewInterpreterFrame, 561
called by clearCmdParts, 505
called by displaySetOptionInformation, 655
called by fetchOutput, 600
called by findFrameInRing, 559
called by getMapSig, 853
called by importFromFrame, 563
called by setHistoryCore, 585
called by undoChanges, 593
called by untraceDomainConstructor, keepTraced?, 882
called by whatConstructors, 945
called by writify, writifyInner, 607
boot-line-stack, 961
used by init-boot/spad-reader, 968
used by next-lines-clear, 972
used by next-lines-show, 971
defvar, 961
bottomUp
called by evaluateType, 916
called by interpret1, 52
called by evaluateType1, 918
bottumUp
called by evaluateType1, 918
bpiname
called by breaklet, 905
called by hashable, 1075
called by mkEvalable, 913
called by spadClosure?, 611
called by spadTrace, isTraceable, 876
called by spadTrace, 877
called by spadUntrace, 896
called by tracelet, 904
bpitrace
called by spadTrace, 877
bpiuntrace
called by spadUntrace, 896
bracketString, 1296
defun, 1296
brCon
called by kcPage, 1358
break, 906
called by letPrint2, 887
called by letPrint3, 888
called by letPrint, 885
called by set1, 809
called by setFortDir, 726
called by setFortPers, 756
called by setFortTmpDir, 724
called by setOutputCharacters, 766
called by setStreamsCalculate, 802
called by spadUntrace, 896
called by zsystemdevelopment1, 954
brightprint, 1048
called by saybrightly1, 1050
called by messageprint, 1048
defun, 1048
brightprint-0, 1048
called by saybrightly1, 1049
called by messageprint-1, 1048
defun, 1048
browse
calls AXSERV;axServer;IMV;2, 493
calls AxiomServer, 493
calls loadLib, 493
calls set, 493
browse help page, 489
manpage, 489
browseOpen, 1005
uses *allconstructors*, 1005
uses *browse-stream*, 1005
uses *browse-stream-stamp*, 1005
uses $spadroot$, 1005
defun, 1005
browseopen
called by resethashtables, 1000
called by restart0, 17
browserAutoloadOnceTrigger
called by make-databases, 1019
buildLibdb
called by make-databases, 1019
buildLibdbConEntry
called by conPageConEntry, 1344
bumpererrorcount
called by spad-syntax-error, 969
buttonNames,
defun, 1276
bvec-and, 1073
defun, 1073
bvec-concat, 1072
defun, 1072
bvec-copy, 1072
defun, 1072
bvec-elt, 1071
defmacro, 1071
bvec-equal, 1072
defun, 1072
bvec-greater, 1073
defun, 1073
bvec-make-full, 1071
defun, 1071
bvec-nand, 1074
defun, 1074
bvec-nor, 1074
defun, 1074
bvec-not, 1074
defun, 1074
bvec-or, 1073
defun, 1073
bvec-setelt, 1072
defmacro, 1072
bvec-size, 1072
defmacro, 1072
bvec-xor, 1073
defun, 1073
cacheKeyedMsg, 328
called by fetchKeyedMsg, 328
uses *msghash*, 329
catches, 328
defun, 328
throws, 329
CallerName
called by processKeyedError, 353
canCoerce
called by unifyStructVar, 429
canCoerceFrom
called by hasCateSpecial, 435
canFuncall?, 1047
defun, 1047
capitalize
called by dbConsHeading, 1395
called by kePage, 1351
called by kiPage, 1351
called by koPage, 1378
capitalize
called by conOpPage1, 1375
Catch
called by intloopSpadProcess, 63
CatchAsCan
called by intloopSpadProcess, 63
catches
cacheKeyedMsg, 328
executeQuietCommand, 45
InterpExecuteSpadSystemCommand, 30
interpretTopLevel, 51
intloop, 24
intloopSpadProcess, 63
kDomainName, 1370
kisValidType, 1373
letPrint2, 887
letPrint3, 888
monitor-file, 1169
monitor-readinterp, 1176
monitor-spadfile, 1178
npListAndRecover, 189
npParse, 141
runspad, 19
safeWritify, 606
ScanOrPairVec, 616
serverReadLine, 42
zsystemdevelopment1, 954
categoryForm?
called by evaluateType1, 917
called by loadLib, 1036
called by localnrlib, 1015
called by make-databases, 1019
categoryOpen, 1006
    uses *category-stream*, 1006
    uses *category-stream-stamp*, 1006
    uses *hasCategory-hash*, 1006
    uses $spadroot, 1006
defun, 1006
categoryopen
called by resethashtables, 1000
called by restart0, 17
categoryParts
called by reportCategory, 1347
cd help page, 495
    manpage, 495
cdancols, 1087
defmacro, 1087
cdanrows, 1086
defmacro, 1086
cdalef2, 1085
defmacro, 1085
cdelt, 1082
defmacro, 1082
cdlen, 1083
defmacro, 1083
cdsetaref2, 1086
defmacro, 1086
cdsetelt, 1083
defmacro, 1083
centerAndHighlight
called by displayExposedConstructors, 705
called by displayExposedGroups, 705
called by displayHiddenConstructors, 705
called by displayOperationsFromLisplib, 820
called by displaySetOptionInformation, 655
defun, 1292
called by displaySetVariableSettings, 657
called by filterAndFormatConstructors, 944
called by printSynonyms, 474
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
called by setExposeAddConstr, 701
called by setExposeAddGroup, 699
called by setExposeAdd, 698
called by setExposeDropConstr, 704
called by setExposeDropGroup, 702
called by setExposeDrop, 701
called by trace1, 848
called by whatCommands, 941
called by workfilesSpad2Cmd, 949
call HistListLen, 589
    called by historySpad2Cmd, 582
calls sayKeyedMsg, 589
calls spaddifference, 589
uses $HistListAct, 589
uses $HistListLen, 589
uses $HistList, 589
defun, 589
callToNamedInterpreterFrame, 560
    called by serverReadLine, 42
calls findFrameInRing, 560
calls nremove, 560
calls updateCurrentInterpreterFrame, 560
calls updateFromCurrentInterpreterFrame, 560
uses $InterpreterFrameRing, 560
defun, 560
char
called by incCommand?, 98
called by makeMsgFromLine, 371
charDigitVal, 617
called by gensymInt, 616
calls error, 617
calls spaddifference, 617
defun, 617
charPosition
called by alqlGetOrigin, 1088
called by alqlGetParams, 1089
called by conPageFastPath, 1344
called by removeUndoLines, 933
checkCondition, 1292
called by allNonNegativeInteger, 1316
childrenOf
called by kccPage, 1364
chkAllNonNegativeInteger, 1316
defun, 1316
chkDirectory, 1315
defun, 1315
chkNameList, 1313
defun, 1313
chkNonNegativeInteger, 1315
defun, 1315
chkOutputFileName, 1315
defun, 1315
chkPosInteger, 1314
defun, 1314
chkRange, 1315
defun, 1315
cleanline, 529
calledby describeSpad2Cmd, 529
defun, 529
cleanupLine, 540
defun, 540
cleanupline
calledby sayexample, 540
clear, 499
calls clearSpad2Cmd, 499
defun, 499
clear help page, 497
manpage, 497
clearAllSlams
calledby updateDatabase, 1017
clearClams
calledby clearCmdCompletely, 502
calledby setExposeAddConstr, 701
calledby setExposeAddGroup, 699
calledby setExposeDropConstr, 704
calledby setExposeDropGroup, 702
calledby updateDatabase, 1017
clearCmdAll, 503
calledby clearCmdCompletely, 502
calledby clearFrame, 929
calledby clearSpad2Cmd, 500
calls clearCmdSortedCaches, 503
calls clearMacroTable, 503
calls deleteFile, 503
calls histFileNotFoundException, 503
calls resetInCoreHist, 503
calls sayKeyedMsg, 503
calls untraceMapSubNames, 503
calls updateCurrentInterpreterFrame, 503
uses $InterpreterFrameName, 504
uses $previousBindings, 503
uses $useInternalHistoryTable, 503
uses $variableNumberAlist, 503
defun, 503
clearCmdCompletely, 502
calledby clearSpad2Cmd, 500
calls clearClams, 502
calls clearCmdAll, 502
calls clearConstructorCaches, 502
calls reclaim, 502
calls sayKeyedMsg, 502
uses $CatOfCatDatabase, 502
uses $DomOfCatDatabase, 502
uses $JoinOfCatDatabase, 502
uses $JoinOfDomDatabase, 502
uses $attributeDb, 502
uses $existingFiles, 503
uses $functionTable, 503
uses $localExposureDataDefault, 503
uses $localExposureData, 502
uses $xdatabase, 502
defun, 502
clearCmdExcept, 504
calledby clearSpad2Cmd, 500
calls clearCmdParts, 505
calls object2String, 505
calls stringPrefix?, 504
uses $clearOptions, 505
defun, 504
clearCmdParts, 505
calledby clearCmdExcept, 505
calledby clearSpad2Cmd, 500
calledby importFromFrame, 563
calls assocleft, 505
calls assoc, 505
calls boot-equal, 505
calls clearDependencies, 505
calls clearParserMacro, 505
calls deleteAssoc, 505
calls exit, 505
calls fixObjectForPrinting, 505
calls getInterpMacroNames, 505
calls getParserMacroNames, 505
calls get, 505
calls isMap, 505
calls updateFromCurrentInterpreterFrame, 562
  uses $interpreterFrameName, 562
  uses $interpreterFrameRing, 562
defun, 562
CLOSEPAREN, 107
defvar, 107
clrhash
  calledby processInteractive, 48
cmpnote, 26
defun, 26
cnstructSubst
  calledby hasSig, 423
coerceInteractive
  calledby coerceSpadArgs2E, 865
calledby coerceSpadFunValue2E, 868
calledby coerceTraceArgs2E, 864
calledby coerceTraceFunValue2E, 867
calledby interpreter2, 53
coerceOrRetract
  calledby evaluateType1, 918
coerceSpadArgs2E, 865
calledby coerceTraceFunValue2E, 867
calls coerceInteractive, 865
calls exit, 865
calls objNewWrap, 865
calls objValUnwrap, 865
calls seq, 865
calls spadsysnamep, 864
uses $OutputForm, 864
uses $mathTraceList, 864
uses $tracedMapSignatures, 864
defun, 864
coerceTraceFunValue2E, 867
calls coerceInteractive, 867
calls coerceSpadFunValue2E, 867
calls lassoc, 867
calls objNewWrap, 867
calls objValUnwrap, 867
calls spadsysnamep, 867
calls $OutputForm, 867
calls $mathTraceList, 867
calls $tracedMapSignatures, 867
defun, 867
collection, 389
syntax, 389
commandAmbiguityError, 452
calledby commandErrorIfAmbiguous, 470
calledby commandErrorMessage, 449
calledby traceOptionError, 857
calledby userLevelErrorMessage, 450
calls bright, 452
calls sayKeyedMsg, 452
calls sayMSG, 452
calls terminateSystemCommand, 452
defun, 452
commandError, 449
calledby commandErrorMessage, 449
defun, 449
commandErrorIfAmbiguous, 470
calledby commandAmbiguityError, 470
calledby commandErrorMessage, 449
calls $oldline, 470
calls line, 470
defun, 470
commandErrorMessage, 449
calledby commandError, 449
calledby optionError, 449
calls commandAmbiguityError, 449
calls sayKeyedMsg, 450
calls terminateSystemCommand, 450
uses $oldline, 450
uses line, 450
defun, 449
commandsForUserLevel, 448
called by synonymsForUserLevel, 833
called by systemCommand, 448
called by unAbbreviateKeyword, 469
called by whatCommands, 942
calls satisfiesUserLevel, 448
defun, 448
commandUserLevelError, 450
calls userLevelErrorMessage, 450
defun, 448
compareposns, 370
called by erMsgCompare, 370
called by poCharPosn, 370
called by poGlobalLinePosn, 370
defun, 370
compFailure
called by getAndEvalConstructorArgument, 958
compile help page, 513
manpage, 513
compileBoot, 907
called by breaklet, 905
called by tracelet, 904
calls /D, 1, 907
defun, 907
compiledLookup, 1076
called by compiledLookupCheck, 501
called by hashable, 1075
calls NRTevalDomain, 1076
calls isDomain, 1076
defun, 1076
compiledLookupCheck, 501
called by clearColorCachedCaches, 501
called by compiledLookup, 501
called by formatSignature, 502
called by keyedSystemError, 501
defun, 501
computeDomainVariableAlist, 1280
defun, 1280
CONCAT
called by xlskip, 87
concat, 1047
called by conOpPage1, 1375
called by copyright, 522
called by dewritify, dewritifyInner, 612
called by digits2Names, 1405
called by displayCondition, 465
called by displayModeMap, 466
called by displayMode, 466
called by displaySetOptionInformation, 655
called by displaySetVariableSettings, 657
called by displayType, 460
called by displayValue, 459
called by doSystemCommand, 446
called by evaDomain, 913
called by getTraceOption, 854
called by handleNoParseCommands, 470
called by incLude1, 80
called by inclmsgIfSyntax, 95
called by intloopReadConsole, 28
called by kDomainName, 1370
called by kedePage, 1366
called by kcnPage, 1369
called by kcuPage, 1367
called by kePage, 1351
called by koPageInputAreaUnchanged?, 1370
called by koPage, 1378
called by lffloat, 127
called by lfinteger, 134
called by mkConform, 1374
called by mkprompt, 40
called by ncloopIncFileName, 622
called by newHelpSpad2Cmd, 572
called by pcounters, 860
called by printLabelledList, 474
called by processSynonyms, 32
called by pspacers, 859
called by ptimers, 859
called by removeUndoLines, 934
called by reportOpsFromLispLib, 818
called by reportOpsFromUnitDirectly, 822
called by reportUndo, 928
called by resetCounters, 858
called by resetSpacers, 858
called by resetTimers, 858
called by scanExponent, 126
called by scanNumber, 132
called by scanS, 131
called by scanW, 128
called by setOutputAlgebra, 763
called by setOutputCharacters, 766
INDEX

called by setOutputFormula, 790

called by setOutputFortran, 770

called by setOutputHtml, 781

called by setOutputMathml, 776

called by setOutputOpenMath, 785

called by setOutputTex, 796

called by spleI1, 122

called by summary, 830

called by traceDomainConstructor, 881

called by traceReply, 899

called by undoCount, 929

called by untraceDomainConstructor, 883

called by writeInputLines, 587

calls string-concatenate, 1047

defun, 1047
condErrorMsg, 1293
defun, 1293
conOpPage, 1375
calls conOpPage1, 1375
calls dbCompositeWithMap, 1375
calls dbExtractUnderlyingDomain, 1375
calls htProperty, 1375
defun, 1375
conOpPage1, 1375
called by conOpPage, 1375
calls capitalize, 1375
calls conPageFastPath, 1375
calls concat, 1375
calls dbSourceFile, 1375
calls dbSpecialOperations, 1375
calls dbXParts, 1375
calls htInitPage, 1375
calls htpSetProperty, 1375
calls ifcar, 1375
calls ifcdr, 1375
calls isExposedConstructor, 1375
calls koPage, 1375
calls lassoc, 1375
calls mkConform, 1375
calls ncParseFromString, 1375
calls opOf, 1375
uses $Primitives, 1376
defun, 1375
conPage, 1343
called by dbSelectCon, 1394
called by dbShowCons1, 1389
calls conPageFastPath, 1343
calls downside, 1343
calls downlink, 1343
calls form2HtString, 1343
calls kPage, 1343
calls lassq, 1343
calls ySearch, 1343
uses $conArgstrings, 1343
defun, 1343
conPageChoose, 1389
calls dbShowCons1, 1389
calls getConstructorForm, 1389
defun, 1389
conPageConEntry, 1344
called by conPageFastPath, 1344
calls buildLibdbConEntry, 1344
uses $conform, 1344
uses $conname, 1344
uses $doc, 1344
uses $exposed?, 1344
uses $kind, 1344
defun, 1344
conPageFastPath, 1344
called by conOpPage1, 1375
called by conPage, 1343
calls charPosition, 1344
calls conPageConEntry, 1344
calls dbRead, 1344
calls lassq, 1344
calls length, 1344
calls stringimage, 1344
uses $lowerCaseConTb, 1344
defun, 1344
consoleinputp

called by spad-syntax-error, 969
conSpecialString?
called by kArgumentCheck, 1371
costoken, 113
called by lineoftoks, 111
called by scanToken, 114
calls ncPutQ, 113
defun, 113
constructor?
called by addTraceItem, 902
called by dbShowCons, 1388
called by evaluateType1, 917
called by evaluateType, 916
called by hasSig, 423
called by mkEvalable, 913
called by reportOpsFromLisplib, 818
called by transTraceItem, 863
called by unifyStructVar, 429
called by updateDatabase, 1017
called by writify, writifyInner, 607
constructSubst, 434
called by hasAtt, 424
called by hasCaty, 420
called by spadTrace, 877
calls internl, 434
calls stringimage, 434
defun, 434

contained
called by isPartialMode, 420
called by unifyStructVar, 429
containsVariables
called by hasCate, 433
containsVars, 430
called by hasCateSpecial, 435
called by unifyStructVar, 429
calls containsVars1, 431
calls isPatternVar, 430
defun, 430
containsVars1, 431
called by containsVars1, 431
called by containsVars, 431
calls containsVars1, 431
calls isPatternVar, 431
defun, 431

copy
called by makeInitialModemapFrame, 35
called by resetWorkspaceVariables, 654
called by undoSteps, 930
called by untrace, 862
copyright, 522
calls concat, 522
calls getenviron, 522
calls obey, 522
defun, 522
copyright help page, 517
manpage, 517
cot, 1104
defun, 1104
coth, 1106
defun, 1106
createCurrentInterpreterFrame, 557
called by updateCurrentInterpreterFrame, 559
uses $HiFiAccess, 557
uses $HistListAct, 557
uses $HistListLen, 557
uses $HistList, 557
uses $HistRecord, 557
uses $IOindex, 557
uses $InteractiveFrame, 557
uses $internalHistoryTable, 557
uses $InterpreterFrameName, 557
uses $localExposureData, 557
defun, 557

creditlist, 3
defvar, 3
credits, 525
used by credits, 525
uses credits, 525
defun, 525
credits help page, 525
manpage, 525
csc, 1105
defun, 1105
csch, 1106
defun, 1106
curinstream, 21
used by ncIntLoop, 24
defvar, 21
curoutstream, 21
used by ncIntLoop, 24
defvar, 21
current-fragment
used by iclear, 972
current-line
used by iclear, 972
current-token
used by token-stack-show, 971
currenttime
called by mkprompt, 40

daaseName, 1023
called by interpOpen, 1004
calls getEnv, 1023
INDEX

uses $spadroot, 1023
defun, 1023
dancols, 1085
defmacro, 1085
danrows, 1085
defmacro, 1085
daref2, 1084
defmacro, 1084
database, 996
defstruct, 996
dbAddChain, 1387
calledby dbAddChain, 1387
calledby dbSearchOrder, 1356
calls dbAddChainDomain, 1387
calls dbAddChain, 1387
defun, 1387
dbAddChainDomain, 1386
calledby dbAddChain, 1387
calls dbInfovec, 1386
calls dbSubConform, 1386
calls devaluate, 1386
calls kFormatSlotDomain, 1386
uses $infovec, 1386
defun, 1386
dbAddDocTable, 1383
calledby dbAddDocTable, 1382
calls getConstructorForm, 1383
calls getdatabase, 1383
calls hget, 1383
calls hput, 1383
calls opOf, 1383
calls sublislis, 1383
uses $docTable, 1383
defun, 1383
dbCompositeWithMap, 1377
calledby conOpPage, 1375
calls dbExtractUnderlyingDomain, 1377
calls htpProperty, 1377
defun, 1377
dbConsExposureMessage, 1391
calledby dbShowCons1, 1390
calls lstSay, 1391
uses $atLeastOneUnexposed, 1391
defun, 1391
dbConsHeading, 1395
calledby dbShowCons1, 1389
calls capitalize, 1395
calls form2HtString, 1395
calls htpProperty, 1395
calls length, 1395
calls member, 1395
calls nequal, 1395
calls pluralize, 1395
calls remdup, 1395
calls stringimage, 1395
uses $exposedOnlyIfTrue, 1395
defun, 1395
dbConstructorDoc, 1381
calledby dbGetDocTable, 1385
calls dbConstructorDoc, fn, 1381
uses $op, 1381
uses $sig, 1381
defun, 1381
dbConstructorDoc, fn
calledby dbConstructorDoc, 1381
dbConstructorDoc, gn, 1381
calls dbConstructorDoc, hn, 1381
uses $op, 1381
defun, 1381
dbConstructorDoc, hn, 1380
calledby dbConstructorDoc, gn, 1381
calls length, 1380
calls sublislis, 1380
uses $FormalMapVariableList, 1381
uses $args, 1381
uses $sig, 1381
defun, 1380
dbConstructorKind
calledby dbShowCons1, 1389
dbDocTable, 1382
calls dbAddDocTable, 1382
calls hget, 1382
calls make-hashtable, 1382
calls originsInOrder, 1382
uses $docTableHash, 1382
uses $docTable, 1382
defun, 1382
dbExtractUnderlyingDomain, 1377
calledby conOpPage, 1375
calledby dbCompositeWithMap, 1377
calls isValidType, 1377
calls kdr, 1377
INDEX

defun, 1377
dbGatherData
called by kePageDisplay, 1354
dbGetDocTable, 1385
calls dbConstructorDoc, 1385
calls hget, 1385
calls qcdr, 1385
calls string2Integer, 1385
uses $conform, 1385
uses $op, 1385
uses $sig, 1385
uses $which, 1385
defun, 1385
dbGetDocTable,gn, 1384
called by dbGetDocTable, 1385
calls dbGetDocTable,hn, 1384
calls lastatom, 1384
uses $conform, 1384
defun, 1384
dbGetDocTable,hn, 1384
called by dbGetDocTable,gn, 1384
calls kdr, 1384
calls qcar, 1384
calls qcdr, 1384
calls sublislis, 1384
uses $FormalMapVariableList, 1384
uses $conform, 1384
uses $sig, 1384
uses $which, 1384
defun, 1384
dbGetInputString
called by dbShowCons, 1387
called by konPageFilterByName, 1380
dbInfovec
called by dbAddChainDomain, 1386
called by dbSearchOrder, 1356
dbMkEvalable, 1372
called by kDomainName, 1370
calls getdatabase, 1372
calls mkEvalable, 1372
defun, 1372
dbMkForm, 1406
defun, 1406
dbName
called by dbShowConstructorLines, 1397
dbNonEmptyPattern, 1325
defun, 1325
dbPart
called by alqlGetKindString, 1089
called by alqlGetOrigin, 1088
called by alqlGetParams, 1089
dbpHasDefaultCategory?
called by kcPage, 1358
dbPresentCons
called by dbShowCons1, 1390
dbRead
called by conPageFastPath, 1344
dbSayItems
called by dbShowConditions, 1395
dbSearchOrder, 1356
called by ksPage, 1355
calls dbAddChain, 1356
calls dbInfovec, 1356
calls dbSubConform, 1356
calls devaluate, 1356
calls getdatabase, 1356
calls kFormatSlotDomain, 1356
calls kTestPred, 1356
calls opOf, 1356
calls simpCatPredicate, 1356
calls sublislis, 1356
uses $domain, 1356
uses $infovec, 1356
uses $predvec, 1356
defun, 1356
dbSelectCon, 1394
called by conPage, 1394
calls htpProperty, 1394
calls opOf, 1394
defun, 1394
dbShowConditions, 1394
called by dbShowCons1, 1390
calls bcConPredTable, 1390
calls dbSayItems, 1395
calls htSayHrule, 1395
calls htSaySaturn, 1395
calls htpProperty, 1395
calls length, 1395
calls opOf, 1394
calls pluralize, 1395
calls splitConTable, 1394
calls stringimage, 1395
defun, 1394
dbShowCons, 1387
calledby dbShowConsKindsFilter, 1392
calledby dbShowCons, 1388
calledby kArgPage, 1346
calledby kcaPage1, 1363
calledby kcePage, 1364
calledby kcdePage, 1366
calledby kcnPage, 1369
calledby kcpPage, 1361
calledby kcuPage, 1367
calledby ksPage, 1355
calls bcErrorPage, 1388
calls constructor?, 1388
calls dbGetInputString, 1387
calls dbShowCons1, 1388
calls dbShowCons, 1388
calls downcase, 1388
calls emptySearchPage, 1388
calls htCopyProplist, 1388
calls htInitPageNoScroll, 1388
calls htShowPageNoScroll, 1390
calls htpProperty, 1389
calls htpSetProperty, 1389
calls isExposedConstructor, 1389
calls listSort, 1389
calls opOf, 1389
calls qlesseqp, 1389
calls remdup, 1389
calls union, 1389
uses $conformsAreDomains, 1390
uses $exposedOnlyIfTrue, 1390
defun, 1389
dbShowConsDoc, 1392
calledby dbShowCons1, 1389
calls dbShowConsDoc1, 1392
calls getConstructorDocumentation, 1393
calls getConstructorSignature, 1393
calls getdatabase, 1393
calls getl, 1393
calls htSaySaturn, 1393
calls htpProperty, 1393
calls isExposedConstructor, 1393
calls member, 1393
calls sublisFormal, 1393
uses $Primitives, 1393
uses $TriangleVariableList, 1393
defun, 1393
dbShowConsKinds
called by dbShowCons1, 1390
calls dbShowConsKindsFilter, 1391
calls dbShowCons, 1392
calls htpProperty, 1392
calls htpSetProperty, 1391
defun, 1391
dbShowConstructorLines, 1397
calls dbName, 1397
calls dbShowCons1, 1397
calls function, 1397
calls getConstructorForm, 1397
calls glesseqp, 1397
calls intern, 1397
calls listSort, 1397
defun, 1397
dbShowOp1
called by dbSpecialOperations, 1399
calls form2HtString, 1399
calls getConstructorForm, 1399
calls getl, 1399
calls htInitPage, 1399
calls htShowPage, 1399
calls kePageDisplay, 1399
defun, 1399
dbSpecialOperations, 1398
called by conOpPage1, 1375
calls dbShowOp1, 1399
calls dbSpecialExpandIfNecessary, 1398
calls form2HtString, 1398
calls getConstructorForm, 1398
calls getl, 1398
calls htInitPage, 1398
calls htpSetProperty, 1398
defun, 1398
dbSplitLibdb
called by make-databases, 1019
dbSubConform, 1386
called by dbAddChainDomain, 1386
called by dbSearchOrder, 1356
called by dbSubConform, 1387
calls dbSubConform, 1387
calls position, 1386
calls $conformsAreDomains, 1387
defun, 1386
dbXParts
called by conOpPage1, 1375
dc1
called by reportOpsFromLisplib, 818
debugmode
used by spad-syntax-error, 969
decideHowMuch, 359
called by getPosStL, 356
calls poFileName, 359
calls poLinePosn, 359
calls poNopos?, 359
calls poPosImmediate?, 359
defun, 359
defaultTargetFE, 437
called by defaultTargetFE, 437
called by hasCateSpecialNew, 436
calls defaultTargetFE, 437
calls icEar, 437
calls isEqualOrSubDomain, 437
calls typeIsASmallInteger, 437
local ref $FunctionalExpression, 437
local ref $Integer, 437
local ref $RationalNumber, 438
local ref $Symbol, 438
defun, 437
defostream, 982
calledby reportOpsFromLisplib1, 817
calledby reportOpsFromUnitDirectly1, 825
calledby sayMSG2File, 331
calledby setOutputAlgebra, 763
calledby setOutputFormula, 790
calledby setOutputFortran, 770
calledby setOutputHtml, 781
calledby setOutputMathml, 776
calledby setOutputOpenMath, 785
calledby setOutputTex, 796
calledby writeInputLines, 587
calledby zsystemdevelopment1, 954
defun, 982
defmacro
ancolsU16, 1060
ancolsU32, 1061
ancolsU8, 1058
anrowsU16, 1059
anrowsU32, 1061
anrowsU8, 1058
aref2U16, 1059
aref2U32, 1060
aref2U8, 1057
assq, 1050
bit-to-truth, 1071
bvec-celt, 1071
bvec-setelt, 1072
bvec-size, 1072
cdancols, 1087
cdanrows, 1086
cdaref2, 1085
cdelt, 1082
cdlen, 1083
cdsetaref2, 1086
cdsetelt, 1083
dancols, 1085
danrows, 1085
daref2, 1084
delt, 1082
DFAcos, 1099
DFAcosh, 1102
DFAdd, 1095
DFAsin, 1099
DFAsinh, 1101
DFAtan, 1100
DFAtan2, 1100
DFAtanh, 1102
DFCos, 1099
DFCosh, 1101
DFDivide, 1096
DFEq., 1096
DFExp, 1098
DFExpt, 1098
DFIntegerDivide, 1097
DFIntegerExpt, 1098
DFIntegerMultiply, 1095
DFLessThan, 1094
DFLog, 1097
DFLogE, 1097
DFMax, 1096
DFMin, 1096
DFMinus, 1094
DFMultiply, 1095
DFSin, 1098
DFSinh, 1100
DFSqrt, 1097
DSum, 1095
DFTan, 1099
DFTanb, 1101
DFUnaryMinus, 1094
DFZerop, 1094
dlen, 1081
dsetaref2, 1084
dsetelt, 1082
e1tU16, 1056
e1tU32, 1057
e1tU8, 1055
funfind, 874
hget, 1044
idChar?, 128
identp, 1046
make-cdouble-matrix, 1085
make-cdouble-vector, 1082
make-double-matrix, 1084
make-double-matrix1, 1084
make-double-vector, 1081
make-double-vector1, 1081
makeMatrix1U16, 1060
makeMatrix1U32, 1062
makeMatrix1U8, 1059
makeMatrixU16, 1060
makeMatrixU32, 1061
makeMatrixU8, 1058
qsabsval, 1069
qsadd1, 1068
qsdifference, 1067
qsDot26432, 1063
qsDot2Mod6432, 1063
qslessp, 1068
qsmmax, 1070
qsmmin, 1070
qsmminus, 1068
qsMod6432, 1062
qsMul6432, 1063
qsMulAdd6432, 1062
qsMulAddMod6432, 1063
qsMulMod32, 1062
qsoddp, 1069
qsplus, 1069
qsplus1, 1068
qstimes, 1069
qszerop, 1070
qvlenU16, 1055
qvlenU32, 1056
qvlenU8, 1054
Rest, 75
setAref2U16, 1059
setAref2U32, 1061
setAref2U8, 1058
seteltU16, 1056
seteltU32, 1057
seteltU8, 1055
spadConstant, 1155
startId?, 1044
trapNumericErrors, 1080
truth-to-bit, 1071
while, 1039
whileWithResult, 1040
defstruct
database, 996
libstream, 1164

monitor-data, 1164
defun
/read, 644
/tracereply, 894
?t, 902
abbQuery, 536
abbreviations, 483
abbreviationsSpad2Cmd, 483
acot, 1104
acoth, 1107
acsc, 1105
acsch, 1106
addBinding, 963
addBindingInteractive, 967
addInputLibrary, 668
addNewInterpreterFrame, 561
addoperations, 1007
addTraceItem, 902
algCoerceInteractive, 1066
allConstructors, 1032
allOperations, 1033
alqlGetKindString, 1089
alqlGetOrigin, 1088
alqlGetParams, 1089
alreadyOpened?, 363
apropos, 945
asec, 1105
asech, 1107
assertCond, 94
augmentHasArgs, 1365
augmentTraceNames, 872
axiomVersion, 476
basicLookup, 1076
basicLookupCheckDefaults, 1078
basicStringize, 1261
bcComplexLimit, 1223
bcComplexLimitGen, 1225
bcCreateVariableString, 1233
bcDefiniteIntegrate, 1190
bcDefiniteIntegrateGen, 1192
bcDifferentiate, 1195
bcDifferentiateGen, 1196
bcDraw, 1197
bcDraw2Dfun, 1198
bcDraw2DfunGen, 1199
bcDraw2Dpar, 1200
INDEX

bcDraw2DparGen, 1201  
bcDraw2DSolve, 1202  
bcDraw2DSolveGen, 1203  
bcDraw3Dfun, 1204  
bcDraw3DfunGen, 1205  
bcDraw3Dpar, 1206  
bcDraw3Dpar1, 1208  
bcDraw3Dpar1Gen, 1209  
bcDraw3DparGen, 1207  
bcDrawIt, 1247  
bcDrawIt2, 1189  
bcError, 1247  
bcFindString, 1244  
bcFinish, 1243  
bcGen, 1244, 1245  
bcGenEquations, 1242  
bcGenExplicitMatrix, 1187  
bcHt, 1260  
bchtMakeButton, 1287  
bcIndefiniteIntegrate, 1189  
bcIndefiniteIntegrateGen, 1190  
bcInputEquations, 1231  
bcInputEquationsEnd, 1235  
bcInputExplicitMatrix, 1185  
bcInputMatrixByFormula, 1183  
bcInputMatrixByFormulaGen, 1185  
bcInputSolveInfo, 1230  
bcIssueHt, 1261  
bcLaurentSeries, 1215  
bcLaurentSeriesGen, 1219  
bcLimit, 1220  
bcLinearExtractMatrix, 1238  
bcLinearMatrixGen, 1240  
bcLinearSolve, 1227  
bcLinearSolveEqns, 1228  
bcLinearSolveEqns1, 1230  
bcLinearSolveEqnsGen, 1242  
bcLinearSolveMatrix, 1237  
bcLinearSolveMatrix1, 1237  
bcLinearSolveMatrixHomo, 1240  
bcLinearSolveMatrixInhomo, 1238  
bcLinearSolveMatrixInhomoGen, 1239  
bcMakeEquations, 1234  
bcMakeLinearEquations, 1234  
bcMakeUnknowns, 1233  
bcMatrix, 1182  
bcMatrixGen, 1188  
bcMkFunction, 1243  
bcNotReady, 1247  
bcOptional, 1245  
bcProduct, 1194  
bcProductGen, 1195  
bcPuiseuxSeries, 1217  
bcPuiseuxSeriesGen, 1219  
bcReadMatrix, 1182  
bcRealLimit, 1221  
bcRealLimitGen, 1222  
bcRealLimitGen1, 1223  
bcSadFaces, 1269  
bcSeries, 1210  
bcSeriesByFormula, 1212  
bcSeriesByFormulaGen, 1215  
bcSeriesExpansion, 1211  
bcSeriesExpansionGen, 1212  
bcSeriesGen, 1219  
bcSolve, 1226  
bcSolveEquations, 1236  
bcSolveEquationsNumerically, 1235  
bcSolveNumerically1, 1236  
bcSolveSingle, 1229  
bcString2HyString, 1244  
bcString2HyString2, 1244  
bcString2WordList, 1246  
bcSum, 1192  
bcSumGen, 1194  
bcSystemSolve, 1228  
bcSystemSolveEqns1, 1229  
bcTaylorSeries, 1213  
bcTaylorSeriesGen, 1219  
bcUnixTable, 1397  
bcVectorGen, 1246  
bcvspace, 1246  
bcwords2liststring, 1246  
beforeAfter, 1272  
BooleanEquality, 1070  
bracketString, 1296  
break, 906  
breaklet, 905  
brightprint, 1048  
brightprint-0, 1048  
browseOpen, 1005  
buttonNames, 1276
bvec-and, 1073
bvec-concat, 1072
bvec-copy, 1072
bvec-equal, 1072
bvec-greater, 1073
bvec-make-full, 1071
bvec-nand, 1074
bvec-nor, 1074
bvec-not, 1074
bvec-or, 1073
bvec-xor, 1073
cacheKeyedMsg, 328
canFuncall?, 1047
categoryOpen, 1006
changeHistListLen, 589
changeToNamedInterpreterFrame, 560
ccheckCondition, 1292
cchkAllNonNegativeInteger, 1316
cchkDirectory, 1315
cchkNameList, 1313
cchkNonNegativeInteger, 1315
cchkOutputFileName, 1315
cchkPosInteger, 1314
cchkRange, 1315
cleanline, 529
cleanupLine, 540
clear, 499
clearCmdAll, 503
clearCmdCompletely, 502
clearCmdExcept, 504
clearCmdParts, 505
clearCmdSortedCaches, 501
clearFrame, 929
clearMacroTable, 504
clearParserMacro, 453
clearSpad2Cmd, 500
close, 510
closeInterpreterFrame, 562
cmpnote, 26
coerceSpadArgs2E, 865
coerceSpadFunValue2E, 868
coerceTraceArgs2E, 864
coerceTraceFunValue2E, 867
commandAmbiguityError, 452
commandError, 449
commandErrorIfAmbiguous, 470
commandErrorMessage, 449
commandsForUserLevel, 448
commandUserLevelError, 450
compareposns, 370
compileBoot, 907
compiledLookup, 1076
compiledLookupCheck, 501
computeDomainVariableAlist, 1280
concat, 1047
condErrorMsg, 1293
conOpPage, 1375
conOpPage1, 1375
conPage, 1343
conPageChoose, 1389
conPageConEntry, 1344
conPageFastPath, 1344
costoken, 113
constructSubst, 434
containsVars, 430
containsVars1, 431
copyright, 522
cot, 1104
coth, 1106
createCurrentInterpreterFrame, 557
credits, 525
csc, 1105
csch, 1106
DaaseName, 1023
dbAddChain, 1387
dbAddChainDomain, 1386
dbAddDocTable, 1383
dbCompositeWithMap, 1377
dbConsExposureMessage, 1391
dbConsHeading, 1395
dbConstructorDoc, 1381
dbConstructorDoc,gn, 1381
dbConstructorDoc,hn, 1380
dbDocTable, 1382
dbExtractUnderlyingDomain, 1377
dbGetDocTable, 1385
dbGetDocTable,gn, 1384
dbGetDocTable,hn, 1384
dbMkEvalable, 1372
dbMkForm, 1406
dbNonEmptyPattern, 1325
INDEX

dbSearchOrder, 1356
dbSelectCon, 1394
dbShowConditions, 1394
dbShowCons, 1387
dbShowCons1, 1389
dbShowConsDoc, 1392
dbShowConsDoc1, 1393
dbShowConsKindsFilter, 1391
dbShowConstructorLines, 1397
dbSpecialDescription, 1398
dbSpecialExpandIfNecessary, 1400
dbSpecialExports, 1399
dbSpecialOperations, 1398
dbSubConform, 1386
decideHowMuch, 359
defaultTargetFE, 437
defiostream, 982
Delay, 102
deldatabase, 1009
deleteFile, 1042
describe, 528
describeFortPersistence, 757
describeInputLibraryArgs, 668
describeOutputLibraryArgs, 665
describeSetFortDir, 727
describeSetFortTmpDir, 725
describeSetLinkerArgs, 729
describeSetNagHost, 755
describeSetOutputAlgebra, 765
describeSetOutputFormula, 792
describeSetOutputFortran, 772
describeSetOutputHtml, 783
describeSetOutputMathml, 778
describeSetOutputOpenMath, 788
describeSetOutputTex, 798
describeSetStreamsCalculate, 802
describeSpad2Cmd, 528
desiredMsg, 352
dewritify, 615
dewritify,dewritifyInner, 612
diffAlist, 924
digit?, 122
digitp, 1045
digits2Names, 1405
DirToString, 1091
disableHist, 603
display, 535
displayCondition, 465
displayExposedConstructors, 705
displayExposedGroups, 705
displayFrameNames, 563
displayHiddenConstructors, 705
displayMacro, 453
displayMacros, 538
displayMode, 466
displayModemap, 466
displayOperations, 537
displayOperationsFromLisplib, 820
displayParserMacro, 464
displayProperties, 461
displayProperties,sayFunctionDeps, 456
displaySetOptionInformation, 655
displaySetVariableSettings, 657
displayType, 460
displayValue, 459
displayWorkspaceNames, 454
divide2, 1087
doDoitButton, 1289
domainDescendantsOf, 1349
domainToGenvar, 861
domArg, 422
domArg2, 422
doSystemCommand, 446
downlink, 1324
downlinkSaturn, 1324
dqAppend, 344
dqConcat, 343
dqToList, 344
dqUnit, 343
dropInputLibrary, 669
dumbTokenize, 467
edit, 544
editFile, 545
editSpad2Cmd, 544
eemptyInterpreterFrame, 556
enPile, 340
eofp, 983
eqpileTree, 339
erMsgCompare, 370
erMsgSep, 370
erMsgSort, 369
evalCategory, 959
evalDomain, 913
evaluateSignature, 920
evaluateType, 916
evaluateType1, 917
executeInterpreterCommand, 1289
ExecuteInterpSystemCommand, 31
executeQuietCommand, 45
explainLinear, 1241
fetchKeyedMsg, 328
fetchOutput, 600
fillerSpaces, 18
filterAndFormatConstructors, 944
filterListOfStrings, 942
filterListOfStringsWithFn, 943
fin, 548
finalExactRequest, 1241
findFirstWord, 469
flushObjectForPrinting, 456
flatten, 531
flattenOperationAlist, 883
float2Sex, 305
fnameDirectory, 1091
fnameExists?, 1092
fnameMake, 1090
fnameName, 1092
fnameNew, 1093
fnameReadable?, 1092
fnameType, 1092
fnameWritable?, 1093
frame, 565
frameEnvironment, 556
frameName, 552
frameNames, 556
frameSpad2Cmd, 566
From, 380
FromTo, 380
funfindLAM, 874
gatherGlossLines, 1334
genDomainTraceName, 862
gensymInt, 616
get-current-directory, 34
getAliasIfTracedMapParameter, 889
getAndEvalConstructorArgument, 958
getAndSay, 461
getBpiNameIfTracedMap, 890
getAddress, 1089
getConstructorDocumentation, 1394
database, 1010
dependentsOfConstructor, 1367
directoryList, 984
directory, 29
depthWord, 469
defKeyedMsg, 329
defl, 1050
deflinePos, 372
deflineText, 372
defMapSig, 853
defMapSubNames, 869
defMsgArgL, 349
defMsgCatAttr, 358
defMsgFTTag?, 359
defMsgInfoFromKey, 366
defMsgKey, 348
defMsgKey?, 362
defMsgLitSym, 362
defMsgPos, 359
defMsgPos2, 378
defMsgPosTagOb, 348
defMsgPrefix, 349
defMsgPrefix?, 350
defMsgTag, 350
defMsgTag?, 350
defMsgText, 349
defMsgToWhere, 363
defOplistForConstructorForm, 824
defOplistWithUniqueSignatures, 825
defOption, 892
defParserMacroNames, 453
defPosStL, 356
defPreStL, 355
defPreviousMapSubNames, 870
defPropList, 964
defRefvU16, 1056
defRefvU32, 1057
defRefvU8, 1055
defStFromMsg, 354
defSystemCommandLine, 833
defTraceOption, 854
defTraceOption, 853
defTraceOptions, 852
defUsersOfConstructor, 1368
getWorkspaceNames, 455  
handleNoParseCommands, 445  
handleParsedSystemCommands, 468  
handleTokensizeSystemCommands, 447  
hasAtt, 424  
hasAttSig, 426  
hasCate, 433  
hasCate1, 427  
hasCateSpecial, 434  
hasCateSpecialNew, 436  
hasCatExpression, 427  
hasCaty, 420  
hasCaty1, 432  
hashable, 1075  
hasOptArgs?, 309  
hasOption, 451  
hasPair, 891  
hasSig, 423  
hasSigAnd, 425  
hasSigOr, 426  
help, 572  
helpSpad2Cmd, 572  
histFileErase, 617  
histFileName, 580  
histInputFileName, 580  
history, 582  
historySpad2Cmd, 582  
hkeys, 1044  
hput, 1044  
htAddHeading, 1262  
htAllOrNum, 1323  
htBcLinks, 1270  
htBcLispLinks, 1271  
htBcRadioButtons, 1274  
htCacheAddChoice, 1320  
htCacheOne, 1323  
htCacheSet, 1321  
htCheck, 1311  
htCheckList, 1312  
htDoneButton, 1289  
htDoNothing, 1311  
htEscapeString, 1297  
htFunctionSetLiteral, 1308  
htGloss, 1332  
htGlossPage, 1332  
htGlossSearch, 1336  
htGreekSearch, 1336  
htInitPage, 1262  
htInputStrings, 1276  
htKill, 1310  
htLispLinks, 1269  
htLispMemoLinks, 1270  
htMakeButton, 1285  
htMakeDoitButton, 1288  
htMakeDoneButton, 1284  
htMakeErrorPage, 1265  
htMakeInputList, 1296  
htMakeLabel, 1321  
htMakePage, 1263  
htMakePage1, 1264  
htMakePathKey, 1317  
htMakePathKey.fn, 1316  
htMakeTemplates, 1283  
htMakeTemplates, substLabel, 1282  
htMarkTree, 1317  
htMkName, 1248  
htpAddInputAreaProp, 1253  
htpAddToPageDescription, 1260  
htpButtonValue, 1252  
htpDestroyPage, 1250  
htpDomainConditions, 1250  
htpDomainPvarSubstList, 1251  
htpDomainVariableAlist, 1251  
htpInputAreaAlist, 1253  
htpLabelDefault, 1258  
htpLabelErrorMsg, 1257  
htpLabelFilter, 1259  
htpLabelFilteredInputString, 1255  
htpLabelInputString, 1254  
htpLabelSpadType, 1258  
htpLabelSpadValue, 1256  
htpLabelType, 1258  
htpName, 1250  
htpPageDescription, 1259  
htpProperty, 1254  
htpPropertyList, 1253  
htpRadioButtonAlist, 1252  
htProcessBcButtons, 1266  
htProcessBcStrings, 1268  
htProcessDoitButton, 1288  
htProcessDomainConditions, 1277  
htProcessDoneButton, 1285
htProcessToggleButtons, 1265
htSetDomainConditions, 1251
htSetDomainPvarSubstList, 1251
htSetDomainVariableAlist, 1251
htSetInputAreaAlist, 1253
htSetLabelErrorMsg, 1257
htSetLabelInputString, 1257
htSetName, 1250
htSetPageDescription, 1259
htSetProperty, 1254
htSetRadioButtonAlist, 1252
htQuote, 1265
htRadioButtons, 1273
htSetCache, 1319
htSetExpose, 1318
htSetFunCommand, 1309
htSetFunCommandContinue, 1309
htSetHistory, 1317
htSetInputLibrary, 1318
htSetInteger, 1306
htSetLinkerArgs, 1319
htSetLiteral, 1304
htSetLiterals, 1303
htSetNotAvailable, 1310
htSetOutputCharacters, 1319
htSetOutputLibrary, 1318
htSetSystemVariable, 1331
htSetSystemVariableKind, 1331
htSetvarDoneButton, 1308
htSetVars, 1298
htShowCount, 1300
htShowFunctionPage, 1306
htShowFunctionPageContinued, 1307
htShowIntegerPage, 1305
htShowLiteralsPage, 1303
htShowPage, 1263
htShowPageNoScroll, 1263
htShowSetPage, 1302
htShowSetTree, 1298
htShowSetTreeValue, 1301
htStringPad, 1248
htsv, 1298
htSystemVariables, 1329
htSystemVariables,displayOptions, 1326
htSystemVariables,fn, 1326
htSystemVariables, functionTail, 1328
htSystemVariables, gn, 1325
htTextSearch, 1338
htTutorialSearch, 1340
ifCond, 87
iht, 1260
importFromFrame, 563
incActive?, 101
incAppend, 85
incAppend1, 85
incBiteOff, 623
incClassify, 97
incCommand?, 98
incConsoleInput, 100
incDrop, 100
incFileInput, 100
incFileName, 622
incHandleMessage, 74
incIgen, 73
incIgen1, 73
inclFname, 100
inclLine, 86
inclLine1, 86
inclmsgCannotRead, 90
inclmsgCmdBug, 96
inclmsgConActive, 92
inclmsgConsole, 93
inclmsgConStill, 92
inclmsgFileCycle, 90
inclmsgFinSkipped, 93
inclmsgIFBug, 96
inclmsgIFSyntax, 95
inclmsgNoSuchFile, 89
inclmsgPrematureEOF, 86
inclmsgPrematureFin, 94
inclmsgSay, 88
include, 75
include1, 79
incNConsoles, 101
incPrefix?, 99
incRemnumber, 72
incRemnumberItem, 74
incRemnumberLine, 73
incRgen, 101
incRgen1, 102
incStream, 71
incString, 37
incZip, 72
incZip1, 72
init-boot/spad-reader, 968
init-memory-config, 32
initHist, 581
initHistList, 581
initializeInterpreterFrameRing, 555
initializeSetVariables, 653
initImPr, 367
initroot, 33
initToWhere, 368
insertpile, 335
insertPos, 379
integer-decode-float-denominator, 1103
integer-decode-float-exponent, 1103
integer-decode-float-numerator, 1102
integer-decode-float-sign, 1103
InterpExecuteSpadSystemCommand, 30
interFunctionDepAlists, 465
interpOpen, 1003
interpret, 52
interpret1, 52
interpret2, 53
interpretTopLevel, 51
intInterpretPform, 65
intloop, 24
intloopEchoParse, 67
intloopInclude, 61
intloopInclude0, 61
intloopPrefix?, 34
intloopProcess, 62
intloopProcessString, 36
intloopReadConsole, 28
intloopSpadProcess, 63
intloopSpadProcess.interp, 64
intnplisp, 34
intProcessSynonyms, 31
ioclear, 972
iostat, 970
isDomainOrPackage, 875
isDomainValuedVariable, 959
isEqualOrSubDomain, 438
isExposedConstructor, 820
isgenvar, 886
isIntegerString, 468
isInterpOnlyMap, 872
isListOfIdentifiers, 868
isListOfIdentifiersOrStrings, 869
isPartialMode, 420
isPatternVar, 431
isSharpVar, 886
isSharpVarWithNum, 886
isSubForRedundantMapName, 873
isSystemDirectory, 1037
isTraceGensym, 875
isUncompiledMap, 871
justifyMyType, 60
kArgPage, 1346
kArgumentCheck, 1371
kcaPage, 1362
kcaPage1, 1363
kccPage, 1364
kcdePage, 1366
kedoPage, 1363
kedPage, 1362
kCheckArgumentNumbers, 1373
kcnPage, 1368
kcPage, 1357
kcpPage, 1361
kcuPage, 1367
kDomainName, 1370
kdPageInfo, 1345
kePage, 1351
kePageDisplay, 1354
kePageOpAlist, 1353
keyword, 121
keyword?, 121
kiPage, 1350
kisValidType, 1372
koaPageFilterByName, 1380
koPage, 1377
koPageAux, 1379
koPageAux1, 1379
koPageFromKKPage, 1378
koPageInputAreaUnchanged?, 1370
ksPage, 1355
kTestPred, 1386
lassocSub, 871
lastTokPosn, 341
leader?, 351
leaveScratchpad, 639
lefts, 1406
letPrint, 885
letPrint2, 887
letPrint3, 888
lfcomment, 116
lferror, 135
lffloat, 127
lfid, 115
lfinteger, 133
lfkey, 122
lfnegcomment, 118
lfprint, 134
lfspaces, 130
lfstring, 130
libConstructorSig, 1406
library, 1013
line?, 351
linearFinalRequest, 1241
lineoftoks, 111
listConstructorAbbreviations, 484
listDecideHowMuch, 361
listOfStrings2String, 1302
listOutputter, 354
lnCreate, 345
lnExtraBlanks, 345
lnFileName, 347
lnFileName?, 347
lnGlobalNum, 346
lnImmediate?, 347
lnLocalNum, 346
lnPlaceOfOrigin, 346
lnSetGlobalNum, 346
lnString, 345
load, 629
loadFunctor, 1038
loadLib, 1035
loadLibNoUpdate, 1037
localdatabase, 1014
localnrlib, 1015
lookupInDomainVector, 1078
loopIters2Sex, 311
lout, 1043
ltrace, 632
mac0Define, 230
mac0ExpandBody, 224
mac0Get, 228
mac0GetName, 226
mac0InfiniteExpansion, 225
mac0InfiniteExpansion,name, 226
mac0MLambdaApply, 223
mac0SubstituteOuter, 231
macApplication, 223
macExpand, 222
macId, 227
macLambda, 228
macLambda,mac, 229
macLambdaParameterHandling, 231
macMacro, 229
macroExpanded, 222
macSubstituteId, 232
macSubstituteOuter, 230
macWhere, 228
macWhere,mac, 228
make-absolute-filename, 35
make-appendstream, 982
make-databases, 1018
make-instream, 981
make-outstream, 981
makeByteWordVec2, 1155
makeColl, 35
makeColl,mac, 35
makeColl,mac,mac, 35
makeColl,mac,mac,mac, 35
makeHistFileName, 579
makeInputFilename, 983
makeLeaderMsg, 377
makeMsgFromLine, 371
makeOrdinary, 920
makePathname, 1042
makeSpadCommand, 1295
makeStream, 983
manexp, 1103
mapLetPrint, 884
mapStringize, 1261
member, 1048
mergePathnames, 1041
messageprint, 1048
messageprint-1, 1049
messageprint-2, 1049
mkConArgSublis, 1405
mkConform, 1374
mkCurryFun, 1272
INDEX

mkDomPvar, 433
mkDomTypeForm, 1348
mkEvalable, 913
mkEvalableMapping, 915
mkEvalableRecord, 915
mkEvalableUnion, 915
mkLineList, 68
mkprompt, 40
mkSetTitle, 1302
mkUnixPattern, 1341
monitor-add, 1166
monitor-apropos, 1179
monitor-autoload, 1174
monitor-checkpoint, 1171
monitor-decr, 1168
monitor-delete, 1166
monitor-dirname, 1174
monitor-disable, 1167
monitor-enable, 1166
monitor-end, 1165
monitor-exposedp, 1176
monitor-file, 1169
monitor-help, 1172
monitor-incr, 1168
monitor-info, 1169
monitor-inititable, 1164
monitor-libname, 1175
monitor-lrilib, 1175
monitor-parse, 1178
monitor-percent, 1179
monitor-readinterp, 1176
monitor-report, 1177
monitor-reset, 1167
monitor-restore, 1171
monitor-results, 1165
monitor-spadfile, 1178
monitor-tested, 1170
monitor-untested, 1169
monitor-write, 1170
msgCreate, 347
msgImPr?, 358
msgNoRep?, 375
msgOutputter, 353
msgText, 60
myWritable?, 1093
namestring, 1040
ncAlist, 415
ncBug, 368
ncConversationPhase, 66
ncConversationPhase, wrapup, 66
ncElitQ, 416
ncError, 67
ncHardError, 352
ncIntLoop, 23
ncloopCommand, 478
ncloopDQlines, 70
ncloopEscaped, 35
ncloopIncFileName, 622
ncloopInclude, 622
ncloopInclude0, 71
ncloopInclude1, 621
ncloopParse, 36
ncloopPrefix?, 479
ncloopPrintLines, 68
ncParseFromString, 1067
ncPutQ, 416
ncSoftError, 351
ncTag, 415
ncTopLevel, 23
newHelpSpad2Cmd, 572
next, 36
next-lines-clear, 972
next-lines-show, 971
next1, 37
nextInterpreterFrame, 560
nextline, 112
nonBlank, 69
npADD, 158
npAdd, 159
npAmpersand, 204
npAmpersandFrom, 202
npAndOr, 181
npAngleBared, 186
npAnyNo, 162
npApplication, 162
npApplication2, 163
npArith, 200
npAssign, 216
npAssignment, 217
npAssignVariable, 217
npAtom1, 183
npAtom2, 159
npBacksetElse, 197
npBackTrack, 148
npBDefinition, 185
npBoot, 472
npBProfileDefinition, 188
npBraced, 186
npBracked, 186
npBracketed, 185
npBreak, 174
npBy, 199
npCategory, 153
npCategoryL, 152
npCoerceTo, 220
npColon, 217
npColonQuery, 219
npComma, 146
npCommaBackSet, 146
npCompMissing, 151
npConditional, 195
npConditionalStatement, 180
npConstTok, 184
npDDInfKey, 208
npDecl, 214
npDef, 187
npDefaultDecl, 170
npDefaultItem, 169
npDefaultItemlist, 168
npDefaultValue, 194
npDefinition, 167
npDefinitionItem, 167
npDefinitionItemlist, 193
npDefinitionOrStatement, 147
npDefn, 187
npDefTail, 194
npDiscrim, 197
npDisjand, 197
npDollar, 183
npDotted, 162
npElse, 196
npEncAp, 182
npEncL, 182
npEnclosed, 211
npEqKey, 145
npEqPeek, 152
npExit, 215
npExport, 171
npExpress, 179
npExpress1, 179
npFirstTok, 143
npFix, 166
npForIn, 177
npFree, 173
npFromdom, 202
npFromdom1, 203
npGives, 148
npId, 204
npImport, 180
npInfGeneric, 207
npInfOp, 161
npInfOperator, 160
npInfKey, 208
npInline, 174
npInterval, 199
npItem, 142
npItem1, 142
npIterate, 174
npIterator, 176
npIterators, 175
npLambda, 148
npLeftAssoc, 206
npLet, 166
npLetQualified, 166
npLisp, 472
npList, 155
npListAndRecover, 189
npListing, 155
npListoffun, 221
npLocal, 173
npLocalDecl, 172
npLocalItem, 172
npLocalItemlist, 171
npLogical, 197
npLoop, 175
npMacro, 164
npMatch, 150
npMDEF, 165
npMdef, 164
npMDEFinition, 165
npMissing, 151
npMissingMate, 215
npMoveTo, 191
npName, 204
INDEX

npNext, 145
npNull, 333
npParened, 185
npParenthesize, 215
npParenthesized, 214
npParse, 141
npPDefinition, 183
npPileBracketed, 188
npPileDefinitionlist, 189
npPileExit, 216
npPop1, 144
npPop2, 144
npPop3, 144
npPower, 202
npPP, 209
npPPf, 211
npPPff, 210
npPPg, 210
npPrefixColon, 161
npPretend, 219
npPrimary, 157
npPrimary1, 164
npPrimary2, 158
npProcessSynonym, 473
npProduct, 202
npPush, 143
npPushId, 209
npQualDef, 145
npQualified, 147
npQualifiedDefinition, 147
npQualType, 181
npQualTypelist, 180
npQuiver, 198
npRecoverTrap, 190
npRelation, 198
npRemainder, 201
npRestore, 152
npRestrict, 220
npReturn, 178
npRightAssoc, 206
npRule, 193
npSCategory, 153
npSDefaultItem, 169
npSegment, 200
npSelector, 163
npSemiBackSet, 193
npSemiListing, 193
npSigDecl, 157
npSigItem, 156
npSigItemList, 154
npSignature, 154
npSignatureDefinee, 156
npSingleRule, 194
npSLocalItem, 172
npSQualTypelist, 181
npState, 212
npStatement, 170
npSuch, 150
npSuchThat, 176
npSum, 201
npSymbolVariable, 205
npSynonym, 473
npSynthetic, 198
npSystem, 472
npTagged, 218
npTerm, 201
npTrap, 212
npTrapForm, 212
npTuple, 146
npType, 149
npTypedForm, 220
npTypedForm1, 218
npTypeStyle, 219
npTypeVariable, 156
npTypeVariableList, 157
npTypified, 218
npTyping, 168
npVariable, 213
npVariableList, 213
npVariableName, 213
npVoid, 179
npWConditional, 195
npWhile, 177
npWith, 150
npZeroOrMore, 177
nRtEvalDomain, 1079
ofCategory, 419
oldCompLookup, 1079
oldHistFileName, 580
oldParseString, 1295
om-bindTCP, 1141
om-closeConn, 1140
pf2Sex, 299
pf2Sex1, 300
pfAbSynOp, 412
pfAbSynOp?, 412
pfAdd, 252
pfAnd, 253
pfAnd?, 254
pfAndLeft, 254
pfAndRight, 255
pfAppend, 255
pfApplication, 253
pfApplication2Sex, 305
pfApplication?, 255
pfApplicationArg, 254
pfApplicationOp, 254
pfAssign, 255
pfAssign?, 256
pfAssignLhsItems, 256
pfAssignRhs, 256
pfAttribute, 253
pfBrace, 257
pfBraceBar, 257
pfBracket, 257
pfBracketBar, 257
pfBreak, 258
pfBreak?, 258
pfBreakFrom, 258
pfCharPosn, 235
pfCheckArg, 241
pfCheckId, 241
pfCheckItOut, 239
pfCheckMacroOut, 240
pfCoerceto, 259
pfCoerceto?, 259
pfCoercetoExpr, 259
pfCoercetoType, 259
pfCollect, 260
pfCollect1?, 242
pfCollect2Sex, 314
pfCollect?, 260
pfCollectArgTran, 317
pfCollectBody, 260
pfCollectIterators, 260
pfCollectVariable1, 242
pfCopyWithPos, 246
pfDefinition, 261
pfDefinition2Sex, 315
pfDefinition?, 261
pfDefinitionLhsItems, 261
pfDefinitionRhs, 261
pfDo, 262
pfDo?, 262
pfDoBody, 262
pfDocument, 246
pfEnSequence, 263
pfExit, 263
pfExit?, 263
pfExitCond, 263
pfExitExpr, 264
pfExport, 264
pfExpression, 264
pfFileName, 236
pfFirst, 264
pfFix, 265
pfFlattenApp, 241
pfForin, 266
pfForin?, 266
pfForinLhs, 267
pfForinWhole, 267
pfFree, 265
pfFree?, 265
pfFreeItems, 266
pfFromDom, 267
pfFromdom, 268
pfFromdom?, 268
pfFromdomDomain, 269
pfFromdomWhat, 268
pfGlobalLinePosn, 235
pfHide, 269
pfId, 246
pfId?, 246
pfIdPos, 246
pfIdSymbol, 247
pfff, 269
pfIf?, 269
pfIfCond, 270
pfIfElse, 270
pfIfThen, 270
pfIfThenOnly, 270
pfImport, 271
pfInfApplication, 271
pfInline, 272
ptimers, 859
punctuation?, 118
put, 1039
putDatabaseStuff, 365
putFTText, 379
putHist, 590
pvarCondList, 1281
pvarCondList1, 1281
pvarPredTran, 322
pvarsOfPattern, 1282
qenum, 1046
qsquotient, 1067
qremainder, 1067
queryClients, 510
queueUpErrors, 372
quit, 638
quitSpad2Cmd, 638
read, 642
readHiFi, 601
readSpad2Cmd, 642
readSpadProfileIfThere, 961
reclaim, 37
recordAndPrint, 54
recordFrame, 923
recordNewValue, 591
recordNewValue0, 591
recordOldValue, 592
recordOldValue0, 592
reduceAlistForDomain, 1362
redundant, 374
remainder2, 1087
remFile, 357
remLine, 362
removeOption, 861
remover, 897
removeTracedMapSigs, 864
removeUndoLines, 933
renamePatternVariables, 1278
renamePatternVariables1, 1278
rep, 371
replaceFile, 985
replacePercentByDollar, 1256
replacePercentByDollar, fn, 1255
replaceSharps, 958
reportAO, 1348
reportCategory, 1347
reportInstantiations, 1053
reportOperations, 815
reportOpsFromLisplib, 818
reportOpsFromLisplib0, 817
reportOpsFromLisplib1, 817
reportOpsFromUnitDirectly, 821
reportOpsFromUnitDirectly0, 821
reportOpsFromUnitDirectly1, 825
reportSpadTrace, 892
reportUndo, 927
reportWhatOptions, 941
reroot, 38
resetCounters, 858
resetHashTables, 1000
resetInCoreHist, 588
resetSpacers, 858
resetStackLimits, 19
resetTimers, 858
resetWorkspaceVariables, 654
restart0, 16
restoreHistory, 597
retract, 1064
rread, 605
ruleLhsTran, 322
rulePredicateTran, 319
runspad, 19
rwrite, 604
safeWritify, 606
sameMsg?, 376
sameUnionBranch, 59
satisfiesRegularExpressions, 943
satisfiesUserLevel, 451
saveDependentsHashTable, 1022
saveHistory, 595
saveMapSig, 853
saveSystem, 650
saveUsersHashTable, 1022
sayBrightly1, 1049
sayExample, 539
INDEX

sayKeyedMsg, 329
sayKeyedMsgLocal, 330
sayMSG, 331
sayMSG2File, 331
sayShowWarning, 826
scanCheckRadix, 134
scanCloser?, 125
scanComment, 116
scanDictCons, 137
scanError, 135
scanEscape, 135
scanExponent, 126
scanIgnoreLine, 113
scanInsert, 138
scanKeyTableCons, 136
scanKeyTr, 120
scanNegComment, 117
scanNumber, 132
ScanOrPairVec, 616
ScanOrPairVec, ScanOrInner, 615
scanPossFloat, 121
scanPunCons, 139
scanS, 131
scanSpace, 129
scanString, 130
scanToken, 114
scanTransform, 132
scanW, 128
scanWord, 126
search, 964
searchCurrentEnv, 964
searchTailEnv, 965
sec, 1104
sech, 1106
segmentKeyedMsg, 330
selectOption, 479
selectOptionLC, 479
separatePiles, 341
serverReadLine, 42
set, 808
set-restart-hook, 13
setL, 808
setCurrentLine, 40
setdatabase, 1009
setExpose, 697
setExposeAdd, 698
setExposeAddConstr, 700
setExposeAddGroup, 699
setExposeDrop, 701
setExposeDropConstr, 703
setExposeDropGroup, 702
setFortDir, 726
setFortPers, 756
setFortTmpDir, 724
setFunctionsCache, 708
setHistoryCore, 584
setInputLibrary, 667
setIOindex, 599
setLinkerArgs, 728
setMsgCatlessAttr, 365
setMsgForcedAttr, 364
setMsgForcedAttrList, 364
setMsgPrefix, 349
setMsgText, 349
setMsgUnforcedAttr, 367
setMsgUnforcedAttrList, 366
setNagHost, 755
setOutputAlgebra, 763
setOutputCharacters, 766
setOutputFormula, 790
setOutputFortran, 770
setOutputHtml, 781
setOutputLibrary, 664
setOutputMathml, 776
setOutputOpenMath, 785
setOutputTex, 796
setStreamsCalculate, 801
setUpDefault, 1275
shortenForPrinting, 891
show, 814
showdatabase, 1008
showInOut, 600
showInput, 599
showMsgPos?, 357
showSpad2Cmd, 814
shut, 982
size, 1045
spad, 18
spad-error-loc, 970
spad-long-error, 969
spad-save, 989
spad-short-error, 970
spad-syntactic-error, 969
spad2BootCoerce, 1066
spadClosure?, 611
SpadInterpretStream, 25
spadReply, 895
spadReply/printName, 894
spadrread, 605
spadrwrite, 605
spadrwrite0, 604
spadStartUpMsgs, 17
spadTrace, 876
spadTrace,g, 875
spadTrace/isTraceable, 875
spadTraceAlias, 891
spadUntrace, 895
specialChar, 980
spleI, 122
spleI1, 122
splitIntoOptionBlocks, 447
stackTraceOptionError, 861
startsComment?, 115
startsNegComment?, 117
statisticsInitialization, 1035
streamChop, 70
StreamNull, 333
stringize, 1262
stringList2String, 1248
stringMatches?, 1090
StringToDir, 1091
stripLisp, 471
stripSpaces, 471
strpos, 1045
strposl, 1046
stupidIsSpadFunction, 906
subMatch, 119
substFromAlist, 1280
substringMatch, 119
subTypes, 866
summary, 830
syGeneralErrorHere, 192
syIgnoredFromTo, 191
synonym, 832
synonymsForUserLevel, 833
synonymSpad2Cmd, 832
sySpecificErrorAtToken, 192
sySpecificErrorHere, 192
systemCommand, 448
tabling, 362
templateParts, 1284
terminateSystemCommand, 452
tersyscommand, 452
thefname, 89
theid, 88
theorigin, 86
thisPosIsEqual, 374
thisPosIsLess, 374
throwEvalTypeMsg, 919
To, 380
toFile?, 363
tokConstruct, 411
token-stack-show, 971
tokPart, 413
tokPosn, 413
tokTran, 467
tokType, 413
topLevelInterpEval, 1372
toScreen?, 351
trace, 847
trace1, 848
traceDomainConstructor, 880
traceDomainLocalOps, 880
tracelet, 904
traceOptionError, 857
traceReply, 899
traceSpad2Cmd, 847
trademark, 523
translateTrueFalse2YesNo, 659
translateYesNo2TrueFalse, 658
translateYesNoToTrueFalse, 1313
transOption, 860
transTraceItem, 863
typeCheckInputAreas, 1290
unAbbreviateKeyword, 469
undo, 922
undoChanges, 593
undoCount, 929
undoFromFile, 594
undoInCore, 592
undoLocalModemapHack, 933
undoSingleStep, 931
INDEX

undoSteps, 930
unescapeStringsInForm, 61
unifyStruct, 428
unifyStructVar, 429
unparseInputForm, 1054
untrace, 862
untraceDomainConstructor, 883
untraceDomainConstructor, keepTraced?, 882
untraceDomainLocalOps, 880
untraceMapSubNames, 873
unwritable?, 606
updateCurrentInterpreterFrame, 559
updateDatabase, 1017
updateFromCurrentInterpreterFrame, 558
updateHist, 589
updateInCoreHist, 590
updateSourceFiles, 546
userLevelErrorMessage, 450
validateOutputDirectory, 724
vec2list, 1064
voidValue, 1054
what, 939
whatCommands, 941
whatConstructors, 945
whatSpad2Cmd, 940
whatSpad2Cmd, fixpat, 939
whichCat, 365
with, 947
workfiles, 949
workfilesSpad2Cmd, 949
wrap, 1043
write-browsedb, 1029
write-categorydb, 1030
write-interpdb, 1027
write-operationdb, 1031
write-warmdata, 1032
writeHiFi, 602
writeHistModesAndValues, 603
writeInputLines, 587
writify, 610
writify, writifyInner, 607
writifyComplain, 606
xlCannotRead, 90
xlCmdBug, 96
xlConActive, 91
xlConsole, 92
xlConStill, 92
xlFileCycle, 90
xlffBug, 95
xlffSyntax, 94
xmlMsg, 84
xlnoSuchFile, 89
xlok, 84
xlok1, 85
xlpPrematureEOF, 84
xlpPrematureFin, 93
xlsay, 88
xlskip, 87
xlskippingFin, 93
yesanswer, 537
zeroOneTran, 66
zsystemdevelopment, 953
zsystemdevelopment1, 954
zsystemDevelopmentSpad2Cmd, 953
defvar
*allOperations*, 1000
*allconstructors*, 1000
*browse-stream*, 999
*browse-stream-stamp*, 999
*category-stream*, 999
*category-stream-stamp*, 1000
*defaultdomain-list*, 996
*eof*, 22
*hasCategory-hash*, 997
*interp-stream*, 998
*interp-stream-stamp*, 998
*miss*, 997
*monitor-domains*, 1163
*monitor-nrlibs*, 1163
*monitor-table*, 1164
*msghash*, 327
*operation-hash*, 997
*operation-stream*, 998
*operation-stream-stamp*, 999
*whitespace*, 22
/editfile, 515
$nagMessages, 758
$reportBottomUpFlag, 747
$BreakMode, 661
$CommandSynonymAlist, 478
$EndServerSession, 42
$HTCompanionWindowID, 50
$HiFiAccess, 733
$IOIndex, 10
$InitialCommandSynonymAlist, 476
$InitialModemapFrame, 7
$InteractiveMode, 22
$NeedToSignalSessionManager, 42
$NonNullStream, 611
$NullStream, 611
$ProcessInteractiveValue, 50
$QuietCommand, 45
$RTspecialCharacters, 978
$SpadServerName, 10
$SpadServer, 10
$UserLevel, 807
$abbreviateTypes, 760
$activePageList, 1249
$algebraFormat, 761
$algebraOutputFile, 762
$algebraOutputStream, 762
$attrCats, 364
$bcParseOnly, 1249
$boot, 23
$cacheAlist, 707
$cacheMessages, 326
$clearExcept, 499
$clearOptions, 499
$compileDontDefineFunctions, 712
$compileRecurrence, 712
$constructors, 899
$curpage, 1249
$current-directory, 5
$currentFrameNum, 41
$dalymode, 663
$defaultFortranType, 717
$defaultMsgDatabaseName, 6
$defaultSpecialCharacters, 975
$describeOptions, 528
$directory-list, 6
$displayDroppedMap, 738
$displayMsgNumber, 746
$displayOptions, 535
$displaySetValue, 748
$displayStartMsgs, 749
$domPVar, 47
$envHashTable, 963
$formulaFormat, 789
$formulaOutputFile, 789
$fortIndent, 715
$fortInts2Floats, 714
$fortLength, 716
$fortPersistence, 756
$fortranArrayStartingIndex, 722
$fortranDirectory, 726
$fortranFormat, 769
$fortranLibraries, 728
$fortranOptimizationLevel, 721
$fortranOutputFile, 769
$fortranPrecision, 718
$fortranSegment, 720
$fortranTmpDir, 723
$fractionDisplayType, 773
$frameAlist, 41
$frameMessages, 741
$frameNumber, 41
$frameRecord, 921
$fullScreenSysVars, 730
$functionTable, 502
$funnyBacks, 1297
$funnyQuote, 1297
$genValue, 51
$giveExposureWarning, 739
$globalExposureGroupAlist, 670
$highlightAllowed, 742
$historyDirectory, 579
$historyDisplayWidth, 731, 1324
$historyFileType, 579
$htLineList, 1249
$htmlFormat, 780
$htmlOutputFile, 780
$imPrGuys, 358
$imPrTagGuys, 367
$inputPromptType, 746
$insertCoerceFailure, 30
$insertRestart, 24
$insertSpadReader, 30
$insertTopLevel, 24
$interpOnly, 47
$lambdaType, 662
$library-directory-list, 7
$linearFormatScripts, 793
$linelength, 774
INDEX

$localExposureDataDefault, 696
$localExposureData, 697
$margin, 774
$mathmlFormat, 775
$mathmlOutputFile, 775
$maximumFortranExpressionLength, 720
$minivectorNames, 47
$msgAlist, 326
$msgDatabaseName, 7
$msgdbNoBlanksAfterGroup, 328
$msgdbNoBlanksBeforeGroup, 328
$msgdbPrims, 327
$msgdbPunct, 327
$nagEnforceDouble, 758
$nagHost, 754
$nagMessages, 745
$ncMsgList, 25
$newcompErrorCount, 26
$newline, 1324
$noEvalTypeMsg, 919
$noParseCommands, 444
$noSubsumption, 962
$nopos, 26
$npTokToNames, 204
$numberOfEquations, 1226
$numericFailure, 1080
$oldBreakMode, 1080
$oldHistoryFileName, 578
$oldline, 449
$openMathFormat, 784
$openMathOutputFile, 785
$openServerIfTrue, 8
$optionAlist, 846
$pfMacros, 97
$plainRTspecialCharacters, 978
$plainSpecialCharacters0, 976
$plainSpecialCharacters1, 976
$plainSpecialCharacters2, 977
$plainSpecialCharacters3, 977
$preLength, 355
$prettyprint, 806
$previousBindings, 921
$printAnyITrue, 735
$printFortranDecs, 716
$printLoadMsgs, 736
$printMsgsToFile, 740
$printStatisticsSummaryIfTrue, 749
$printTimeIfTrue, 751
$printTypeIfTrue, 752
$printVoidIfTrue, 753
$promptMsg, 26
$quitCommandType, 800
$quitTag, 18
$relative-directory-list, 8
$relative-library-directory-list, 9
$repGuys, 375
$reportBottomUpFlag, 737
$reportCoerceIfTrue, 738
$reportCompilation, 804
$reportInstantiations, 742
$reportInterpOnly, 744
$reportOptimization, 805
$reportSpadtrace, 846
$reportundo, 922
$setOptionNames, 808
$sockBufferLength, 42
$solutionMethod, 1226
$spad-errors, 967
$spadroot, 9
$specialCharacterAlist, 979
$specialCharacters, 979
$streamCount, 801
$streamsShowAll, 803
$syscommands, 444
$systemCommands, 443
$systemType, 1225
$testingErrorPrefix, 326
$testingSystem, 750
$txFormatting, 327
$txFormat, 795
$txOutputFile, 796
$toWhereGuys, 363
$tokenCommands, 475
$traceNoisely, 846
$traceOptionList, 846
$tracedMapSignatures, 846
$underbar, 586
$undoFlag, 921
$useEditorForShowOutput, 794
$useFullScreenHelp, 732
$useInternalHistoryTable, 579
$useIntrinsicFunctions, 719
$whatOptions, 939
boot-line-stack, 961
CLOSEPAREN, 107
creditlist, 3
curinstream, 21
curoutstream, 21
DOT, 106
dotdot, 323
echo-meta, 962
Else?, 78
Elseif?, 78
ElseifKeepPart, 76
ElseifSkipPart, 76
ElseifSkipToEnd, 76
ElseKeepPart, 77
ElseSkipToEnd, 77
erroринstream, 21
erroroutstream, 22
ESCAPE, 105
EXPONENT1, 107
EXPONENT2, 107
le-closed, 962
If?, 77
IfKeepPart, 76
IfSkipPart, 76
IfSkipToEnd, 75
in-stream, 961
incCommands, 96
infgeneric, 110
input-libraries, 668
KeepPart?, 79
line-handler, 967
MINUSCOMMENT, 106
npPParg, 209
out-stream, 962
output-library, 665
PLUSCOMMENT, 106
QUESTION, 107
RADIXCHAR, 106
scanCloser, 125
scanDict, 137
scanKeyTable, 136
scanKeyWords, 108
scanPun, 139
SkipEnd?, 78
SkipPart?, 79
Skipping?, 79
SPACE, 105
StreamNil, 102
STRINGCHAR, 105
Top, 75
Top?, 77
xtokenreader, 968
delasc
calledby spadUntrace, 896
Delay, 102
calledby incAppend, 85
calledby incIgen, 73
calledby incLude, 75
calledby incRgen, 101
calledby incZip, 72
calledby next, 36
defun, 102
deldatabase, 1009
calledby abbreviationsSpad2Cmd, 483
defun, 1009
delete
calledby breaklet, 905
calledby domainDescendantsOf, 1349
calledby setExposeAddConstr, 700
calledby setExposeDropConstr, 704
calledby setExposeDropGroup, 702
calledby trace1, 848
calledby tracelet, 904
calledby untraceDomainConstructor, 883
calledby workfilesSpad2Cmd, 949
deleteAssoc
calledby clearCmdParts, 505
deleteFile, 1042
calledby clearCmdAll, 503
calls erase, 1042
calls pathname, 1043
uses $erase, 1043
defun, 1042
delt, 1082
defmacro, 1082
descendantsOf
calledby kcdPage, 1362
describe, 528
calls describespad2cmd, 528
defun, 528
describe help page, 527
describeFortPersistence, 757
called by setFortPers, 756
calls sayBrightly, 757
uses $fortPersistence, 757
defun, 757
describeInputLibraryArgs, 668
called by setInputLibrary, 667
calls sayBrightly, 668
defun, 668
describeOutputLibraryArgs, 665
called by setOutputLibrary, 664
calls sayBrightly, 665
defun, 665
describeSetFortDir, 727
called by setFortDir, 726
calls sayBrightly, 727
uses $fortranDirectory, 727
defun, 727
describeSetFortTmpDir, 725
called by setFortTmpDir, 724
calls sayBrightly, 725
uses $fortranTmpDir, 725
defun, 725
describeSetLinkerArgs, 729
called by setLinkerArgs, 728
calls sayBrightly, 729
uses $fortranLibraries, 729
defun, 729
describeSetNagHost, 755
called by setNagHost, 755
calls sayBrightly, 755
uses $nagHost, 755
defun, 755
describeSetOutputAlgebra, 765
called by setOutputAlgebra, 763
calls sayBrightly, 765
calls setOutputAlgebra, 765
defun, 765
describeSetOutputFormula, 792
called by setOutputFormula, 790
calls sayBrightly, 792
calls setOutputFormula, 792
defun, 792
describeSetOutputFortran, 772
called by setOutputFortran, 770
calls sayBrightly, 772
calls setOutputFortran, 772
defun, 772
describeSetOutputHtml, 783
called by setOutputHtml, 781
calls sayBrightly, 783
calls setOutputHtml, 783
defun, 783
describeSetOutputMathml, 778
called by setOutputMathml, 776
calls sayBrightly, 778
calls setOutputMathml, 778
defun, 778
describeSetOutputOpenMath, 788
called by setOutputOpenMath, 785
calls sayBrightly, 788
calls setOutputOpenMath, 788
defun, 788
describeSetOutputTex, 798
called by setOutputTex, 796
calls sayBrightly, 798
calls setOutputTex, 798
defun, 798
describeSetStreamsCalculate, 802
called by setStreamsCalculate, 801
calls sayKeyedMsg, 802
uses $streamCount, 802
defun, 802
describeSpad2Cmd, 528
calls cleanline, 529
calls flatten, 529
calls getdatabase, 529
calls sayMessage, 529
calls selectOptionLC, 529
uses $EmptyEnvironment, 529
uses $describeOptions, 529
uses $e, 529
defun, 528
describeSpad2Cmd
called by describe, 528
desiredMsg, 352
called by ncHardError, 352
called by ncSoftError, 351
defun, 352
devaluate
called by /tracerply, 894
called by ?t, 903
called by addTraceItem, 902
called by dbAddChainDomain, 1386
called by dbSearchOrder, 1356
called by mkEvalable, 913
called by prTraceNames,fn, 898
called by printTypeAndTimeSaturn, 58
called by shortenForPrinting, 891
called by spadReply,printName, 894
called by spadUntrace, 895
called by trace1, 848
called by transTraceItem, 863
called by untraceDomainConstructor,keep?, 882
called by writify,writifyInner, 607
dewritify, 615
called by SPADRREAD, 605
calls ScanOrPairVec, 615
calls dewritify,dewritifyInner, 615
calls function, 615
uses $seen, 615
defun, 615
dewritify,dewritifyInner, 612
called by dewritify,dewritifyInner, 612
called by dewritify, 615
calls concat, 612
calls dewritify,dewritifyInner, 612
calls error, 612
calls exit, 612
calls gensymmer, 612
calls hget, 612
calls hput, 612
calls intp, 612
calls make-instream, 612
calls poundsign, 612
calls qcar, 612
calls qcdr, 612
calls qrplaca, 612
calls qrplacd, 612
calls qsetvelt, 612
calls qvelt, 612
calls qvmaxindex, 612
calls seq, 612
calls spaddiff, 612
calls vcep, 612
calls vmread, 612
uses $NonNullStream, 612
uses $NullStream, 612
uses $seen, 612
defun, 612
DFAcos, 1099
defmacro, 1099
DFAcosh, 1102
defmacro, 1102
DFAdd, 1095
defmacro, 1095
DFAsin, 1099
defmacro, 1099
DFAtanh, 1101
defmacro, 1101
DFAtan, 1100
defmacro, 1100
DFAtan2, 1100
defmacro, 1100
DFAtan, 1102
defmacro, 1102
DFCos, 1099
defmacro, 1099
DFCosh, 1101
defmacro, 1101
DFDivide, 1096
defmacro, 1096
DFEq, 1096
defmacro, 1096
DFExp, 1098
defmacro, 1098
DFExpt, 1098
defmacro, 1098
DFinishDivide, 1097
defmacro, 1097
DFinishExpt, 1098
defmacro, 1098
DFinishMultiply, 1095
defmacro, 1095
DFLessThan, 1094
defmacro, 1094
DFLog, 1097
defmacro, 1097
DFLogE, 1097
defmacro, 1097
DFMax, 1096
defmacro, 1096
INDEX

DFMin, 1096
defmacro, 1096
DFMinusp, 1094
defmacro, 1094
DFMultiply, 1095
defmacro, 1095
DFSin, 1098
defmacro, 1098
DFSinh, 1100
defmacro, 1100
DFSqrt, 1097
defmacro, 1097
DFSubtract, 1095
defmacro, 1095
DFTan, 1099
defmacro, 1099
DFTanh, 1101
defmacro, 1101
DFUnaryMinus, 1094
defmacro, 1094
DFZerop, 1094
defmacro, 1094
diffAlist, 924
calledby recordFrame, 923
calls assoc, 925
calls assq, 925
calls exit, 925
calls lassq, 925
calls reportUndo, 925
calls seq, 925
calls tmp1, 925
defun, 924
dig2fix
calledby isSharpVarWithNum, 886
digit-char-p
calledby digits2Names, 1405
digit?, 122
calledby scanExponent, 126
calledby scanPossFloat, 121
calledby scanToken, 114
calls digitp, 122
defun, 122
digitp, 1045
calledby digit?, 122
calledby digitp, 1045
calledby isSharpVarWithNum, 886
calledby isgenvar, 887
calledby mkConArgSublis, 1405
calls digitp, 1045
defun, 1045
digits2Names, 1405
calledby mkConArgSublis, 1405
calls concat, 1405
calls digit-char-p, 1405
defun, 1405
DirToString, 1091
calledby fnameDirectory, 1091
defun, 1091
disableHist, 603
calledby fetchOutput, 600
calledby historySpad2Cmd, 582
calledby restoreHistory, 597
calledby setHistoryCore, 585
calledby showInOut, 600
calledby showInput, 599
calledby undoFromFile, 594
calledby undoInCore, 593
calledby updateHist, 589
calls histFileErase, 603
calls histFileName, 603
uses $HiFiAccess, 603
defun, 603
display, 535
calls displaypad2cmd, 535
defun, 535
display help page, 533
manpage, 533
displayCondition, 465
calledby displayProperties, 461
calls bright, 465
calls concat, 465
calls pred2English, 465
calls sayBrightly, 465
defun, 465
displayDomainOp
calledby dbShowConsDoc1, 1393
displayExposedConstructors, 705
calledby setExposeAddConstr, 701
calledby setExposeAddGroup, 699
calledby setExposeAdd, 698
calledby setExposeDropConstr, 704
calledby setExposeDropGroup, 702
called by setExpose, 697
calls centerAndHighlight, 705
calls sayKeyedMsg, 705
uses $localExposureData, 705
defun, 705
displayExposedGroups, 705
called by setExposeAddGroup, 699
called by setExposeAdd, 698
called by setExposeDropGroup, 702
called by setExpose, 697
calls centerAndHighlight, 705
calls sayKeyedMsg, 705
uses $interpreterFrameName, 705
uses $localExposureData, 705
defun, 705
displayFrameNames, 563
called by frameSpad2Cmd, 566
calls bright, 563
calls framename, 563
calls sayKeyedMsg, 563
uses $interpreterFrameRing, 563
defun, 563
displayHiddenConstructors, 705
called by setExposeAddGroup, 699
called by setExposeDropConstr, 704
called by setExposeDropGroup, 702
called by setExposeDrop, 701
called by setExpose, 697
calls centerAndHighlight, 705
calls sayKeyedMsg, 705
uses $localExposureData, 706
defun, 705
displayMacro, 453
called by displayMacros, 538
called by displayProperties, 462
calls bright, 454
calls isInterpMacro, 453
calls mathprint, 454
calls object2String, 454
calls sayBrightly, 454
calls strconc, 454
uses $op, 454
defun, 453
displayMacros, 538
called by displaySpad2Cmd, 535
calls displayMacro, 538
calls displayParserMacro, 538
calls exit, 538
calls getInterpMacroNames, 538
calls getParserMacroNames, 538
calls member, 538
calls remdup, 538
calls sayBrightly, 538
calls seq, 538
defun, 538
displayMode, 466
called by displayProperties, 461
calls bright, 466
calls concat, 466
calls fixObjectForPrinting, 466
calls prefix2String, 467
calls sayBrightly, 466
defun, 466
displayModemap, 466
called by displayProperties, 462
calls bright, 466
calls concat, 466
calls formatSignature, 466
calls sayBrightly, 466
defun, 466
displayOperations, 537
called by displaySpad2Cmd, 535
calls reportOpSymbol, 537
calls sayKeyedMsg, 537
calls yesanswer, 537
defun, 537
displayOperationsFromLispLib, 820
called by reportOpsFromLispLib, 818
calls centerAndHighlight, 820
calls eqsubstlist, 820
calls formatOperationAlistEntry, 821
calls getdatabase, 820
calls msort, 820
calls remdup, 820
calls reportOpsFromUnitDirectly, 820
calls say2PerLine, 821
calls specialChar, 820
uses $FormalMapVariableList, 821
uses $linelength, 821
defun, 820
displayParserMacro, 464
called by displayMacros, 538
displayProperties, 461
called by displaySpad2Cmd, 535
calls bright, 462
calls displayCondition, 461
calls displayMacro, 462
calls displayModemap, 462
calls displayMode, 461
calls displayParserMacro, 462
calls displayProperties, sayFunctionDeps, 462
calls displayType, 461
calls displayValue, 461
calls exit, 462
calls fixObjectForPrinting, 462
calls getAndSay, 461
calls getIPropList, 461
calls getInterpMacroNames, 461
calls getI, 461
calls getParserMacroNames, 461
calls getWorkspaceNames, 461
calls interpFunctionDepAlists, 461
calls isInternalMapName, 461
calls isInterpMacro, 462
calls member, 462
calls msort, 461
calls prefix2String, 462
calls qcar, 461
calls qcdr, 461
calls remdup, 461
calls sayKeyedMsg, 461
calls sayMSG, 462
calls seq, 462
calls terminateSystemCommand, 462
uses $dependeeAlist, 462
uses $dependentAlist, 462
defun, 461
defun, 461
calls getalists, 458
calls sayMSG, 458
calls seq, 458
uses $dependeeAlist, 458
uses $dependentAlist, 458
defun, 456
calls bright, 462
calls exit, 462
calls getalist, 458
calls sayMSG, 458
calls seq, 458
calls sayMSG, 458
calls sayMessage, 655
calls specialChar, 655
calls translateTrueFalse2YesNo, 655
uses $linelength, 655
defun, 655
calls eval, 655
calls literals, 655
calls object2String, 655
calls sayBrightly, 655
calls sayMSG, 655
calls sayMessage, 655
calls specialChar, 655
calls translateTrueFalse2YesNo, 655
uses $linelength, 655
defun, 655
calls eval, 657
calls literals, 657
calls object2String, 657
calls satisifiesUserLevel, 657
calls sayBrightly, 657
calls say, 657
calls spaddifference, 657
calls specialChar, 657
calls translateTrueFalse2YesNo, 657
calls tree, 657
uses $linelength, 657
defun, 657
calls getalists, 458
calls sayMSG, 458
calls seq, 458
calls $dependeeAlist, 458
uses $dependentAlist, 458
defun, 456
calls bright, 462
calls displayCondition, 461
calls displayMacro, 462
calls displayModemap, 462
calls displayMode, 461
calls displayParserMacro, 462
calls displayProperties, sayFunctionDeps, 462
calls displayType, 461
calls displayValue, 461
calls exit, 462
calls fixObjectForPrinting, 462
calls getAndSay, 461
calls getIPropList, 461
calls getInterpMacroNames, 461
calls getI, 461
calls getParserMacroNames, 461
calls getWorkspaceNames, 461
calls interpFunctionDepAlists, 461
calls isInternalMapName, 461
calls isInterpMacro, 462
calls member, 462
calls msort, 461
calls prefix2String, 462
calls qcar, 461
calls qcdr, 461
calls remdup, 461
calls sayKeyedMsg, 461
calls sayMSG, 462
calls seq, 462
calls terminateSystemCommand, 462
uses $dependeeAlist, 462
uses $dependentAlist, 462
defun, 461
defun, 461
INDEX

domArg2, 422
called by hasCaty, 420
calls isSharpVar, 422
calls subCopy, 422
local ref $domPvar, 422
defun, 422
done
called by insertPos, 379
doSystemCommand, 446
called by ExecuteInterpSystemCommand, 31
calls concat, 446
calls expand-tabs, 446
calls getFirstWord, 446
calls handleNoParseCommands, 446
calls handleParsedSystemCommands, 446
calls handleTokensizeSystemCommands, 446
calls member, 446
calls processSynonyms, 446
calls splitIntoOptionBlocks, 446
calls substring, 446
calls unAbbreviateKeyword, 446
uses $noParseCommands, 446
uses $tokenCommands, 446
uses line, 446
defun, 446
DOT, 106
defvar, 106
dotdot, 323
defvar, 323
downcase
called by apropos, 945
called by conPage, 1343
called by dbShowCons, 1388
called by koaPageFilterByName, 1380
called by selectOptionLC, 479
called by set1, 809
called by setOutputCharacters, 767
called by whatSpad2Cmd, fixpat, 939
downlink, 1324
called by conPage, 1343
defun, 1324
downlinkSaturn, 1324
defun, 1324
dqAppend, 344
called by dqConcat, 343
called by pileCtree, 340
defun, 344
dqConcat, 343
called by dqConcat, 343
called by enPile, 340
called by separatePiles, 341
calls dqAppend, 343
calls dqConcat, 343
defun, 343
dqToList, 344
called by intloopEchoParse, 67
called by ncloopParse, 36
defun, 344
dqUnit, 343
called by enPile, 340
called by lineoftoks, 111
called by scanToken, 114
called by separatePiles, 341
defun, 343
drop[9]
called by frameSpad2Cmd, 566
dropInputLibrary, 669
called by addInputLibrary, 668
called by openOutputLibrary, 666
called by setInputLibrary, 667
uses input-libraries, 669
defun, 669
dropLeadingBlanks
called by processSynonymLine, removeKeyFromLine, 834
dsetaref2, 1084
defmacro, 1084
dsetelt, 1082
defmacro, 1082
dumbTokenize, 467
called by handleParsedSystemCommands, 468
called by handleTokensizeSystemCommands, 447
called by parseSystemCmd, 469
calls stripSpaces, 467
defun, 467
echo-meta, 962
defvar, 962
INDEX

defun, 339
eqsubstlist
called by displayOperationsFromLispLib, 820
called by reportOpsFromLispLib, 818
erase
called by deleteFile, 1042
called by reportOpsFromLispLib1, 817
called by reportOpsFromUnitDirectly1, 825
called by saveDependentsHashTable, 1022
called by saveUsersHashTable, 1022
erMsgCompare, 370
calls compareposns, 370
calls getMsgPos, 370
defun, 370
erMsgSep, 370
called by erMsgSort, 369
calls getMsgPos, 370
calls poNopos?, 370
defun, 370
erMsgSort, 369
called by processMsgList, 369
calls erMsgSep, 369
calls listSort, 369
defun, 369
error
called by basicLookupCheckDefaults, 1078
called by basicLookup, 1076
called by charDigitVal, 617
called by dewritify, dewritifyInner, 612
called by gensymInt, 616
called by myWritable?, 1093
errorinStream, 21
defvar, 21
erroroutStream, 22
defvar, 22
errorPage
called by kcPage, 1357
called by kcaPage1, 1363
called by kccPage, 1364
called by kcnPage, 1368
called by kcpPage, 1361
called by kePage, 1351
called by kiPage, 1351
called by koPage, 1378
called by ksPage, 1355
ESCAPE, 105
defvar, 105
escapeSpecialChars
called by reportAO, 1348
eval
called by NRTevalDomain, 1079
called by displaySetOptionInformation, 655
called by displaySetVariableSettings, 657
called by evalDomain, 913
evalCategory, 959
called by evaluateType1, 918
calls isPartialMode, 959
calls ofCategory, 959
defun, 959
evalDomain, 913
called by NRTevalDomain, 1079
called by domainToGenvar, 861
called by reportOpsFromUnitDirectly, 822
calls concat, 913
calls eval, 913
calls mkEvalable, 913
calls prefix2String, 913
calls sayMSG, 913
calls startTimingProcess, 913
calls stopTimingProcess, 913
local ref $evalDomain, 913
defun, 913
evaluateSignature, 920
called by evaluateType, 916
calls evaluateType, 920
defun, 920
evaluateType, 916
called by evaluateSignature, 920
called by evaluateType1, 918
called by evaluateType, 916
called by reportOperations, 816
calls bottomUp, 916
calls constructor?, 916
calls evaluateSignature, 916
calls evaluateType, 916
calls getValue, 916
calls isDomainValuedVariable, 916
calls member, 916
calls mkAtree, 916
calls objVal, 916
calls qcar, 916
calls qcdr, 916
calls throwEvalTypeMsg, 916
local def $expandSegments, 916
local ref $EmptyMode, 916
defun, 916
evaluateType1, 917
calls bottomUp, 918
calls categoryForm?, 917
calls coerceOrRetract, 918
calls constructor?, 917
calls evalCategory, 918
calls evaluateType, 918
calls getAndEvalConstructorArgument, 918
calls getConstructorSignature, 918
calls getdatabase, 918
calls makeOrdinal, 918
calls mkAtree, 918
calls objValUnwrap, 918
calls putTarget, 918
calls qcar, 918
calls qcdr, 918
calls replaceSharps, 917
calls throwEvalTypeMsg, 917
calls throwKeyedMsgCannotCoerceWith-Value, 918
local ref $EmptyMode, 918
local ref $QuadSymbol, 918
defun, 917
executeInterpreterCommand, 1289
defun, 1289
ExecuteInterpSystemCommand, 31
calledby InterpExecuteSpadSystemCom- mand, 30
calls doSystemCommand, 31
calls intProcessSynonyms, 31
calls substring, 31
uses $currentLine, 31
defun, 31
executeQuietCommand, 45
calledby serverReadLine, 43
calls make-string, 46
calls parseAndInterpret, 46
calls sockGetString, 46
uses $MenuServer, 46
uses $QuietCommand, 46
catches, 45
defun, 45
exit
calledby /traceReply, 894
calledby abbreviationsSpad2Cmd, 483
calledby apropos, 945
calledby clearCmdParts, 505
calledby coerceSpadArgs2E, 865
calledby dewritify, dewritifyInner, 612
calledby diffList, 925
calledby displayMacros, 538
calledby displayProperties, sayFunctionDeps, 458
calledby displayProperties, 462
calledby flattenOperationList, 883
calledby funfind, LAM, 874
calledby getAliasIfTracedMapParameter, 889
calledby getBpiNameIfTracedMap, 890
calledby getPreviousMapSubNames, 870
calledby getOrderOption, 853
calledby getTraceOptions, 852
calledby getTraceOption, 854
calledby getWorkspaceNames, 455
calledby historySpad2Cmd, 583
calledby importFromFrame, 563
calledby isListOfIdentifiersOrStrings, 869
calledby isListOfIdentifiers, 868
calledby orderBySlotNumber, 893
calledby prTraceNames, fn, 898
calledby prTraceNames, 898
calledby recordFrame, 923
calledby removeUndoLines, 933
calledby reportUndo, 928
calledby runspad, 19
calledby set1, 809
calledby spadReply, printName, 894
calledby spadReply, 895
calledby spadTrace, isTraceable, 876
calledby spadTrace, 876
calledby spadUntrace, 896
calledby subTypes, 866
calledby trace1, 848
calledby traceDomainConstructor, 881
calledby traceReply, 899
calledby undoFromFile, 594
calledby undoLocalModemapHack, 933
INDEX

called by undoSingleStep, 931
called by untraceDomainConstructor, keepTraceDefVar, 962
     882
called by untraceDomainConstructor, 883
called by whatConstructors, 945
called by whatSpad2Cmd, 940
called by writify, writifyInner, 607
expand-tabs
called by doSystemCommand, 446
called by incLude1, 80

explainExact
called by finalExactRequest, 1242
explainLinear, 1241
called by linearFinalRequest, 1241
defun, 1241
EXPONENT1, 107
defvar, 107
EXPONENT2, 107
defvar, 107
extendLocalLibdb
called by library, 1013
extractFileNameFromPath
called by kdPageInfo, 1345
extractHasArgs
called by augmentHasArgs, 1365

fbpip
called by mkEvalable, 913
fetchKeyedMsg, 328
called by getKeyedMsg, 329
calls cacheKeyedMsg, 328
calls object2Identifier, 328
uses *msgHash*, 328
uses $defaultMsgDatabaseName, 328
defun, 328
fetchOutput, 600
calls assq, 600
calls boot-equal, 600
calls disableHist, 600
calls getI, 600
calls readHiFi, 600
calls spaddifference, 600
calls spaddifference, 600
defun, 600
file-closed, 962

used by init-boot/spad-reader, 968
filep
called by setOutputLibrary, 664
fillerSpaces, 18
called by displaySetVariableSettings, 657
called by justifyMyType, 60
called by printLabelledList, 474
called by spadStartUpMsgs, 17
calls ifcar, 18
defun, 18
filterAndFormatConstructors, 944
called by whatSpad2Cmd, 940
calls blankList, 944
calls centerAndHighlight, 944
calls filterListOfStringsWithFn, 944
calls function, 944
calls pp2Cols, 944
calls sayMessage, 944
calls specialChar, 944
calls whatConstructors, 944
uses $lineLength, 944
defun, 944

filterListOfStrings, 942
called by apropos, 945
called by whatCommands, 942
calls satisfiesRegularExpressions, 942
defun, 942

fin, 548
uses eof, 548
defun, 548
throws, 548
fin help page, 547
manpage, 547

finalExactRequest, 1241
called by beQueryInteger, 1241
calls explainExact, 1242
calls moreExactSolution, 1242
calls sayBrightly, 1241
defun, 1241
findfile
called by bcUnixTable, 1397
called by readSpad2Cmd, 642
findFrameInRing, 559
called by changeToNamedInterpreterFrame, 560
calls boot-equal, 559
calls frameName, 559
uses $interpreterFrameRing, 559
defun, 559
firstTokPosn, 341
called by enPile, 340
calls tokPosn, 341
defun, 341
fixObjectForPrinting, 456
called by clearCmdParts, 505
called by displayMode, 466
called by displayProperties, 462
called by displayType, 460
called by displayValue, 459
calls member, 456
calls object2Identifier, 456
calls pname, 456
calls strconc, 456
uses $msgdbPrims, 456
defun, 456
flatten, 531
called by describeSpad2Cmd, 529
defun, 531
flattenOperationAlist, 883
called by spadTrace, 876
calls exit, 883
calls seq, 883
defun, 883
float
called by ptimers, 859
float2Sex, 305
called by pfLiteral2Sex, 304
uses $useBFasDefault, 305
defun, 305
flowSegmentedMsg
called by msgOutputter, 353
called by msgText, 60
called by sayKeyedMsgLocal, 330
called by traceReply, 899
flung
used by intloopSpadProcess, 63
fnameDirectory, 1091
called by myWritable?, 1093
calls DirToString, 1091
defun, 1091
fnameExists?, 1092
called by myWritable?, 1093
defun, 1092
fnameMake, 1090
called by fnameNew, 1093
calls StringToDir, 1090
defun, 1090
fnameName, 1092
defun, 1092
fnameNew, 1093
calls fnameMake, 1093
defun, 1093
fnameReadable?, 1092
defun, 1092
fnameType, 1092
defun, 1092
fnameWritable?, 1093
calls myWriteable?, 1093
defun, 1093
for, 391
syntax, 391
form2HtString
called by conPage, 1343
called by dbConsHeading, 1395
called by dbSpecialDescription, 1398
called by dbSpecialExports, 1399
called by dbSpecialOperations, 1398
called by kcPage, 1357
called by kcaPage1, 1363
called by kccPage, 1364
called by kcnPage, 1368
called by kcpPage, 1361
called by kePage, 1352
called by koPage, 1378
called by ksPage, 1355
called by reportAO, 1348
form2LispString
called by libConstructorSig, 1407
form2String
called by displayValue, 459
called by kArgumentCheck, 1371
called by reportOpsFromLisplib, 818
INDEX

form2StringAsTeX
called by printTypeAndTimeSaturn, 58
form2StringWithWhere
called by reportOpsFromLisplib, 818
formatAttribute
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
formatOperation
called by reportOpsFromUnitDirectly, 822
formatOperationAlistEntry
called by displayOperationsFromLisplib, 821
formatOpType
called by reportOpsFromUnitDirectly, 822
formatSignature
called by compiledLookupCheck, 502
called by displayModemap, 466
frame, 565
calls frameSpad2Cmd, 565
defun, 565
frame help page, 549
manpage, 549
frameEnvironment, 556
called by importFromFrame, 563
calls frameInteractive, 556
defun, 556
frameInteractive
called by frameEnvironment, 556
frameName, 552
called by findFrameInRing, 559
defun, 552
framename
called by addNewInterpreterFrame, 561
called by closeInterpreterFrame, 562
called by displayFrameNames, 563
called by importFromFrame, 563
frameNames, 556
called by importFromFrame, 563
uses $interpreterFrameRing, 556
defun, 556
frameSpad2Cmd, 566
called by frame, 565
calls addNewInterpreterFrame, 566
calls closeInterpreterFrame, 566
calls displayFrameNames, 566
calls drop[9], 566
calls helpSpad2Cmd, 566
calls importFromFrame, 566
calls import, 566
calls last, 566
calls names, 566
calls new, 566
calls nextInterpreterFrame, 566
calls next, 566
calls object2Identifier, 566
calls previousInterpreterFrame, 566
calls qcar, 566
calls qcdr, 566
calls selectOptionLC, 566
calls throwKeyedMsg, 566
uses $options, 566
defun, 566
From, 380
called by syIgnoredFromTo, 191
defun, 380
FromTo, 380
called by syIgnoredFromTo, 191
defun, 380
function
called by dbShowCons1, 1389
called by dbShowConstructorLines, 1397
called by dewritify, 615
called by domainDescendantsOf, 1349
called by filterAndFormatConstructors, 944
called by isSystemDirectory, 1037
called by kcaPage1, 1363
called by parseNoMacroFromString, 1373
called by writify, 610
funfind, 874
defmacro, 874
funfind,LAM, 874
calls SEQ, 874
calls exit, 874
calls isFunctor, 874
calls qcar, 874
defun, 874
gatherGlossLines, 1334
defun, 1334
gbc-time
called by statisticsInitialization, 1035
genCategoryTable
called by write-categorydb, 1030
genDomainTraceName, 862
called by domainToGenvar, 861
calls genvar, 862
calls lassoc, 862
uses $domainTraceNameAssoc, 862
defun, 862
gensymInt, 616
calls charDigitVal, 616
calls error, 616
calls gensymp, 616
calls pname, 616
defun, 616
gensymp

called by dewritify,dewritifyInner, 612
gensymp
called by breaklet, 905
called by gensymInt, 616
called by isTraceGensym, 875
called by letPrint2, 887
called by letPrint3, 888
called by letPrint, 885
called by spadTrace, isTraceable, 876
called by tracelet, 904
genvar
called by genDomainTraceName, 862
called by ?t, 902
called by augmentTraceNames, 872
called by clearCmdParts, 505
called by getAliasIfTracedMapParameter, 889
called by getBpiNameIfTracedMap, 890
called by getMapSig, 853
called by getMapSubNames, 869
called by getPreviousMapSubNames, 870
called by importFromFrame, 563
called by isDomainValuedVariable, 959
called by isInterpOnlyMap, 872
called by isUncompiledMap, 871
called by putHist, 590
called by restoreHistory, 597
called by transTraceItem, 863
called by writeHistModesAndValues, 603
defun, 34
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
defun, 889
getaliaslist

called by displayProperties,sayFunctionDeps, 458
called by importFromFrame, 563
called by interpFunctionDepAlists, 465
called by isExposedConstructor, 820
called by setExposeAddGroup, 699
called by setExposeDropGroup, 702
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
defun, 958
getArgValue

called by interpret1, 52
called by displayProperties, 461
called by getI, 461
called by sayMSG, 461
defun, 461
defun, 461
defun, 461
getBpiNameIfTracedMap, 890
called by mapLetPrint, 884
called by restart, 15
called by /tracenames, 890
called by $InteractiveFrame, 890
called by $InteractiveFrame, 890

called by restart, 15

called by /tracenames, 890

called by restart, 15
INDEX

1543

defun, 890
getBrowseDatabase, 1089
calls grepConstruct, 1090
calls member, 1090
uses $includeUnexposed?, 1090
defun, 1089
getCDTEntry
called by dbShowCons1, 1389
getConstructorDocumentation
called by dbShowConsDoc1, 1393
calls getdatabase, 1394
calls lassoc, 1394
calls qcaar, 1394
calls qcdar, 1394
calls qcar, 1394
calls qcdar, 1394
defun, 1394
getConstructorExports
called by kePage, 1352
getConstructorForm
called by augmentHasArgs, 1365
called by conPageChoose, 1389
called by dbAddDocTable, 1383
called by dbShowConsDoc, 1392
called by dbShowConstructorLines, 1397
called by dbSpecialDescription, 1398
called by dbSpecialExports, 1399
called by dbSpecialOperations, 1398
called by kcdePage, 1366
called by kcuPage, 1367
called by make-databases, 1019
getConstructorModemap
called by kArgPage, 1346
getConstructorSignature
called by dbShowConsDoc1, 1393
called by evaluateType1, 917
called by reportOpsFromLisplib, 818
getdatabase, 1010
called by abbQuery, 536
called by addoperations, 1007
called by dbAddDocTable, 1383
called by dbMkEvalable, 1372
called by dbSearchOrder, 1356
called by dbShowCons1, 1389
called by dbShowConsDoc1, 1393
called by describeSpad2Cmd, 529
called by displayOperationsFromLisplib, 820
called by displayValue, 459
called by domainToGenvar, 861
called by evaluateType1, 918
called by getConstructorDocumentation, 1394
called by hasAtt, 424
called by initial-getdatabase, 1001
called by kCheckArgumentNumbers, 1373
called by kDomainName, 1370
called by kdPageInfo, 1345
called by libConstructorSig, 1406
called by loadLibNoUpdate, 1037
called by loadLib, 1035
called by localnrlib, 1015
called by mkEvalable, 913
called by originsInOrder, 1382
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
called by setExposexAddConstr, 700
called by setExposexDropConstr, 703
called by showdatabase, 1008
called by whatConstructors, 945
calls warn, 1010
uses *browse-stream*, 1010
uses *category-stream*, 1010
uses *defaultdomain-list*, 1010
uses *hasCategory-hash*, 1010
uses *hascategory-hash*, 1010
uses *interp-stream*, 1010
uses *miss*, 1010
uses *operation-hash*, 1010
uses *operation-stream*, 1010
uses $spadroot, 1010
defun, 1010
called by getDependentsOfConstructor, 1367
called by kcdePage, 1366
called by pathname, 1367
called by readLibPathFast, 1367
called by rread, 1367
called by rshut, 1367
defun, 1367
called by getDirectoryList, 984
uses $UserLevel, 984
uses $current-directory, 984
uses $directory-list, 984
defun, 984
defEnv
    calledby DaaseName, 1023
calledby initial-getdatabase, 1001
calledby make-databases, 1019
calledby restart0, 17
defEnv
    calledby getenv, 29
defEnv
    calledby copyright, 522
calledby initial-getroot, 33
calledby summary, 830
calls getenv, 29
defun, 29
defErFromDbL
    calledby getMsgInfoFromKey, 366
defFirstWord, 469
calledby doSystemCommand, 446
calls stringSpaces, 469
calls subseq, 469
defun, 469
defFlag
    calledby interpFunctionDepAlists, 465
defI
    calledby displayProperties, 461
calledby fetchOutput, 600
calledby getAndSay, 461
defImports
    calledby kcnPage, 1369
defInfovec
    calledby hasAtt, 424
defInterpMacroNames
    calledby clearCmdParts, 505
calledby displayMacros, 538
calledby displayProperties, 461
calledby displayWorkspaceNames, 454
defPropList
    calledby displayProperties, 461
defKeyedMsg, 329
calledby msgText, 60
calledby sayKeyedMsgLocal, 330
calls fetchKeyedMsg, 329
defun, 329
defl, 1050
calledby dbShowConsDoc1, 1393
calledby dbSpecialExports, 1399
calledby dbSpecialOperations, 1398
calledby reportOpsFromUnitDirectly, 822
defun, 1050
defLinePos, 372
calledby makeMsgFromLine, 371
defun, 372
defLineText, 372
calledby makeMsgFromLine, 371
defun, 372
defMapSig, 853
calledby saveMapSig, 853
calls boot-equal, 853
calls get, 853
calls unionq, 853
calls union, 853
calledby /tracenames, 853
calledby $InteractiveFrame, 853
defun, 853
defMapSubNames, 869
calledby traceSpad2Cmd, 847
calledby getPreviousMapSubNames, 869
calledby get, 869
calls unionq, 869
calls union, 869
calledby /tracenames, 869
calledby $InteractiveFrame, 869
calledby $lastUntraced, 869
defun, 869
defMsgArgL, 349
calledby getMsgInfoFromKey, 366
calledby sameMsg?, 376
defun, 349
defMsgCatAttr, 358
calledby getMsgToWhere, 363
calledby initToWhere, 368
calledby msgImPr?, 358
calledby msgNoRep?, 375
calls ifcdr, 358
calls ncAlist, 358
calls qassq, 358
defun, 358
defMsgFTTag
    usedby posPointers, 378
defMsgFTTag?, 359
calledby getMsgPos2, 378
calledby getMsgPos, 359
calledby processChPosesForOneLine, 376
INDEX

called by putFTText, 379

calls getMsgPosTagOb, 359
calls ifcar, 359
defun, 359

getMsgInfoFromKey, 366
called by putDatabaseStuff, 365
calls getErFromDbL, 366
calls getMsgArgL, 366
calls getMsgKey?, 366
calls getMsgKey, 366
calls removeAttributes, 366
calls segmentKeyedMsg, 366
calls substituteSegmentedMsg, 366
uses $msgDatabaseName, 366
defun, 366

getMsgKey, 348
called by getMsgInfoFromKey, 366
called by processKeyedError, 353
called by sameMsg?, 376
defun, 348

getMsgKey?, 362
called by getMsgInfoFromKey, 366
called by getMsgLitSym, 362
called by getStFromMsg, 354
calls identp, 362
defun, 362

getMsgLitSym, 362
called by getStFromMsg, 354
calls getMsgKey?, 362
defun, 362

getMsgPos, 359
called by erMsgCompare, 370
called by erMsgSep, 370
called by getPosStL, 356
called by posPointers, 378
called by processChPosesForOneLine, 376
called by processMsgList, 369
called by putFTText, 379
calls getMsgFTTag?, 359
calls getMsgPosTagOb, 359
defun, 359

getMsgPos2, 378
called by posPointers, 378
called by putFTText, 379
calls getMsgFTTag?, 378
calls getMsgPosTagOb, 378
calls ncBug, 379
defun, 378

getMsgPosTagOb, 348
called by getMsgFTTag?, 359
called by getMsgPos2, 378
called by getMsgPos, 359
defun, 348

getMsgPrefix, 349
called by processChPosesForOneLine, 376
defun, 349

getMsgPrefix?, 350
called by getStFromMsg, 354
called by processKeyedMsg, 354
called by initImPr, 367
called by leader?, 351
called by line?, 351
calls ncTag, 350
defun, 350

getMsgTag, 350
called by getMsgTag?, 350
called by getStFromMsg, 354
called by processKeyedError, 353
called by tabbing, 362
defun, 350

getMsgTag?, 350
called by processKeyedError, 353
calls IFCAR, 350
calls getMsgTag, 350
defun, 350

getMsgText, 349
called by getStFromMsg, 354
called by putFTText, 379
defun, 349

getMsgToWhere, 363
called by toFile?, 363
called by toScreen?, 351
calls getMsgCatAttr, 363
defun, 363

getOperationAlistFromLisplib

called by hasSig, 423
called by spadTrace, 876

getoplistForConstructorForm, 824
called by reportOpsFromUnitDirectly, 822
defun, 824

getoplistWithUniqueSignatures, 825
defun, 825

ggetOption, 892
calledby spadTrace, 876
calledby spadUntrace, 895
calledby traceDomainConstructor, 880
calls assoc, 892
defun, 892
getParserMacroNames, 453
calledby clearCmdParts, 505
calledby displayMacros, 538
calledby displayProperties, 461
calledby displayWorkspaceNames, 454
uses $pfMacros, 453
defun, 453
getPosStL, 356
calledby getStFromMsg, 354
calls decideHowMuch, 356
calls getMsgPos, 356
calls listDecideHowMuch, 356
calls ppos, 356
calls remFile, 356
calls remLine, 356
calls showMsgPos?, 356
uses $lastPos, 356
defun, 356
getPreStL, 355
calledby getStFromMsg, 354
calls size, 355
uses $preLength, 355
defun, 355
getPreviousMapSubNames, 870
calledby getMapSubNames, 869
calledby untraceMapSubNames, 873
calls exit, 870
calls get, 870
calls seq, 870
defun, 870
getProplist, 964
calledby addBinding, 963
calledby getProplist, 964
calls getProplist, 964
calls search, 964
uses $categoryFrame, 964
defun, 964
defun, 1056
defun, 1056
defun, 1057
getRefvU16, 1056
defun, 1055
defun, 1055
getStFromMsg, 354
calledby msgOutputter, 353
calls getMsgKey?, 354
calls getMsgLitSym, 354
calls getMsgPrefix?, 354
calls getMsgTag, 354
calls getMsgText, 354
calls getPosStL, 354
calls getPreStL, 354
calls pname, 354
calls tabbing, 354
defun, 354
getSystemCommandLine, 833
calledby synonymSpad2Cmd, 832
calls strpos, 833
calls substring, 833
uses $currentLine, 834
defun, 833
defun, 854
calledby getTraceOptions, 852
calledby trace1, 848
calls concat, 854
calls exit, 854
calls getTraceOption,hn, 854
calls idemp, 854
calls isListOfIdentifiersOrStrings, 854
calls isListOfIdentifiers, 854
calls object2String, 854
calls qcar, 854
calls qcdr, 854
calls seq, 854
calls seq, 854
calls stackTraceOptionError, 854
calls throwKeyedMsg, 854
calls transOnlyOption, 854
uses $traceOptionList, 854
defun, 854
defun, 853
calledby getTraceOption, 854
calls domainToGenvar, 854
calls exit, 853
calls isDomainOrPackage, 854
calls seq, 853
calls stackTraceOptionError, 854
defun, 853
getTraceOptions, 852
called by trace1, 848
calls exit, 852
calls getTraceOption, 852
calls poundsign, 852
calls seq, 852
calls throwKeyedMsg, 852
calls throwListOfKeyedMsgs, 852
uses $traceErrorStack, 852
defun, 852
getUsersOfConstructor, 1368
called by kcuPage, 1367
calls pathname, 1368
calls readLibPathFast, 1368
calls read, 1368
calls rshut, 1368
defun, 1368
getValue
called by evaluateType, 916
called by getAndEvalConstructorArgument, 958
called by interpret1, 52
getWorkspaceNames, 455
called by displayProperties, 461
called by displayWorkspaceNames, 454
calls exit, 455
calls umsort, 455
calls seq, 455
uses $InteractiveFrame, 455
defun, 455
glesseqp
called by dbShowConstructorLines, 1397
grepConstruct
called by getBrowseDatabase, 1090
handleNoParseCommands, 445
called by doSystemCommand, 446
calls concat, 470
calls member, 470
calls npboot, 470
calls npboot, 470
calls npboot, 470
calls npboot, 470
calls sayKeyedMsg, 470
calls stripLisp, 470
calls stripSpaces, 470
defun, 445
handleParsedSystemCommands, 468
called by doSystemCommand, 446
calls dumbTokenize, 468
calls parseSystemCmd, 468
calls systemCommand, 468
calls tokTran, 468
defun, 468
handleTokenSizeSystemCommands, 447
called by doSystemCommand, 446
calls dumbTokenize, 447
calls systemCommand, 447
calls tokTran, 447
defun, 447
hasAtt, 424
called by hasAttSig, 426
called by hasCaty, 420
calls constructSubst, 424
calls getInfvec, 424
calls getdatabase, 424
calls hasCatExpression, 424
calls subCopy, 424
calls unifyStruct, 424
local def $domPvar, 424
defun, 424
hasAttSig, 426
called by hasCaty, 420
called by hasAtt, 426
called by hasSig, 426
calls keyedSystemError, 426
defun, 426
hasCat
called by hasCaty, 420
hasCate, 433
called by hasCatExpression, 427
called by hasCate1, 427
called by hasCateSpecial, 435
called by hasCaty1, 432
called by hasSigAnd, 425
called by hasSigOr, 426
called by hasSig, 423
calls containsVariables, 433
calls hasCate1, 433
calls hasCateSpecial, 433
calls hasCaty, 433
calls isPatternVar, 433
calls subCopy, 433
local def $hope, 433
local ref $EmptyMode, 433
local ref $Subst, 433
defun, 433
hasCat1, 427
calledby hasCat, 433
calls hasCat, 427
local def $domPvar, 427
defun, 427
HasCategory
calledby basicLookup, 1076
hascategory-hash
usedby lefts, 1406
hasCateSpecial, 434
calledby hasCat, 433
calls augmentSub, 435
calls canCoerceFrom, 435
calls containsVars, 435
calls eqcar, 435
calls hasCateSpecialNew, 435
calls hasCat, 435
calls hasCaty, 435
calls isSubDomain, 435
local ref $Integer, 435
local ref $QuotientField, 435
defun, 434
hasCateSpecialNew, 436
calledby hasCateSpecial, 435
calls augmentSub, 436
calls defaultTargetFE, 436
calls eqcar, 436
calls hasCaty, 436
calls isEqualOrSubDomain, 436
calls member, 436
calls underDomainOf, 436
local ref $ComplexInteger, 436
local ref $Integer, 436
local ref $RationalNumber, 436
defun, 436
hasCatExpression, 427
calledby hasAtt, 424
calledby hasCatExpression, 427
calls hasCatExpression, 427
calls hasCat, 427
calls keyedSystemError, 427
defun, 427
hasCat, 420
calledby hasCateSpecialNew, 436
calledby hasCat, 433
calledby ofCategory, 419
calls augmentSub, 420
calls constructSubst, 420
calls domArg2, 420
calls domArg, 420
calls hasAttSig, 420
calls hasAtt, 420
calls hasCaty1, 421
calls hasCat, 420
calls hasSig, 420
calls kdr, 420
calls mkDomPvar, 420
calls opOf, 420
calls subCopy, 420
calls unifyStruct, 420
local ref $domPvar, 421
defun, 420
hasCate1, 432
calledby hasCaty1, 432
calledby hasCat, 421
calls hasCat, 432
calls hasCaty1, 432
calls keyedSystemError, 432
local def $domPvar, 432
defun, 432
hashable, 1075
calls Boolean, 1075
calls bpiname, 1075
calls compiledLookup, 1075
calls knownEqualPred, 1075
defun, 1075
hashCode?
calledby basicLookupCheckDefaults, 1078
calledby basicLookup, 1076
hashString
calledby basicLookupCheckDefaults, 1078
calledby basicLookup, 1076
hashtable-class
calledby writify, writifyInner, 607
hashType
called by basicLookupCheckDefaults, 1078
called by basicLookup, 1076

hasIndent
called by mkDomTypeForm, 1348

hasNewInfoAlist
called by kcPage, 1358

hasOptArgs?, 309
called by pfApplication2Sex, 306
defun, 309

hasOption, 451
called by trace1, 848
calls pname, 451
calls stringPrefix?, 451
defun, 451

hasPair, 891
called by hasPair, 891
called by letPrint2, 887
called by letPrint3, 888
called by letPrint, 885
calls hasPair, 891
calls qcar, 891
calls qcdr, 891
defun, 891

hasSig, 423
called by hasAttSig, 426
called by hasCate, 420
calls assq, 423
calls constructSubst, 423
calls constructor?, 423
calls getOperationAlistFromLisplib, 423
calls hasCate, 423
calls hasSigAnd, 423
calls hasSigOr, 423
calls keyedSystemError, 423
calls subCopy, 423
calls unifyStruct, 423
calls $domPvar, 423
defun, 423

hasSigAnd, 425
called by hasSigOr, 426
called by hasSig, 423
calls hasCate, 425
calls keyedSystemError, 425
calls subCopy, 425
defun, 425

hasSigOr, 426
called by hasSig, 423
calls hasCate, 426
calls hasSigAnd, 426
calls keyedSystemError, 426
defun, 426

help, 572
called by helpSpad2Cmd, 572
defun, 572

called by helpSpad2Cmd, 572
called by abbreviationsSpad2Cmd, 483
called by frameSpad2Cmd, 566
called by help, 572
called by savesystem, 650
called by showSpad2Cmd, 814
called by systemCommand, 448
called by newHelpSpad2Cmd, 572
calls sayKeyedMsg, 572
defun, 572

hget, 1044
called by ScanOrPairVec, ScanOrInner, 615
called by dbAddDocTable, 1383
called by dbDocTable, 1382
called by dbGetDocTable, 1385
called by dewritify, dewritifyInner, 612
called by kcPage, 1358
called by keyword?, 121
called by keyword, 121
called by saveDependentsHashTable, 1022
called by saveUsersHashTable, 1022
called by writify, writifyInner, 607
defmacro, 1044

histFileErase, 617
called by disableHist, 603
called by historySpad2Cmd, 582
called by initHist, 581
called by restoreHistory, 597
called by saveHistory, 595
called by setHistoryCore, 585
called by writeInputLines, 587
defun, 617

histFileName, 580
called by addNewInterpreterFrame, 561
called by clearCmdAll, 503
called by disableHist, 603
called by historySpad2Cmd, 582
called by initHist, 581
called by readHiFi, 601
called by restoreHistory, 597
called by saveHistory, 595
called by setHistoryCore, 585
called by writeHiFi, 602

calls makeHistFileName, 580
uses $interpreterFrameName, 580
defun, 580

histInputFileName, 580
called by saveHistory, 595
called by writeInputLines, 587

calls makePathname, 580
uses $historyDirectory, 580
uses $interpreterFrameName, 580
defun, 580

history, 582
calls historySpad2Cmd, 582
calls sayKeyedMsg, 582
uses $options, 582
defun, 582

hkeys, 1044
called by lefts, 1406
called by saveDependentsHashTable, 1022
called by saveUsersHashTable, 1022
called by scanDictCons, 137
called by scanPunCons, 139
called by writify,writifyInner, 607
defun, 1044

hput, 1044
called by ScanOrPairVec,ScanOrInner, 615
called by addBinding, 963
called by dbAddDocTable, 1383
called by dewritify,dewritifyInner, 612
called by writify,writifyInner, 607
defun, 1044

htAddHeading, 1262
defun, 1262

htAllOrNum, 1323
defun, 1323

htBeLinks, 1270
defun, 1270

htBeLispLinks, 1271
defun, 1271

htBeRadioButtons, 1274
defun, 1274

htBeginMenu
called by kcPage, 1358

htBeginTable
called by bcUnixTable, 1397

htBigSkip
called by kePage, 1352

htCacheAddChoice, 1320
defun, 1320

htCacheOne, 1323
defun, 1323

htCacheSet, 1321
defun, 1321

htCheck, 1311
defun, 1311

htCheckList, 1312
defun, 1312
htCopyProplist
called by dbShowCons1, 1389
called by dbShowCons, 1388
called by kcPage, 1358
called by kcpPage, 1364
called by kcPage, 1361
called by kePage, 1352
called by ksPage, 1351
called by ksPage, 1355
htDoneButton, 1289
defun, 1289
htDoNothing, 1311
defun, 1311
htEndMenu
called by kcPage, 1358
htEndTable
called by bcUnixTable, 1397
htEscapeString, 1297
defun, 1297
htFunctionSetLiteral, 1308
defun, 1308
htGloss, 1332
defun, 1332
htGlossPage, 1332
defun, 1332
htGlossSearch, 1336
defun, 1336
htGreekSearch, 1336
defun, 1336
htInitPage, 1262
called by bcComplexLimit, 1223
called by bcDefiniteIntegrate, 1190
called by bcDifferentiate, 1195
called by bcDraw2DSolve, 1202
called by bcDraw2Dfun, 1198
called by bcDraw2Dpar, 1200
called by bcDraw3Dfun, 1204
called by bcDraw3Dpar1, 1208
called by bcDraw3Dpar, 1206
called by bcDraw, 1197
called by bcGen, 1244, 1245
called by bcIndefiniteIntegrate, 1189
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1186
called by bcInputMatrixByFormula, 1183
called by bcInputSolveInfo, 1230
called by bcLaurentSeries, 1215
called by bcLimit, 1220
called by bcLinearSolveEqns, 1228
called by bcLinearSolveMatrix1, 1237
called by bcLinearSolveMatrixinhomo, 1238
called by bcLinearSolve, 1227
called by bcNotReady, 1247
called by bcProduct, 1194
called by bcPuiusexSeries, 1217
called by bcReadMatrix, 1182
called by bcRealLimitGen, 1222
called by bcRealLimit, 1221
called by bcSeriesByFormula, 1212
called by bcSeriesExpansion, 1211
called by bcSeries, 1210
called by bcSolveEquationsNumerically, 1235
called by bcSolve, 1226
called by bcSum, 1192
called by bcSystemSolve, 1228
called by bcTaylorSeries, 1213
called by conOpPage1, 1375
called by dbSpecialDescription, 1398
called by dbSpecialExports, 1399
called by dbSpecialOperations, 1398
called by kcPage, 1357
called by kccPage, 1364
called by kcpPage, 1361
called by kePage, 1352
called by kiPage, 1351
defun, 1262
htInputStrings, 1276
defun, 1276
htKill, 1310
defun, 1310
htLispLinks, 1269
defun, 1269
htLispMemoLinks, 1270
defun, 1270
htMakeButton, 1285
defun, 1285
htMakeDoitButton, 1288
defun, 1288
INDEX

htMakeDoneButton, 1284
called by bcDifferentiate, 1195
called by bcDraw2DSolve, 1202
called by bcDraw3Dpar1, 1208
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1186
called by bcInputMatrixByFormula, 1183
called by bcLinearSolveEqns, 1228
called by bcSeriesExpansion, 1211
called by bcSolveEquationsNumerically, 1235
called by bcSystemSolve, 1228
defun, 1284

htMakeErrorPage, 1265
defun, 1265

htMakeInputList, 1296
defun, 1296
defun, 1321

htMakePage, 1263
called by bcComplexLimit, 1223
called by bcDefiniteIntegrate, 1190
called by bcDifferentiate, 1195
called by bcDraw2DSolve, 1202
called by bcDraw2Dfun, 1198
called by bcDraw2Dpar, 1200
called by bcDraw3Dfun, 1204
called by bcDraw3Dpar1, 1208
called by bcIndefiniteIntegrate, 1189
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1186
called by bcInputMatrixByFormula, 1183
called by bcInputSolveInfo, 1230
called by bcLaurentSeries, 1215
called by bcLimit, 1220
called by bcLinearSolveEqns, 1228
called by bcLinearSolveMatrix1, 1237
called by bcLinearSolveMatrixInhom, 1238
called by bcLinearSolve, 1227
called by bcNotReady, 1247
called by bcProduct, 1194
called by bcPuiseuxSeries, 1217
called by bcReadMatrix, 1182
called by bcRealLimitGen, 1222
called by bcRealLimit, 1221
called by bcSeriesByFormula, 1212
called by bcSeriesExpansion, 1211
called by bcSeries, 1210
called by bcSolveEquationsNumerically, 1235
called by bcSolve, 1226
called by bcSum, 1193
called by bcSystemSolve, 1228
called by bcTaylorSeries, 1213
called by bcUnixTable, 1397
called by kePage, 1358
called by kdPageInfo, 1345
called by kePageDisplay, 1354
called by kePage, 1352
defun, 1263

htMakePage1, 1264
defun, 1264

htMakePathKey, 1317
defun, 1317

htMakePathKey, fn, 1316
defun, 1316

htMakeTemplates, 1283
defun, 1283

htMakeTemplates, substLabel, 1282
defun, 1282

htMarkTree, 1317
defun, 1317

htMkName, 1248
calls strconc, 1248
calls stringimage, 1248
defun, 1248

htAddInputAreaProp, 1253
defun, 1253

htAddToPageDescription, 1260
defun, 1260

htButtonDownValue, 1252
called by bcComplexLimitGen, 1225
called by bcDefiniteIntegrateGen, 1192
called by bcRealLimitGen, 1222
called by bcSolveEquations, 1236
defun, 1252

htDestroyPage, 1250
defun, 1250

htDomainConditions, 1250
defun, 1250

htDomainPvarSubstList, 1251
defun, 1251
INDEX

htpDomainVariableAlist, 1251  
defun, 1251
htpInputAreaAlist, 1253  
called by bcGenExplicitMatrix, 1187  
called by bcLinearExtractMatrix, 1238  
called by bcLinearMatrixGen, 1240  
called by bcLinearSolveEqnsGen, 1242  
defun, 1253
htpInputAreaList  
called by bcInputSolveInfo, 1230
htpLabelDefault, 1258  
defun, 1258
htpLabelErrorMsg, 1257  
defun, 1257
htpLabelFilter, 1259  
defun, 1259
htpLabelFilteredInputString, 1255  
defun, 1255
htpLabelInputString, 1254  
called by bcInputExplicitMatrix, 1185  
called by bcInputMatrixByFormula, 1183  
defun, 1256
htpLabelType, 1258  
defun, 1258
htpName, 1250  
defun, 1250
htPageDescription, 1259  
defun, 1259
htProperty, 1254  
called by bcGenExplicitMatrix, 1187  
called by bcInputMatrixByFormulaGen, 1185  
called by bcLinearSolveMatrixInhomo, 1238  
called by bcMatrixGen, 1188  
called by bcRealLimitGen1, 1223  
called by bcSolveEquations, 1236  
called by conOpPage, 1375  
called by dbCompositeWithMap, 1377  
called by dbConsHeading, 1395  
called by dbSelectCon, 1394  
called by dbShowConditions, 1394  
called by dbShowCons1, 1389  
called by dbShowConsDoc1, 1393  
called by dbShowConsDoc, 1392  
called by dbShowConsKindsFilter, 1392  
called by dbShowCons, 1387  
called by kArgPage, 1346  
called by kcPage, 1357  
called by kcaPage1, 1363  
called by kccPage, 1364  
called by kcdePage, 1366  
called by kcnPage, 1368  
called by kcpPage, 1361  
called by kcuPage, 1367  
called by kePage, 1351  
called by kiPage, 1350  
called by koPageAux1, 1379  
called by koPageFromKKPage, 1378  
called by koPageInputAreaUnchanged?, 1370  
called by koaPageFilterByName, 1380  
defun, 1254
htpLabelSpadType, 1258  
defun, 1258
htpLabelSpadValue, 1256  
called by bcInputEquations, 1231  
called by bcInputExplicitMatrix, 1186
INDEX

called by `dbShowConditions`, 1395
defun, 1298

htSaySaturn
called by `bcUnixTable`, 1397
called by `dbShowConditions`, 1395
called by `dbShowConsDoc1`, 1393
called by `kePageDisplay`, 1354
defun, 1300

htSayStandard
called by `dbShowCons1`, 1389
called by `kcPage`, 1358
called by `kdPageInfo`, 1345
called by `kePageDisplay`, 1354
called by `kePage`, 1352
called by `ksPage`, 1355
defun, 1306

htSetCache, 1319
defun, 1319
defun, 1300

htSetExposé, 1318
defun, 1318
defun, 1306
defun, 1304
defun, 1303
defun, 1303
defun, 1310
defun, 1307
defun, 1304
defun, 1301
defun, 1301
defun, 1301
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298

defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
defun, 1298
called by `bcSystemSolve`, 1229
called by `bcTaylorSeries`, 1213
called by `dbSpecialDescription`, 1398
called by `dbSpecialExports`, 1399
called by `kcPage`, 1358
called by `kiPage`, 1351
defun, 1263

`htShowPageNoScroll`, 1263
called by `dbShowCons1`, 1390
defun, 1263

`htShowSetPage`, 1302
defun, 1302

`htShowSetTree`, 1298
defun, 1298

`htShowSetTreeValue`, 1301
defun, 1301

`htSowPage`
called by `kePage`, 1352

`htStringPad`, 1248
calls `strconc`, 1248
calls `stringimage`, 1248
defun, 1248

`htsv`, 1298
defun, 1298

`htSystemVariables`, 1329
defun, 1329

`htSystemVariables`, `displayOptions`, 1326
defun, 1326

`htSystemVariables`, `fn`, 1326
defun, 1326

`htSystemVariables`, `functionTail`, 1328
defun, 1328

`htSystemVariables`, `gn`, 1325
defun, 1325

`htTextSearch`, 1338
defun, 1338

`htTutorialSearch`, 1340

defun, 1340

`id`
called by `inclmsgConActive`, 92
called by `inclmsgConStill`, 92
called by `inclmsgFileCycle`, 91
called by `inclmsgIfSyntax`, 95
called by `inclmsgSay`, 88

`idChar?`, 128
called by `scanW`, 128
defmacro, 128

`identp`, 1046
called by `getMsgKey?`, 362
called by `getTraceOption`, 854
called by `isDomainValuedVariable`, 959
called by `isListOfIdentifiersOrStrings`, 869
called by `isListOfIdentifiers`, 868
called by `isSharpVar`, 886
called by `isgenvar`, 887
called by `messageprint-1`, 1049
called by `ncAlist`, 415
called by `ncTag`, 415
called by `ofCategory`, 419
called by `restoreHistory`, 597
called by `rwrite`, 605
called by `selectOption`, 479
called by `undo`, 922
defmacro, 1046

`if`, 395

syntax, 395

`If?`, 77
called by `include1`, 80
calls `quotient`, 77
defvar, 77

`IFCAR`
called by `getMsgTag?`, 350
called by `posPointers`, 378
called by `remLine`, 357

`ifcar`
called by `conOpPage1`, 1375
called by `dbShowCons`, 1387
called by `defaultTargetFE`, 437
called by `fillerSpaces`, 18
called by `getMsgFTTag?`, 359
called by `pfAbSynOp`, 412
called by `pfExpression`, 264
called by `pfLeaf`, 247
called by `pfSymbol`, 251
called by `pfSymb`, 251
called by `tokConstruct`, 411

`IFCDR`
called by `remFile`, 357

`ifcdr`
called by `clearParserMacro`, 453
called by `conOpPage1`, 1375
INDEX

1557

called by domainDescendantsOf, 1349
called by getMsgCatAttr, 358
called by kePage, 1352
called by mac0Get, 228
called by setMsgCatlessAttr, 365
called by specialChar, 980

ifCond, 87
called by incLude1, 80
calls MakeSymbol, 87
calls incCommandTail, 87
uses $incAssertions, 87
defun, 87

IfKeepPart, 76
defvar, 76

IfSkipPart, 76
defvar, 76

IfSkipToEnd, 75
defvar, 75

iht, 1260
defun, 1260

images
Restart, 14

import
called by frameSpad2Cmd, 566
importFromFile, 563
called by frameSpad2Cmd, 566
called by importFromFile, 563
calls boot-equal, 563
calls clearCmdParts, 563
calls exit, 563
calls frameEnvironment, 563
calls frameNames, 563
calls frameName, 563
calls getalist, 563
calls get, 563
calls importFromFile, 563
calls member, 563
calls putHist, 563
calls queryUserKeyedMsg, 563
calls sayKeyedMsg, 563
calls seq, 563
calls string2id-n, 563
calls throwKeyedMsg, 563
calls upcase, 563
uses $interpreterFrameRing, 563
defun, 563

in-stream, 961
used by ncTopLevel, 23
used by serverReadLine, 43
defvar, 961

incActive?, 101
called by incLude1, 80
defun, 101

incAppend, 85
called by incAppend1, 85
called by incLude1, 80
called by next1, 37
calls Delay, 85
calls incAppend1, 85
defun, 85

incAppend1, 85
called by incAppend, 85
calls StreamNull, 85
calls incAppend, 85
defun, 85

incBiteOff, 623
called by incFileName, 622
defun, 623

incClassify, 97
called by incLude1, 80
calls incCommand?, 97
uses incCommands, 97
defun, 97

incCommand?, 98
called by incClassify, 97
calls char, 98
defun, 98

incCommands, 96
used by incClassify, 97
defun, 96

incCommandTail, 99
called by assertCond, 94
called by ifCond, 87
called by incLude1, 80
called by incFname, 100
calls incDrop, 99
defun, 99

incConsoleInput, 100
called by incLude1, 80
calls incRgen, 100
calls make-instream, 100
defun, 100
incDrop, 100
called by incCommandTail, 99
calls substring, 100
defun, 100
incFileInput, 100
called by inclFname, 100
calls incRgen, 100
calls make-instream, 100
defun, 100
incFileName, 622
called by inclFname, 100
called by ncloopIncFileName, 622
calls incBiteOff, 622
defun, 622
incHandleMessage, 74
called by incRenumberLine, 74
calls ncBug, 74
calls ncSoftError, 74
defun, 74
inclgen, 73
called by inclgen1, 73
called by incRenumber, 72
calls Delay, 73
calls inclgen1, 73
defun, 73
inclgen1, 73
called by inclgen, 73
calls inclgen, 73
defun, 73
inclfname, 100
called by inclFname, 100
called by incCommandTail, 100
calls incFileName, 100
defun, 100
incLine, 86
called by ximsg, 84
called by xilskip, 87
calls incLine1, 86
defun, 86
incLine1, 86
called by incLine, 86
called by xilOK1, 85
calls incCreate, 86
defun, 86
inclmsgCannotRead, 90
called by xilCannotRead, 90
calls thefname, 90
defun, 90
inclmsgCmdBug, 96
called by xilCmdBug, 96
defun, 96
inclmsgConActive, 92
called by xilConActive, 91
calls id, 92
defun, 92
inclmsgConsole, 93
called by xilConsole, 92
defun, 93
inclmsgConStill, 92
called by xilConStill, 92
calls id, 92
defun, 92
inclmsgFileCycle, 90
called by xilFileCycle, 90
calls id, 91
calls porigin, 91
defun, 90
inclmsgFinSkipped, 93
called by xilSkippingFin, 93
defun, 93
inclmsgIfBug, 96
called by xilIfBug, 95
defun, 96
inclmsgIfSyntax, 95
called by xilIfSyntax, 95
calls concat, 95
calls id, 95
calls origin, 95
defun, 95
inclmsgNoSuchFile, 89
called by xilNoSuchFile, 89
calls thefname, 89
defun, 89
inclmsgPrematureEOF, 86
called by xilPrematureEOF, 84
calls origin, 86
defun, 86
inclmsgPrematureFin, 94
called by xilPrematureFin, 93
calls origin, 94
defun, 94
inclmsgSay, 88
INDEX

called by xlSay, 88
calls id, 88
defun, 88

incLude, 75
called by incLude1, 80
called by incStream, 71
called by incString, 37
calls Delay, 75
defun, 75

include
calls incLude1, 75

include help page, 621
manpage, 621

incLude1, 79
called by include, 75
calls Elseif?, 80
calls If?, 80
calls KeepPart?, 80
calls Rest, 80
calls SkipEnd?, 80
calls SkipPart?, 80
calls Skipping?, 80
calls StreamNull, 79
calls Top?, 79
calls assertCond, 80
calls concat, 80
calls expand-tabs, 80
calls ifCond, 80
calls incActive?, 80
calls incAppend, 80
calls incClassify, 80
calls incCommandTail, 80
calls incConsoleInput, 80
calls incFileInput, 80
calls incLude, 80
calls incNConsoles, 80
calls incFname, 80
calls xlCannotRead, 80
calls xICmdBug, 80
calls xlConActive, 80
calls xlConStill, 80
calls xlConsole, 80
calls xIFileCycle, 80
calls xIFfBug, 80
calls xIFfSyntax, 80
calls xINoSuchFile, 80
calls xlOK1, 80
calls xlOK, 80
calls xlPrematureEOF, 79
calls xlPrematureFin, 80
calls xlSay, 80
calls xlSkippingFin, 80
calls xlSkip, 80
defun, 79

incNConsoles, 101
called by incLude1, 80
called by incNConsoles, 101
calls incNConsoles, 101
defun, 101

incPrefix?, 99
called by lineoftoks, 111
called by scanIgnoreLine, 113
defun, 99

incRenumber, 72
called by incStream, 71
called by incString, 37
calls incIgen, 72
calls incZip, 72
defun, 72

incRenumberItem, 74
called by incRenumberLine, 73
calls lnSetGlobalNum, 74
defun, 74

incRenumberLine, 73
calls incHandleMessage, 74
calls incRenumberItem, 73
defun, 73

incRgen, 101
called by incConsoleInput, 100
called by incFileInput, 100
called by incRgen1, 102
called by incStream, 71
calls Delay, 101
calls incRgen1, 101
defun, 101

incRgen1, 102
called by incRgen, 101
calls incRgen, 102
uses StreamNil, 102
defun, 102

incStream, 71
called by intloopInclude0, 61
called by ncloopInclude0, 71
calls incLude, 71
calls incRnumber, 71
calls incRgen, 71
uses Top, 72
defun, 71

incString, 37
called by intloopProcessString, 36
called by parseFromString, 47
called by parseNoMacroFromString, 1374
calls incLude, 37
calls incRnumber, 37
uses Top, 37
defun, 37

incZip, 72
called by incRnumber, 72
called by incZip1, 72
calls Delay, 72
calls incZip1, 72
defun, 72

incZip1, 72
called by incZip, 72
calls StreamNull, 72
calls incZip, 72
defun, 72

infgeneric, 110
defvar, 110

init-boot/spad-reader, 968
calls icclear, 968
calls next-lines-clear, 968
uses *standard-output*, 968
uses $spad-errors, 968
uses boot-line-stack, 968
uses file-closed, 968
uses line-handler, 968
uses meta-error-handler, 968
uses spaderrorstream, 968
uses xtokenreader, 968
defun, 968

init-memory-config, 32
called by restart, 15
calls allocate-contiguous-pages, 33
calls allocate-relocatable-pages, 33
calls allocate, 33
calls set-hole-size, 33
defun, 32

initHist, 581
called by restart, 15
calls $replace, 581
calls histFileErase, 581
calls histFileName, 581
calls initHistList, 581
calls makeInputFilename, 581
calls oldHistFileName, 581
uses $HiFiAccess, 581
uses $useInternalHistoryTable, 581
defun, 581

initial-getdatabase, 1001
called by resethashables, 1000
calls getEnv, 1001
calls getdatabase, 1001
defun, 1001

initializeInterpreterFrameRing, 555
called by restart, 15
calls emptyInterpreterFrame, 555
calls updateFromCurrentInterpreterFrame, 555
uses $interpreterFrameName, 555
uses $interpreterFrameRing, 555
defun, 555

initializeSetVariables, 653
called by initializeSetVariables, 653
called by resetWorkspaceVariables, 654
calls initializeSetVariables, 653
calls literals, 653
calls sayMSG, 653
calls translateYesNo2TrueFalse, 653
calls tree, 653
defun, 653

initializeTimedNames
called by processInteractive, 48

initIMPr, 367
called by msgCreate, 348
INDEX

calls getMsgTag, 367
calls setMsgUnforcedAttr, 367
uses $erMsgToss, 367
uses $imPrTagGuys, 367
defun, 367
initroot, 33
calledby restart, 15
calls getenvironment, 33
calls reroot, 33
uses $spadroot, 33
defun, 33
initToWhere, 368
calledby msgCreate, 348
calls getMsgCatAttr, 368
calls setMsgUnforcedAttr, 368
defun, 368
input-libraries, 668
usedby addInputLibrary, 668
usedby dropInputLibrary, 669
usedby openOutputLibrary, 666
usedby setInputLibrary, 667
defvar, 668
insert
calledby originsInOrder, 1382
calledby updateSourceFiles, 546
insertAlist
calledby kePageOpAlist, 1353
insertpile, 335
calledby intloopInclude0, 61
calledby nclloopInclude0, 71
calls npNull, 335
calls pileCforest, 335
calls pilePlusComments, 335
calls pilePlusComment, 335
calls pileTree, 335
defun, 335
insertPos, 379
calledby posPointers, 378
calls done, 379
defun, 379
installConstructor
calledby loadLibNoUpdate, 1037
calledby loadLib, 1035
calledby localrlib, 1015
integer-decode-float-denominator, 1103
defun, 1103
integer-decode-float-exponent, 1103
defun, 1103
integer-decode-float-numerator, 1102
defun, 1102
integer-decode-float-sign, 1103
defun, 1103
intern
calledby dbShowConstructorLines, 1397
intern
calledby constructSubst, 434
calledby spadTraceAlias, 891
InterpExecuteSpadSystemCommand, 30
calls ExecuteInterpSystemCommand, 30
uses $intCoerceFailure, 30
uses $intSpadReader, 30
catches, 30
defun, 30
interFunctionDepAlists, 465
calledby displayProperties, 461
calls getFlag, 465
calls getalist, 465
calls putalist, 465
uses $InteractiveFrame, 465
uses $dependeeAlist, 465
uses $dependentAlist, 465
uses $e, 465
defun, 465
interpOpen, 1003
calls DaaseName, 1004
calls make-database, 1004
uses *allconstructors*, 1004
uses *interp-stream*, 1004
uses *interp-stream-stamp*, 1004
uses $spadroot, 1004
defun, 1003
interpopen
calledby resethashables, 1000
calledby restart0, 16
interpret, 52
calledby interpretTopLevel, 51
calls interpret1, 52
uses $env, 52
uses $eval, 52
uses $genValue, 52
defun, 52
interpret1, 52
called by interpret, 52
  calls bottomUp, 52
  calls getArgValue, 52
  calls getValue, 52
  calls interpret2, 52
  calls keyedSystemError, 53
  calls mkAtreeWithSrcPos, 52
  calls objNew, 52
  calls putTarget, 52
  uses $eval, 53
  uses $genValue, 53
  defun, 52
interpret2, 53
  called by interpret1, 52
  calls coerceInteractive, 53
  calls member, 53
  calls objMode, 53
  calls objNew, 53
  calls objVal, 53
  calls systemErrorHere, 53
  calls throwKeyedMsgCannotCoerceWithValue, 53
  uses $EmptyMode, 53
  uses $ThrowAwayMode, 53
  defun, 53
interpretTopLevel, 51
  called by interpretTopLevel, 51
  called by processInteractive1, 50
  calls interpretTopLevel, 51
  calls interpret, 51
  calls peekTimedName, 51
  calls stopTimingProcess, 51
  uses $timedNameStack, 51
  catches, 51
  defun, 51
interrupt
  called by break, 906
interruptPform, 65
  called by phiInterpret, 65
  calls pf2Sex, 66
  calls processInteractive, 65
  calls zeroOneTran, 66
  defun, 65
intloop, 24
  called by ncIntLoop, 23
  calls SpadInterpretStream, 24
  calls resetStackLimits, 24
  uses $intRestart, 24
  uses $intTopLevel, 24
  catches, 24
  defun, 24
intloopEchoParse, 67
  called by intloopInclude0, 61
  calls dqToList, 67
  calls mkLineList, 67
  calls ncloopDQlines, 67
  calls ncloopPrintLines, 67
  calls npParse, 67
  calls setCurrentLine, 67
  uses $EchoLines, 67
  uses $lines, 67
  defun, 67
intloopInclude, 61
  called by SpadInterpretStream, 27
  calls ST, 61
  calls intloopInclude0, 61
  defun, 61
intloopInclude0, 61
  called by intloopInclude, 61
  calls incStream, 61
  calls insertpile, 61
  calls intloopEchoParse, 61
  calls intloopProcess, 61
  calls lineoftoks, 62
  calls next, 61
  uses $lines, 62
  defun, 61
intloopPrefix?, 34
  called by intloopReadConsole, 28
  defun, 34
intloopProcess, 62
  called by intloopInclude0, 61
  called by intloopProcessString, 36
  called by intloopProcess, 62
  calls StreamNull, 62
  calls $systemCommandFunction, 62
  calls intloopProcess, 62
  calls intloopSpadProcess, 62
  calls pfAbSynOp?, 62
  calls setCurrentLine, 62
  calls tokPart, 62
  uses $systemCommandFunction, 62
INDEX

defun, 62
intloopProcessString, 36
called by intloopReadConsole, 28
calls incString, 36
calls intloopProcess, 36
calls next, 36
calls setCurrentLine, 36
defun, 36
intloopReadConsole, 28
called by SpadInterpretStream, 27
called by intloopReadConsole, 28
calls concat, 28
calls intloopPrefix?, 28
calls intloopProcessString, 28
calls intloopReadConsole, 28
calls intnplisp, 28
calls leaveScratchpad, 28
calls mkprompt, 28
calls ncloopCommand, 28
calls ncloopEscaped, 28
calls serverReadLine, 28
calls setCurrentLine, 28
uses $dalymode, 29
defun, 28
throws, 28
intloopSpadProcess, 63
called by intloopProcess, 62
calls CatchAsCan, 63
calls Catch, 63
calls intloopSpadProcess, interp, 63
calls ncPutQ, 63
uses $NeedToSignalSessionManager, 63
uses $currentCarrier, 63
uses $intCoerceFailure, 63
uses $intSpadReader, 63
uses $ncMsgList, 63
defun, 63
is-console[9] called by serverReadLine, 43
called by shut, 982
isDomain called by ?t, 903
called by addTraceItem, 902
called by compiledLookup, 1076
isDomainOrPackage, 875
called by /traceresponse, 894
called by ?t, 903
called by addTraceItem, 902
called by getTraceOption, 854
called by prTraceNames, fn, 898
called by shortenForPrinting, 891
called by spadReply.printName, 894
called by spadTrace, 876
called by spadUntrace, 895
called by traceReply, 899
called by untraceDomainConstructor, keepTraced?, 882
called by writify, writifyInner, 607
calls isFunctor, 875
calls opOf, 875
calls poundsign, 875
calls refvecp, 875
defun, 875
isDomainValuedVariable, 959
called by evaluateType, 916
called by reportOperations, 815
calls get, 959
calls identp, 959
calls member, 959
calls objMode, 959
calls objValUnwrap, 959
called by $InteractiveFrame, 959
called by $env, 959
called by $e, 959
defun, 959
isEqualOrSubDomain, 438
called by defaultTargetFE, 437
called by hasCateSpecialNew, 436
called by isSubDomain, 438
defun, 438
isExposedConstructor, 820
called by conOpPage1, 1375
called by dbShowCons1, 1389
called by dbShowConsDoc1, 1393
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
calls getalist, 820
local ref $globalExposureGroupAlist, 820
local ref $localExposureData, 820
defun, 820
isFunctor

called by funfunf,LAM, 874
called by isDomainOrPackage, 875
called by trace1, 848
called by traceReply, 899

called by tracelet, 904
calls digitp, 887
calls identp, 887
calls size, 886
defun, 886
isIntegerString, 468
called by tokTran, 467
defun, 468
isInternalMapName

called by displayProperties, 461
isInterpMacro

called by displayMacro, 453
called by displayProperties, 462
isInterpOnlyMap, 872
called by getTraceOption, 872
called by $InteractiveFrame, 872
defun, 872
isListOfIdentifiers, 868
called by getTraceOption, 854
called by exit, 868
called by identp, 868
called by seq, 868
defun, 868
isListOfIdentifiersOrStrings, 869
called by getTraceOption, 854
called by exit, 869
called by identp, 869
called by seq, 869
defun, 869
isLocalVar

called by getAndEvalConstructorArgument, 958
isMap

called by clearCmdParts, 505
isNameOfType

called by reportOperations, 815
isNewWorldDomain

called by basicLookup, 1076
isPartialMode, 420
called by evalCategory, 959
INDEX

calls contained, 420
local ref $EmptyMode, 420
defun, 420
isPatternVar, 431
called by containsVars1, 431
called by containsVars, 430
called by hasCate, 433
called by unifyStructVar, 429
called by unifyStruct, 428
defun, 431
isSharpVar, 886
called by domArg2, 422
called by isSharpVarWithNum, 886
calls identp, 886
defun, 886
isSharpVarWithNum, 886
called by getAliasIfTracedMapParameter, 889
called by letPrint2, 887
called by letPrint3, 888
called by letPrint, 885
calls digitp, 886
calls digitp, 886
calls digits, 886
calls isSharpVar, 886
calls pnum, 886
calls qsize, 886
defun, 886
isSubDomain
called by hasCateSpecial, 435
called by isEqualOrSubDomain, 438
isSubForRedundantMapName, 873
called by traceReply, 899
calls assocleft, 873
calls member, 873
calls rassocSub, 873
uses $mapSubNameAlist, 873
defun, 873
isSystemDirectory, 1037
called by loadLib, 1035
calls function, 1037
local ref $spadroot, 1037
defun, 1037
isTraceGensym, 875
calls gensym, 875
defun, 875
isType
called by reportOperations, 816
isUncompiledMap, 871
calls get, 871
uses $InteractiveFrame, 871
defun, 871
isValidType
called by dbExtractUnderlyingDomain, 1377
isWrapped
called by getAndEvalConstructorArgument, 958
called by retract, 1064
iterate, 397
syntax, 397
justifyMyType, 60
called by printTypeAndTimeNormal, 57
calls fillSpaces, 60
uses $linelength, 60
defun, 60
kaddr
called by zsystemdevelopment1, 954
kadr
called by zsystemdevelopment1, 954
kar
called by recordFrame, 923
called by untraceDomainConstructor,keepTraced?, 882
called by zsystemdevelopment1, 954
kArgPage, 1346
calls dbShowCons, 1346
calls domainDescendantsOf, 1346
calls getConstructorModemap, 1346
calls htpProperty, 1346
calls htpSetProperty, 1346
calls mkDomTypeForm, 1346
calls position, 1346
calls sublisFormal, 1346
defun, 1346
kArgumentCheck, 1371
called by kDomainName, 1370
calls conSpecialString?, 1371
calls form2String, 1371
calls kdr, 1371
calls opOf, 1371
calls stringimage, 1371
defun, 1371
kcaPage, 1362
calls ancestorsOf, 1362
calls kcaPage1, 1362
defun, 1362
kcaPage1, 1363
calledby kcaPage, 1362
calledby kcdPage, 1362
calls augmentHasArgs, 1363
calls dbShowCons, 1363
calls errorPage, 1363
calls form2HtString, 1363
calls function, 1363
calls htpProperty, 1363
calls htpSetProperty, 1363
calls kDomainName, 1363
calls listSort, 1363
calls opOf, 1363
defun, 1363
kccPage, 1364
calls augmentHasArgs, 1364
calls childrenOf, 1364
calls dbShowCons, 1364
calls errorPage, 1364
calls form2HtString, 1364
calls htpProperty, 1364
calls htpSetProperty, 1364
calls kDomainName, 1364
calls opOf, 1364
calls qcar, 1364
calls reduceAListForDomain, 1364
defun, 1364
kcedePage, 1366
calls concat, 1366
calls dbShowCons, 1366
calls getConstructorForm, 1366
calls getDependentsOfConstructor, 1366
calls htpProperty, 1366
calls htpSetProperty, 1366
calls ncParseFromString, 1366
calls nequal, 1366
calls opOf, 1366
defun, 1366
kcdPage, 1362
calls descendantsOf, 1362
calls kcaPage1, 1362
defun, 1362
kCheckArgumentNumber
calledby kCheckArgumentNumbers, 1373
kCheckArgumentNumbers, 1373
calledby kisValidType, 1373
calls getdatabase, 1373
calls kCheckArgumentNumber, 1373
calls kdr, 1373
defun, 1373
kcnPage, 1368
calls concat, 1369
calls dbShowCons, 1369
calls errorPage, 1368
calls form2HtString, 1368
calls getImports, 1369
calls htpProperty, 1368
calls htpSetProperty, 1369
calls kDomainName, 1368
calls opOf, 1369
calls pname, 1369
calls qcar, 1368
calls sublislis, 1369
defun, 1368
kcPage, 1357
calls asharpConstructorName?, 1358
calls brCon, 1358
calls dbpHasDefaultValue?, 1358
calls errorPage, 1357
calls form2HtString, 1357
calls hasNewInfoAlist, 1358
calls hget, 1358
calls htBeginMenu, 1358
calls htCopyPropList, 1358
calls htEndMenu, 1358
calls htInitPage, 1357
calls htMakePage, 1358
calls htSayStandard, 1358
calls htSay, 1358
calls htShowPage, 1358
calls htpProperty, 1357
  calls htpSetProperty, 1358
  calls kDomainName, 1357
  calls nequal, 1358
  calls opOf, 1357
  calls qcar, 1357
calls satBreak, 1358
uses $defaultPackageNamesHT, 1358
defun, 1357

kcpPage, 1361
calls dbShowCons, 1361
calls errorPage, 1361
calls form2HtString, 1361
calls htCopyProplist, 1361
calls htInitPage, 1361
calls htpProperty, 1361
calls htpSetProperty, 1361
calls kDomainName, 1361
calls opOf, 1361
calls parentsOf, 1361
calls qcar, 1361
calls sublisis, 1361
defun, 1361

kcuPage, 1367
calls concat, 1367
calls dbShowCons, 1367
calls errorPage, 1367
calls get2HtString, 1367
calls htCopyProplist, 1367
calls htInitPage, 1367
calls htpProperty, 1367
calls htpSetProperty, 1367
calls ncParseFromString, 1367
calls nequal, 1367
calls opOf, 1367
defun, 1367

kDomainName, 1370
called by kcPage, 1357
called by kcaPage1, 1363
called by kccPage, 1364
called by kcnPage, 1368
called by kcpPage, 1361
called by kcPage, 1351
called by kipage, 1351
called by koPage, 1378
called by ksPage, 1355
calls concat, 1370
calls dbMkEvalable, 1370
calls getdatabase, 1370
calls htpLabelInputString, 1370
calls htpSetProperty, 1370
calls kArgumentCheck, 1370
calls kdr, 1370
calls kisValidate, 1370
calls mkConform, 1370
calls unabrev, 1370
uses $PatternVariableList, 1370
catches, 1370
defun, 1370

kdPageInfo, 1345
called by bcHt, 1345
called by extractFileNameFromPath, 1345
called by getdatabase, 1345
called by htMakePage, 1345
called by htSaturnBreak, 1345
called by htSayStandard, 1345
called by htSay, 1345
called by kPageArgs, 1345
called by length, 1345
called by nequal, 1345
called by stringimage, 1345
called by subseq, 1345
defun, 1345

called by augmentHasArgs, 1365
called by dbExtractUnderlyingDomain, 1377
called by dbGetDocTable.hn, 1384
called by hasCaty, 420
called by kArgumentCheck, 1371
called by kCheckArgumentNumbers, 1373
called by kDomainName, 1370
called by reportOpsFromLispLib, 818
called by searchCurrentEnv, 964
called by searchTailEnv, 965
called by set1, 809
called by spadTrace, 876

KeepPart?, 79
called by Skipping?, 79
called by incLude1, 80
called by remainder, 79
defvar, 79

kePage, 1351
called by bcConPredTable, 1352
called by capitalize, 1351
calls concat, 1351
calls errorPage, 1351
calls form2HTString, 1352
calls getConstructorExports, 1352
calls htBigSkip, 1352
calls htCopyProplist, 1352
calls htInitPage, 1352
calls htMakePage, 1352
calls htSayStandard, 1352
calls htSay, 1352
calls htSowPage, 1352
calls htpProperty, 1351
calls htpSetProperty, 1351, 1352
calls ifcdr, 1352
calls kDomainName, 1351
calls kePageDisplay, 1352
calls kePageOpAlist, 1352
calls length, 1352
calls menuButton, 1352
calls mkConform, 1351
calls opOf, 1351
calls pluralSay, 1352
calls simpHasPred, 1352
calls sublisFormal, 1352
uses $confomrsAreDomains, 1352
defun, 1351
kePageDisplay, 1354
calledby dbSpecialExports, 1399
calledby kePage, 1352
calls dbGatherData, 1354
calls dbSowOpItems, 1354
calls htMakePage, 1354
calls htSaySaturn, 1354
calls htSayStandard, 1354
calls htSay, 1354
calls htpSetProperty, 1354
calls length, 1354
calls menuButton, 1354
calls pluralize, 1354
calls stringimage, 1354
defun, 1354
kePageOpAlist, 1353
calledby kePage, 1352
calls insertAlist, 1353
calls lassoc, 1353
calls zeroOneConvert, 1353
defun, 1353
keyedSystemError
calledby compiledLookupCheck, 501
calledby hasAttSig, 426
calledby hasCatExpression, 427
calledby hasCaty1, 432
calledby hasSigAnd, 425
calledby hasSigOr, 426
calledby hasSig, 423
calledby interpret1, 53
calledby pf2Sex1, 301
calledby pfLiteral2Sex, 304
calledby readHiFi, 601
keyword, 121
calledby ifkey, 122
calledby scanCloser?, 125
calledby scanKeyTr, 120
calls hget, 121
defun, 121
keyword?, 121
calledby scanWord, 126
calls hget, 121
defun, 121
kFormatSlotDomain
calledby dbAddChainDomain, 1386
calledby dbSearchOrder, 1356
kiPage, 1350
calledby capitalize, 1351
calledby dbShowConsDoc1, 1351
calledby errorPage, 1351
calledby htCopyProplist, 1351
calledby htInitPage, 1351
calledby htShowPage, 1351
calledby htpProperty, 1350
called by kDomainName, 1351
called by mkConform, 1351
uses $conformsAreDomains, 1351
defun, 1350
kisValidType, 1372
calledby kDomainName, 1370
called by kCheckArgumentNumbers, 1373
called by member, 1373
called by processInteractive, 1372
uses $ProcessInteractiveValue, 1373
uses $noEvalTypeMsg, 1373
catches, 1373
INDEX

1569

defun, 1372

knownEqualPred
called by hashable, 1075

koaPageFilterByCategory
called by koaPageFilterByName, 1380

table
  calls dbGetInputString, 1380
calls downcase, 1380
calls htpLabelInputString, 1380
calls htpProperty, 1380
calls htpSetProperty, 1380
calls koaPageFilterByCategory, 1380
calls pmTransFilter, 1380
calls stringimage, 1380
calls superMatch?, 1380
defun, 1380

koAttrs
called by koPageAux, 1379

koOps
called by koPageAux, 1379

koPage, 1377
called by conOpPage1, 1375
calls capitalize, 1378
calls concat, 1378
calls errorPage, 1378
calls form2HtString, 1378
calls htpProperty, 1377
calls htpSetProperty, 1378
calls kDomainName, 1378
calls koPageAux, 1378
calls koPageInputAreaUnchanged?, 1378
defun, 1377

koPageAux, 1379
called by koPageFromKKPage, 1378
called by koPage, 1378
calls assoc, 1379
calls dbShowOperationsFromConform, 1379
calls htpSetProperty, 1379
calls koAttrs, 1379
calls koOps, 1379
calls systemError, 1379
defun, 1379

koPageAux1, 1379
calls dbShowOperationsFromConform, 1379
calls htpProperty, 1379
defun, 1379

koPageFromKKPage, 1378
calls htpProperty, 1378
calls koPageAux, 1378
defun, 1378

koPageInputAreaUnchanged?, 1370
called by koPage, 1378
calls concat, 1370
calls htpLabelInputString, 1370
calls htpProperty, 1370
calls stringimage, 1370
defun, 1370

kPage
called by conPage, 1343

kPageArgs
called by kdPageInfo, 1345

ksPage, 1355
called by dbSearchOrder, 1355
called by dbShowCons, 1355
called by errorPage, 1355
called by form2HtString, 1355
called by htpCopyProplist, 1355
called by htpInitPageNoScroll, 1355
called by htpSayStandard, 1355
called by htpSay, 1355
called by htpProperty, 1355
called by htpSetProperty, 1355
called by kDomainName, 1355
defun, 1355

kTestPred, 1386
called by dbSearchOrder, 1356
called by simpHasPred, 1386
called by testBitVector, 1386
uses $domain, 1386
uses $predvec, 1386
defun, 1386

called by bcMatrixGen, 1188
called by breaklet, 905
called by coerceTraceFunValue2E, 867
called by conOpPage1, 1375
called by genDomainTraceName, 862
called by getConstructorDocumentation, 1394

called by kePageOpAlist, 1353
called by letPrint2, 887
called by letPrint3, 888
  called by letPrint, 885
  called by processSynonyms, 31
  called by reportUndo, 928
  called by serverReadLine, 42
  called by set1, 809
  called by subTypes, 866
  called by trace1, 848
  called by tracelet, 904
  called by undoSingleStep, 931
  called by unifyStructVar, 429
lassocSub, 871
  called by untrace, 862
  calls lassq, 871
  defun, 871
lassq
  called by conPageFastPath, 1344
  called by conPage, 1343
  called by diffAlist, 925
  called by lassocSub, 871
last
  called by frameSpad2Cmd, 566
lastatom
  called by dbGet2Cmd, gn, 1384
lastc
  called by StringToDir, 1091
lastTokPosn, 341
  called by enPile, 340
  called by separatePiles, 341
  calls tokPosn, 341
  defun, 341
leader?, 351
  called by msgOutputter, 353
  called by showMsgPos?, 358
  calls getMsgTag, 351
  defun, 351
leave, 398
  syntax, 398
leaveScratchpad, 639
  called by intloopReadConsole, 28
  called by quitSpad2Cmd, 638
  defun, 639
lefts, 1406
  calls hkeys, 1406
  uses hascategory-hash, 1406
  defun, 1406
length
  called by augmentHasArgs, 1365
  called by bcDifferentiateGen, 1196
  called by bcInputExplicitMatrix, 1185
  called by conPageFastPath, 1344
  called by dbConsHeading, 1395
  called by dbConstructorDoc, hm, 1380
  called by dbShowConditions, 1395
  called by kdPageInfo, 1345
  called by kePageDisplay, 1354
  called by kePage, 1352
  called by libConstructorSig, 1407
  called by mkDomPvar, 433
letPrint, 885
  called by mapLetPrint, 884
  calls break, 885
  calls bright, 885
  calls gensymp, 885
  calls hasPair, 885
  calls isSharpVarWithNum, 885
  calls isgenvar, 885
  calls lassoc, 885
  calls puane, 885
  calls sayBrightlyNT, 885
  calls shortenForPrinting, 885
  uses $letAssoc, 885
  defun, 885
letPrint2, 887
  calls break, 887
  calls bright, 887
  calls gensymp, 887
  calls hasPair, 887
  calls isSharpVarWithNum, 887
  calls isgenvar, 887
  calls lassoc, 887
  calls mathprint, 887
  calls puane, 887
  calls print, 887
  uses $BreakMode, 887
  uses $letAssoc, 887
  catches, 887
  defun, 887
letPrint3, 888
  calls break, 888
  calls bright, 888
  calls gensymp, 888
calls hasPair, 888
calls isSharpVarWithNum, 888
calls isgenvar, 888
calls lassoc, 888
calls mathprint, 888
calls pname, 888
calls print, 888
calls spadcall, 888
uses $BreakMode, 888
uses $letAssoc, 888
catches, 888
defun, 888
lfcomment, 116
calledby scanComment, 116
defun, 116
lferror, 135
calledby scanError, 135
defun, 135
lffloat, 127
calledby scanExponent, 126
calls concat, 127
defun, 127
lfid, 115
calledby scanToken, 114
calledby scanWord, 126
defun, 115
lfinteger, 133
calledby scanNumber, 132
defun, 133
lfkey, 122
calledby scanKeyTr, 120
calledby scanPossFloat, 121
calledby scanWord, 126
calls keyword, 122
defun, 122
lfnegcomment, 118
calledby scanNegComment, 117
defun, 118
lfinteger, 134
calledby scanNumber, 132
calls concat, 134
defun, 134
lfspaces, 130
calledby scanSpace, 129
defun, 130
lfstring, 130
calledby scanString, 130
defun, 130
libConstructorSig, 1406
calls form2LispString, 1407
calls getdatabase, 1406
calls length, 1407
calls ncParseFromString, 1407
calls sayBrightly, 1407
calls sublislis, 1407
calls take, 1407
uses $TriangleVariableList, 1407
defun, 1406
library, 1013
calledby with, 947
calls extendLocalLibdb, 1013
calls localdatabase, 1013
calls tersyscommand, 1013
uses $newConlist, 1013
uses $options, 1013
defun, 1013
library help page, 625
manpage, 625
libstream, 1164
defstruct, 1164
line
usedby commandErrorIfAmbiguous, 470
usedby commandErrorMessage, 450
usedby doSystemCommand, 446
usedby intProcessSynonyms, 31
usedby processSynonyms, 32
usedby unAbbreviateKeyword, 470
line-clear
calledby ioclear, 972
line-handler, 967
usedby init-boot/spad-reader, 968
defvar, 967
line-past-end-p
calledby iostat, 970
calledby spad-short-error, 970
line-print
calledby iostat, 970
calledby spad-short-error, 970
line?, 351
calledby msgOutputter, 353
calls getMsgTag, 351
defun, 351
INDEX

linearFinalRequest, 1241
calls bcQueryInteger, 1241
calls explainLinear, 1241
calls sayBrightly, 1241
defun, 1241
lineoftoks, 111
called by intloopInclude0, 62
called by nclloopInclude0, 71
called by parseFromString, 47
called by parseNoMacroFromString, 1374
calls constoken, 111
calls dqUnit, 111
calls incPrefix?, 111
calls nextline, 111
calls scanIgnoreLine, 111
calls substring, 111
uses $floatok, 111
uses $f, 111
uses $linepos, 112
uses $ln, 112
uses $n, 112
uses $r, 112
uses $sz, 112
defun, 111
lisp help page, 627
manpage, 627
listConstructorAbbreviations, 484
called by abbreviationsSpad2Cmd, 483
called by displaySpad2Cmd, 535
calls queryUserKeyedMsg, 484
calls sayKeyedMsg, 485
calls string2id-n, 484
calls upcase, 484
calls whatSpad2Cmd, 485
defun, 484
listDecideHowMuch, 361
called by getPosStL, 356
calls poGlobalLinePosn, 361
calls poNpos?, 361
calls poPosImmediate?, 361
defun, 361
ListMember?
called by whichCat, 365
listOfStrings2String, 1302
defun, 1302
listOutpufter, 354
called by processMsgList, 369
calls msgOutpufter, 354
defun, 354
listSort
called by dbShowCons1, 1389
called by dbShowConstructorLines, 1397
called by domainDescendantsOf, 1349
called by errMsgSort, 369
called by kcaPage1, 1363
literals
called by displaySetOptionInformation, 655
called by displaySetVariableSettings, 657
called by initializeSetVariables, 653
called by set1, 809
lnCreate, 345
called by incLine1, 86
defun, 345
lnExtraBlanks, 345
called by scanError, 135
called by scanS, 131
called by scanToken, 114
defun, 345
lnFileName, 347
called by poFileName, 360
calls lnFileName?, 347
calls ncBug, 347
defun, 347
lnFileName?, 347
called by lnFileName, 347
called by lnImmediat?, 347
defun, 347
lnGlobalNum, 346
called by poGlobalLinePosn, 70
defun, 346
lnImmediate?, 347
called by poPosImmediate?, 360
calls lnFileName?, 347
defun, 347
lnLocalNum, 346
called by poLinePosn, 361
defun, 346
lnPlaceOfOrigin, 346
defun, 346
lnSetGlobalNum, 346
called by incRenumberItem, 74
defun, 346
lnString, 345
defun, 345
load, 629
calls sayKeyedMsg, 629
defun, 629
load help page, 629
manpage, 629
loadFunctor, 1038
called by loadFunctor, 1038
called by traceDomainConstructor, 881
calls loadFunctor, 1038
calls loadLibInNotLoaded, 1038
defun, 1038
loadIfNecessary
called by mkEvalable, 913
loadLib, 1035
called by browse, 493
calls categoryForm?, 1036
calls clearConstructorCache, 1035
calls getdatabase, 1035
calls installConstructor, 1035
calls isSystemDirectory, 1035
calls loadLibNoUpdate, 1035
calls nameString, 1035
calls pathnameDirectory, 1035
calls remprop, 1036
calls sayKeyedMsg, 1035
calls startTimingProcess, 1035
calls stopTimingProcess, 1036
calls updateCategoryTable, 1036
calls updateDatabase, 1035
local def $CategoryFrame, 1036
local ref $InteractiveMode, 1036
local ref $printLoadMsgs, 1036
defun, 1035
loadLibInNotLoaded
called by loadFunctor, 1038
loadLibNoUpdate, 1037
called by loadLib, 1035
called by localrllib, 1015
calls clearConstructorCache, 1037
calls getdatabase, 1037
calls installConstructor, 1037
calls sayKeyedMsg, 1037
calls stopTimingProcess, 1037
calls toplevel, 1037
local def $CategoryFrame, 1037
local ref $InteractiveMode, 1037
local ref $printLoadMsgs, 1037
defun, 1037
localdatabase, 1014
called by library, 1013
called by make-databases, 1019
calls localrllib, 1014
calls sayKeyedMsg, 1014
uses *index-filename*, 1014
uses $ConstructorCache, 1014
uses $forceDatabaseUpdate, 1014
defun, 1014
localrllib, 1015
called by localdatabase, 1014
calls addoperations, 1015
calls categoryForm?, 1015
calls getdatabase, 1015
calls installConstructor, 1015
calls loadLibNoUpdate, 1015
calls make-database, 1015
calls sayKeyedMsg, 1016
calls setExposeAddConstr, 1015
calls startTimingProcess, 1015
calls sublisls, 1015
calls updateCategoryTable, 1015
calls updateDatabase, 1015
uses *allOperations*, 1016
uses *allconstructors*, 1016
uses $FormalMapVariableList, 1016
defun, 1015
lookupInDomainVector, 1078
called by basicLookup, 1076
called by oldCompLookup, 1079
calls basicLookupCheckDefaults, 1078
calls spadcall, 1078
defun, 1078
loopIters2Sex, 311
called by pf2Sex1, 300
called by pfCollect2Sex, 314
calls pf2Sex1, 312
defun, 311
lotsof, 1043
called by wrap, 1043
defun, 1043
ltrace, 632
calls trace, 632
defun, 632
ltrace help page, 631
manpage, 631
lxOK1
called by xlOK, 84
mac0Define, 230
called by mac0MLambdaApply, 224
called by macMacro, 229
uses $pfMacros, 230
defun, 230
mac0ExpandBody, 224
called by mac0MLambdaApply, 224
called by macId, 227
calls mac0 InfiniteExpansion, 224
calls macExpand, 225
calls pfSourcePosition, 224
uses $macActive, 225
uses $posActive, 225
defun, 224
mac0Get, 228
called by macId, 227
calls ifcdr, 228
uses $pfMacros, 228
defun, 228
mac0GetName, 226
called by mac0 InfiniteExpansion, name, 226
calls pfMLambdaBody, 226
uses $pfMacros, 226
defun, 226
mac0 InfiniteExpansion, 225
called by mac0 ExpandBody, 224
calls mac0 InfiniteExpansion, name, 225
calls ncSoftError, 225
calls pform, 225
defun, 225
mac0 InfiniteExpansion, name, 226
called by mac0 InfiniteExpansion, 225
calls mac0 GetName, 226
calls pname, 226
defun, 226
mac0MLambdaApply, 223
called by macApplication, 223
calls mac0Define, 224
calls mac0ExpandBody, 224
calls ncHardError, 223
calls pf0MLambdaArgs, 223
calls pfId?, 224
calls pfMLambdaBody, 223
calls pfSourcePosition, 223
calls pform, 224
uses $macActive, 224
uses $pfMacros, 224
uses $posActive, 224
defun, 223
mac0SubstituteOuter, 231
called by mac0SubstituteOuter, 231
called by macSubstituteOuter, 230
calls mac0SubstituteOuter, 231
calls macLambdaParameterHandling, 231
calls macSubstituteld, 231
calls pfId?, 231
calls pfLambda?, 231
calls pfLeaf?, 231
calls pfParts, 231
defun, 231
macApplication, 223
called by macExpand, 222
calls mac0MLambdaApply, 223
calls macExpand, 223
calls pf0ApplicationArgs, 223
calls pfApplicationOp, 223
calls pfMLambda?, 223
calls pfMapParts, 231
defun, 223
macExpand, 222
called by mac0ExpandBody, 225
called by macApplication, 223
called by macExpand, 222
called by macLambda, mac, 229
called by macWhere, mac, 228
called by macroExpanded, 222
calls macApplication, 222
calls macExpand, 222
calls macId, 222
calls macLambda, 222
calls macMacro, 222
calls macWhere, 222
calls pfApplication?, 222
calls pfId?, 222
calls pfLambda?, 222
calls pfMacro?, 222
calls pfMapParts, 222
calls pfWhere?, 222
defun, 222

macId, 227
called by macExpand, 222
calls mac0ExpandBody, 227
calls mac0Get, 227
calls pfCopyWithPos, 227
calls pfIdSymbol, 227
calls pfSourcePosition, 227
uses $macActive, 227
uses $posActive, 227
defun, 227

macLambda, 228
called by macExpand, 222
calls macLambda,mac, 228
uses $pfMacros, 229
defun, 228

macLambda,mac, 229
called by macLambda, 228
calls macExpand, 229
calls pfMapParts, 229
uses $pfMacros, 229
defun, 229

macLambdaParameterHandling, 231
called by mac0SubstituteOuter, 231
called by macLambdaParameterHandling, 231
called by macSubstituteOuter, 230
calls macLambdaParameterHandling, 231
calls pf0LambdaArgs, 231
calls pf0MLambdaArgs, 231
calls pfAbSynOp, 231
calls pfIdSymbol, 231
calls pfLambda?, 231
calls pfLeaf?, 231
calls pfLeafPosition, 231
calls pfLeaf, 231
calls pfMLambda?, 231
calls pfParts, 231
calls pfTypedId, 231
defun, 231

macMacro, 229
called by macExpand, 222
calls mac0Define, 229
calls macSubstituteOuter, 229
calls ncSoftError, 229
calls pfId?, 229
calls pfIdSymbol, 229
calls pfMLambda?, 229
calls pfMacroLhs, 229
calls pfMacroRhs, 229
calls pfMacro, 229
calls pfNothing?, 229
calls pfNothing, 229
calls pfSourcePosition, 229
calls pfForm, 229
defun, 229

macroExpanded, 222
called by parseFromString, 47
called by phMacro, 221
called by macExpand, 222
uses $macActive, 222
uses $posActive, 222
defun, 222

macSubstituteId, 232
called by mac0SubstituteOuter, 231
calls pfIdSymbol, 232
defun, 232

macSubstituteOuter, 230
called by mac0SubstituteOuter, 230
called by macLambdaParameterHandling, 230
defun, 230

macWhere, 228
called by macExpand, 222
calls macWhere,mac, 228
uses $pfMacros, 228
defun, 228

macWhere,mac, 228
called by macWhere, 228
called by macExpand, 228
called by pfMapParts, 228
uses $pfMacros, 228
defun, 228

make-absolute-filename, 35
called by reroot, 39
uses $spadroot, 35
defun, 35
make-appendstream, 982
called by makeStream, 983
calls make-filename, 982
defun, 982
make-cdouble-matrix, 1085
defmacro, 1085
make-cdouble-vector, 1082
defmacro, 1082
make-database
called by interpOpen, 1004
called by localnrlib, 1015
called by setdatabase, 1009
make-databases, 1018
calls allConstructors, 1019
calls browserAutoloadOnceTrigger, 1019
calls buildLibdb, 1019
calls categoryForm?, 1019
calls dbSplitLibdb, 1019
calls domainsOf, 1019
calls getConstructorForm, 1019
calls getEnv, 1019
calls localdatabase, 1019
calls mkDependantsHashTable, 1019
calls mkTopicHashTable, 1019
calls mkUsersHashTable, 1019
calls mkDependentHashTable, 1019
calls mkUsersHashTable, 1019
calls write-browsedb, 1019
calls write-categorydb, 1019
calls write-interpdb, 1019
calls write-operationdb, 1019
calls write-warmdata, 1019
uses *allconstructors*, 1019
uses *operation-hash*, 1019
uses *sourcefiles*, 1019
uses $constructorList$, 1019
defun, 1018
make-double-matrix, 1084
defmacro, 1084
make-double-matrix1, 1084
defmacro, 1084
make-double-vector, 1081
defmacro, 1081
make-double-vector1, 1081
defmacro, 1081
make-filename
called by make-appendstream, 982
called by make-outstream, 981
called by pathname, 1042
make-hashable
called by dbDocTable, 1382
make-instream, 981
called by dewritify, dewritifyInner, 612
called by incConsoleInput, 100
called by incFileInput, 100
called by newHelpSpad2Cmd, 573
calls makeInputFilename, 981
defun, 981
make-monitor-data
called by monitor-add, 1166
make-outstream, 981
called by makeStream, 983
called by setOutputAlgebra, 763
called by setOutputFormula, 790
called by setOutputHtml, 781
called by setOutputMathml, 776
called by setOutputOpenMath, 786
called by setOutputTex, 797
calls make-filename, 981
defun, 981
make-string
called by execucuteQuietCommand, 46
makeByteWordVec2, 1155
defun, 1155
makeFullNamestring, 985
called by makeStream, 985
defun, 985
makeHistFileName, 579
called by closeInterpreterFrame, 562
called by histFileName, 580
called by oldHistFileName, 580
called by restoreHistory, 597
called by saveHistory, 595
calls makePathname, 579
defun, 579
makeInitialModemapFrame, 35
called by restart, 15
calls copy, 35
uses $InitialModemapFrame$, 35
defun, 35
makeInputFilename, 983
called by initHist, 581
INDEX

called by make-instream, 981

called by newHelpSpad2Cmd, 572
called by restoreHistory, 597
called by saveHistory, 595
called by updateSourceFiles, 546
called by workfilesSpad2Cmd, 949
defun, 983

makeLeaderMsg, 377
called by processChPosesForOneLine, 376
uses $nopos, 377
uses $preLength, 377
defun, 377

makeLongSpaceString
called by printStorage, 56

makeLongTimeString
called by printTypeAndTimeNormal, 57
called by printTypeAndTimeSaturn, 58

makeMatrix1U16, 1060
defmacro, 1060
makeMatrix1U32, 1062
defmacro, 1062
makeMatrix1U8, 1059
defmacro, 1059
makeMatrixU16, 1060
defmacro, 1060
makeMatrixU32, 1061
defmacro, 1061
makeMatrixU8, 1058
defmacro, 1058

makeMsgFromLine, 371
called by processMsgList, 369
calls char, 371
calls getLinePos, 371
calls getLineText, 371
calls poGlobalLinePosn, 371
calls poLinePosn, 371
calls rep, 371
calls size, 371
calls strconc, 371
uses $preLength, 371
defun, 371

makeOrdinal, 920
called by evaluateType1, 918
defun, 920

makePathname, 1042
called by histInputFileName, 580

called by makeHistFileName, 579
called by readSpad2Cmd, 642
called by sayMSG2File, 331
calls object2String, 1042
calls pathname, 1042
defun, 1042

makeSpadCommand, 1295
defun, 1295

makeStream, 983
called by setOutputFortran, 770
calls make-appendstream, 983
calls make-outstream, 983
calls makeFullNamestring, 985
defun, 983

MakeSymbol
called by assertCond, 94
called by ifCond, 87
manexp, 1103
defun, 1103
mapage
abbreviations help page, 481
boot help page, 487
browse help page, 489
cd help page, 495
clear help page, 497
close help page, 509
compile help page, 513
copyright help page, 517
credits help page, 525
describe help page, 527
display help page, 533
delimit help page, 543
find help page, 547
frame help page, 549
help help page, 569
history help page, 575
include help page, 621
library help page, 625
lisp help page, 627
load help page, 629
ltrace help page, 631
pquit help page, 633
quit help page, 637
read help page, 641
regress help page, 645
savesystem help page, 649
mapletPrint, 884

called by getAliasIfTracedMapParameter, 884

called by getBpiNameIfTracedMap, 884

defun, 884

mapStringize, 1261
defun, 1261

mathprint
called by displayMacro, 454
called by displayValue, 459
called by letPrint2, 887
called by letPrint3, 888

maxindex
called by mkConArgSublis, 1405
called by processSynonymLine, removeKeyFromLine, 834
called by removeUndoLines, 934

defun, 1048

member, 1048
called by bcSolveEquations, 1236
called by clearCmdParts, 505
called by dbConsHeading, 1395
called by dbShowConsDoc1, 1393
called by dbShowCons, 1388
called by displayMacros, 538
called by displayProperties, 462
called by doSystemCommand, 446
called by evaluateType, 916
called by fixObjectForPrinting, 456
called by getBrowseDatabase, 1090
called by handleNoParseCommands, 470
called by hasCateSpecialNew, 436
called by historySpad2Cmd, 582
called by importFromFrame, 563
called by interpret2, 53
called by isDomainValuedVariable, 959
called by isSubForRedundantMapName, 873
called by isValType, 1373
called by readSpad2Cmd, 642
called by reportOpsFromUnitDirectly, 821
called by selectOption, 479
called by setExposeAddConstr, 700
called by setExposeAddGroup, 699
called by setExposeDropConstr, 704
called by setExposeDropGroup, 702
called by setOutputAlgebra, 763
called by setOutputFormula, 790
called by setOutputFortran, 770
called by setOutputHtml, 781
called by setOutputMathml, 776
called by setOutputOpenMath, 786
called by setOutputTex, 797
called by showSpad2Cmd, 814
called by transTraceItem, 863
called by translateYesNoToTrueFalse, 658
called by updateSourceFiles, 546
defun, 1048

menuButton
called by kePageDisplay, 1354
called by kePage, 1352

mergePathnames, 1041

called by readSpad2Cmd, 642
calls pathnameDirectory, 1041
calls pathnameName, 1041
calls pathnameType, 1041
defun, 1041

messageprint, 1048
called by brightprint, 1048
defun, 1048

messageprint-1, 1049
called by brightprint-0, 1048
called by messageprint-1, 1049
called by messageprint-2, 1049
calls identp, 1049
calls messageprint-1, 1049
calls messageprint-2, 1049
defun, 1049

messageprint-2, 1049
called by messageprint-1, 1049
called by messageprint-2, 1049
calls messageprint-1, 1049
calls messageprint-2, 1049
defun, 1049
meta-error-handler
  used by init-boot/spad-reader, 968
MINUSCOMMENT, 106
defvar, 106
mkAtree
  called by evaluateType1, 918
called by evaluateType, 916
called by reportOperations, 816
mkAtreeWithSrcPos
  called by interpret1, 52
mkCompanionPage
  called by recordAndPrint, 54
mkConArgSublis, 1405
calls digitp, 1405
calls digits2Names, 1405
calls maxindex, 1405
calls pname, 1405
defun, 1405
mkConform, 1374
called by conOpPage1, 1375
called by kDomainName, 1370
called by kePage, 1351
called by kiPage, 1351
calls concat, 1374
calls ncParseFromString, 1374
calls nequal, 1374
calls parseNoMacroFromstring, 1374
calls pp, 1374
calls sayBrightlyNT, 1374
calls systemError, 1374
defun, 1374
mkCurryFun, 1272
defun, 1272
mkDependentsHashTable
called by make-databases, 1019
mkDomPvar, 433
called by hasCaty, 420
calls domArg, 433
calls length, 433
local ref $FormalMapVariableList, 433
defun, 433
mkDomTypeForm, 1348
called by kArgPage, 1346
called by mkDomTypeForm, 1348
called hasIndent, 1348
called mkDomTypeForm, 1348
called sublis, 1348
defun, 1348
mkEvalable, 913
called by dbMkEvalable, 1372
called by evalDomain, 913
called by mkEvalableMapping, 915
called by mkEvalableRecord, 915
called by mkEvalableUnion, 915
called by mkEvalable, 913
called by writify,writifyInner, 607
calls bpiname, 913
calls constructor?, 913
calls devaluate, 913
calls fbpip, 913
calls getdatabase, 913
calls loadIfNecessary, 913
calls mkEvalableMapping, 913
calls mkEvalableRecord, 913
calls mkEvalableUnion, 913
called by mkEvalable, 913
called by mkq, 913
called qcar, 913
called qcdr, 913
local ref $EmptyMode, 913
local ref $Integer, 913
defun, 913
mkEvalableMapping, 915
called by mkEvalable, 913
called by mkEvalable, 915
defun, 915
mkEvalableRecord, 915
called by mkEvalable, 913
called by mkEvalable, 915
defun, 915
mkEvalableUnion, 915
called by mkEvalable, 913
called by mkEvalable, 915
defun, 915
mkLineList, 68
called by intloopEchoParse, 67
defun, 68
mkprompt, 40
called by SpadInterpretStream, 27
called by intloopReadConsole, 28
called by serverReadLine, 42
calls concat, 40
calls currenttime, 40
calls substring, 40
uses $IOindex, 40
uses $inputPromptType, 40
uses $interpreterFrameName, 40
defun, 40
mkq
called by mkEvalable, 913
called by traceDomainConstructor, 881
mkSetTitle, 1302
defun, 1302
mkTopicHashTable
called by make-databases, 1019
mkUnixPattern, 1341
defun, 1341
mkUserConstructorAbbreviation
called by abbreviationsSpad2Cmd, 483
mkUsersHashTable
called by make-databases, 1019
modes
called by clearCmdParts, 505
MONITOR,EVALTRAN
called by break, 906
monitor-add, 1166
called by monitor-file, 1169
calls make-monitor-data, 1166
calls monitor-delete, 1166
uses *monitor-table*, 1166
defun, 1166
monitor-apropos, 1179
uses *monitor-table*, 1179
defun, 1179
monitor-autoload, 1174
defun, 1174
monitor-checkpoint, 1171
uses *monitor-table*, 1171
uses *print-package*, 1171
defun, 1171
monitor-data, 1164
defstruct, 1164
monitor-decr, 1168
uses *monitor-table*, 1168
defun, 1168
monitor-delete, 1166
called by monitor-add, 1166
called by monitor-tested, 1170
uses *monitor-table*, 1170
defun, 1166
monitor-dirname, 1174
uses *monitor-nrlibs*, 1174
defun, 1174
monitor-disable, 1167
uses *monitor-table*, 1167
defun, 1167
monitor-enable, 1166
uses *monitor-table*, 1166
defun, 1166
monitor-end, 1165
uses *monitor-table*, 1165
defun, 1165
monitor-exposedp, 1176
defun, 1176
monitor-file, 1169
calls monitor-add, 1169
catches, 1169
defun, 1169
throws, 1169
monitor-help, 1172
defun, 1172
monitor-incr, 1168
uses *monitor-table*, 1168
defun, 1168
monitor-info, 1169
uses *monitor-table*, 1169
defun, 1169
monitor-inittable, 1164
uses *monitor-table*, 1164
defun, 1164
monitor-libname, 1175
defun, 1175
monitor-nrlib, 1175
uses *monitor-table*, 1175
defun, 1175
monitor-parse, 1178
called by monitor-spadfile, 1178
defun, 1178
monitor-percent, 1179
uses *monitor-table*, 1179
defun, 1179
INDEX

monitor-readinterp, 1176
  called by monitor-report, 1177
  uses *monitor-domains*, 1176
  catches, 1176
defun, 1176
  throws, 1176

monitor-report, 1177
  calls monitor-readinterp, 1177
  uses *monitor-domains*, 1177
defun, 1177

monitor-reset, 1167
  uses *monitor-table*, 1167
defun, 1167

monitor-restore, 1171
defun, 1171

monitor-results, 1165
  uses *monitor-table*, 1165
defun, 1165

monitor-spadfile, 1178
  calls monitor-parse, 1178
  uses *monitor-domains*, 1178
  catches, 1178
defun, 1178
  throws, 1178

monitor-tested, 1170
  calls monitor-delete, 1170
  uses *monitor-table*, 1170
defun, 1170

monitor-untested, 1169
  uses *monitor-table*, 1169
defun, 1169

monitor-write, 1170
defun, 1170

moreExactSolution
  called by finalExactRequest, 1242

msgCreate, 347
  called by ncBug, 368
  called by ncHardError, 352
  called by ncSoftError, 351
calls initImPr, 348
calls initToWhere, 348
calls putDatabaseStuff, 348
calls setMsgForcedAttrList, 348
defun, 347

msgImPr?, 358
  called by alreadyOpened?, 363
called by getPosStL, 356
called by processKeyedError, 353
called by showMsgPos?, 357
calls getMsgCatAttr, 358
defun, 358

msgNoRep?, 375
called by redundant, 374
calls getMsgCatAttr, 375
defun, 375

msgOutputter, 353
  called by listOutputter, 354
called by processKeyedError, 353
calls alreadyOpened?, 353
calls flowSegmentedMsg, 353
calls getStFromMsg, 353
calls leader?, 353
calls line?, 353
calls sayBrightly, 353
calls toFile?, 353
calls toScreen?, 353
uses $linelength, 353
defun, 353

msgText, 60
  called by printTypeAndTimeNormal, 57
calls flowSegmentedMsg, 60
calls getKeyedMsg, 60
calls segmentKeyedMsg, 60
calls substituteSegmentedMsg, 60
uses $linelength, 60
uses $margin, 60
defun, 60

msort
  called by apropos, 945
called by displayOperationsFromLisplib, 820
called by displayProperties, 461
called by displayWorkspaceNames, 454
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
called by saveDependentsHashTable, 1022
called by saveUsersHashTable, 1022
called by setExposeAddConstr, 700
called by setExposeAddGroup, 699
called by setExposeDropConstr, 704
called by whatConstructors, 945
myWritable?, 1093
calls error, 1093
    calls fnameDirectory, 1093
    calls fnameExists?, 1093
    calls writeablep, 1093
defun, 1093
myWriteable?
called by fnameWritable?, 1093

names
called by frameSpad2Cmd, 566
namestring, 1040
called by bcUnixTable, 1397
called by editFile, 545
called by loadLib, 1035
called by newHelpSpad2Cmd, 573
called by readSpad2Cmd, 642
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
called by restoreHistory, 597
called by saveHistory, 596
called by setExposeAddGroup, 699
called by setExpose, 697
called by workfilesSpad2Cmd, 949
called by writeInputLines, 587
calls pathname, 1040
defun, 1040
ncAbort
called by ncBug, 368
ncAlist, 415
called by getMsgCatAttr, 358
called by ncElQ, 416
called by ncPutQ, 417
called by setMsgCatAttr, 365
called by setMsgUnforcedAttr, 367
called by tokPosn, 413
called by identp, 415
called by ncBug, 415
called by qcar, 415
called by qcdr, 416
defun, 415
ncBug, 368
called by getMsgPos2, 379
called by incHandleMessage, 74
called by InFileName, 347
called by ncAlist, 415
called by ncElQ, 416
called by ncTag, 415
called by poGlobalLinePosn, 70
called by enable-backtrace, 368
called by msgCreate, 368
called by ncAbort, 368
called by processKeyedError, 368
uses $newcompErrorCount, 368
defun, 368
ncConversationPhase, 66
called by intloopSpadProcess, interp, 64
called by ncConversationPhase, wrapup, 66
uses $ncMsgList, 66
defun, 66
ncConversationPhase, wrapup, 66
called by ncConversationPhase, 66
uses $ncMsgList, 66
defun, 66
ncElQ, 416
called by intloopSpadProcess, interp, 64
called by phIntReportmsgs, 64
called by phInterpret, 65
called by phMacro, 221
called by ncAlist, 416
called by ncBug, 416
called by qassq, 416
defun, 416
ncError, 67
called by intloopSpadProcess, interp, 64
called by ncHardError, 352
defun, 67
throws, 67
ncHardError, 352
called by macOMLambdaApply, 223
called by desiredMsg, 352
called by msgCreate, 352
called by ncError, 352
called by processKeyedError, 352
uses $newcompErrorCount, 352
defun, 352
ncIntLoop, 23
called by ncTopLevel, 23
called by intloop, 23
uses curinstream, 24
uses curoutstream, 24
defun, 23
ncloopCommand, 478
   called by intloopReadConsole, 28
   calls $systemCommandFunction, 478
   calls ncloopInclude1, 478
   calls ncloopPrefix?, 478
   uses $systemCommandFunction, 478
   defun, 478
ncloopDQlines, 70
   called by intloopEchoParse, 67
   called by ncloopParse, 36
   calls StreamNull, 70
   calls poGlobalLinePosn, 70
   calls streamChop, 70
   calls tokPosn, 70
   defun, 70
ncloopEchoParse
   called by ncloopInclude0, 71
ncloopEscaped, 35
   called by intloopReadConsole, 28
   defun, 35
ncloopIncFileName, 622
   called by ncloopInclude1, 621
   calls concat, 622
   calls incFileName, 622
   defun, 622
ncloopInclude, 622
   called by ncloopInclude1, 621
   calls ncloopInclude0, 622
   defun, 622
ncloopInclude0, 71
   called by ncloopInclude, 622
   calls incStream, 71
   calls insertpile, 71
   calls lineoftoks, 71
   calls ncloopEchoParse, 71
   calls ncloopProcess, 71
   calls next, 71
   uses $lines, 71
   defun, 71
ncloopInclude1, 621
   called by ncloopCommand, 478
   calls ncloopIncFileName, 621
   calls ncloopInclude, 621
   defun, 621
ncloopParse, 36
   called by parseFromString, 47
   called by parseNoMacroFromString, 1373
   calls dqToList, 36
   calls ncloopDQlines, 36
   calls npParse, 36
   defun, 36
ncloopPrefix?, 479
   called by ncloopCommand, 478
   called by streamChop, 70
   defun, 479
ncloopPrintLines, 68
   called by intloopEchoParse, 67
   defun, 68
ncloopProcess
   called by ncloopInclude0, 71
ncParseFromString, 1067
   called by conOpPage1, 1375
   called by kcdePage, 1366
   called by kcuPage, 1367
   called by libConstructorSig, 1407
   called by mkConform, 1374
   defun, 1067
ncPutQ, 416
   called by constoken, 113
   called by inttoken, 63
   called by phIntReport, 65
   called by phMacro, 221
   called by phParse, 64
   called by setMsgCatlessAttr, 365
   called by setMsgForcedAttr, 364
   called by setMsgUnforcedAttr, 367
   called by tokConstruct, 411
   calls ncAlist, 417
   calls ncTag, 417
   calls qassq, 417
   defun, 416
ncSoftError, 351
   called by incHandleMessage, 74
   called by intloopSpadProcess, 63
   called by phIntReportMsgs, 65
   called by phMacro, 221
   called by phParse, 64
called by scanS, 131
called by sigIgnoredFromTo, 191
called by sySpecificErrorAtToken, 192
calls desiredMsg, 351
calls msgCreate, 351
calls processKeyedError, 351
uses $newcompErrorCount, 351
defun, 351
ncTag, 415
called by getMsgTag, 350
called by ncPutQ, 417
called by tokType, 413
calls identity, 415
calls ncBug, 415
calls qcar, 415
defun, 415
ncTopLevel, 23
called by runspad, 19
calls nclntLoop, 23
uses *cof*, 23
uses $InteractiveFrame, 23
uses $InteractiveMode, 23
uses $boot, 23
uses $e, 23
uses $newspad, 23
uses $spad, 23
uses in-stream, 23
defun, 23
nequal
called by dbConsHeading, 1395
called by kcPage, 1358
called by kcedePage, 1366
called by kcuPage, 1367
called by kdpagelnfo, 1345
called by mkConform, 1374
new
called by fframeSpad2Cmd, 566
newfHelpSpad2Cmd, 572
called by helpSpad2Cmd, 572
calls concat, 572
calls make-instream, 573
calls makeInputFilename, 572
calls nameofstring, 573
calls obey, 572
calls pname, 573
calls poundsign, 573
calls sayKeyedMsg, 573
calls say, 573
calls selectOptionLC, 573
uses $syscommands, 573
uses $useFullScreenHelp, 573
defun, 572
next, 36
called by frameSpad2Cmd, 566
called by intloopInclude0, 61
called by intloopProcessString, 36
called by nclnloopInclude0, 71
called by next1, 37
called by parseFromString, 46
called by parseNoMacroFromString, 1373
called by zsystemdevelopment1, 954
calls delay, 36
calls next1, 36
defun, 36
next-lines-clear, 972
called by init-boot/spad-reader, 968
uses boot-line-stack, 972
defun, 972
next-lines-show, 971
called by iostat, 970
uses boot-line-stack, 971
defun, 971
next-token
used by token-stack-show, 971
next1, 37
called by next, 36
calls StreamNull, 37
calls incAppend, 37
calls next, 37
defun, 37
nextInterpreterFrame, 560
called by frameSpad2Cmd, 566
calls updateFromCurrentInterpreterFrame, 560
uses $InterpreterFrameRing, 560
defun, 560
nextline, 112
called by lineoftoks, 111
called by scanEsc, 124
calls npNull, 112
calls strposl, 112
uses $f, 113
uses $linepos, 113
uses $ln, 113
uses $n, 113
uses $r, 113
uses $sz, 112
defun, 112

nmsort
called by getWorkspaceNames, 455

nonBlank, 69
defun, 69

npADD, 158
called by npExpress1, 179
calls npAdd, 158
calls npPop1, 158
calls npPush, 158
calls npType, 158
defun, 158

npAdd, 159
called by npADD, 158
called by npPrimary2, 158
calls npCompMissing, 159
calls npDefinitionOrStatement, 159
calls npEqKey, 159
calls npEqPeek, 159
calls npPop1, 159
calls npPop2, 159
calls npPush, 159
calls npRestore, 159
calls npState, 159
calls npTrap, 159
calls npVariable, 159
calls pfAdd, 159
calls pfNothing, 159
defun, 159

npAmpersand, 204
called by npAmpersandFrom, 202
called by npAtom2, 160
calls npEqKey, 204
calls npName, 204
calls npTrap, 204
defun, 204

npAmpersandFrom, 202
called by npSynthetic, 198
calls npAmpersand, 202
calls npFromdom, 202
defun, 202

npAndOr, 181
called by npImport, 180
called by npInline, 174
called by npSuchThat, 176
called by npVoid, 179
called by npWhile, 177
calls npEqKey, 181
calls npPop1, 182
calls npPush, 182
calls npTrap, 182
defun, 181

npAngleBared, 186
called by npBracketed, 185
calls npEnclosed, 186
calls pfHide, 186
defun, 186

npAnyNo, 162
called by npColon, 217
called by npEncap, 182
called by npTypified, 218
defun, 162

npApplication, 162
called by npFromdom1, 203
called by npFromdom, 203
called by npSCategory, 153
called by npTypedForm, 220
called by npTypified, 218
calls npApplication2, 162
calls npDotted, 162
calls npPop1, 162
calls npPop2, 162
calls npPrimary, 162
calls npPush, 162
calls pfApplication, 162
defun, 162

npApplication2, 163
called by npApplication2, 163
called by npApplication, 162
calls npApplication2, 162
calls npDotted, 163
calls npPop1, 163
calls npPop2, 163
calls npPrimary, 163
calls npPush, 163
calls pfApplication, 163
defun, 163
npArith, 200
called by npInterval, 199
calls npLeftAssoc, 200
calls npSum, 200
defun, 200

npAssign, 216
called by npExit, 216
called by npLoop, 175
called by npPileExit, 216
calls npAssignment, 216
calls npBackTrack, 216
calls npMDEF, 216
defun, 216

npAssignment, 217
called by npAssign, 216
calls npAssignVariable, 217
calls npEqKey, 217
calls npGives, 217
calls npPop1, 217
calls npPop2, 217
calls npPush, 217
calls npTrap, 217
calls pfAssign, 217
defun, 217

npAssignVariable, 217
called by npAssignment, 217
calls npColon, 217
calls npPop1, 217
calls npPop2, 217
calls npPush, 217
calls pfListOf, 217
defun, 217

npAtom1, 183
called by npPrimary1, 164
calls npBDefinition, 183
calls npConstTok, 183
calls npDollar, 183
calls npFromdom, 183
calls npName, 183
calls npPDefinition, 183
defun, 183

npAtom2, 159
called by npPrimary2, 158
calls npAmpersand, 160
calls npFromdom, 160
calls npInfixOperator, 159
calls npPrefixColon, 160
defun, 159

npBacksetElse, 197
called by npElse, 196
calls npEqKey, 197
defun, 197

npBackTrack, 148
called by npAssign, 216
called by npDefinitionOrStatement, 147
called by npExit, 215
called by npGives, 148
called by npMDEF, 165
calls npEqPeek, 148
calls npRestore, 148
calls npState, 148
calls npTrap, 148
defun, 148

npBDefinition, 185
called by npAtom1, 183
called by npEncl, 182
calls npBracketed, 185
calls npDefinitionlist, 185
calls npPDefinition, 185
defun, 185

npboot, 472
called by handleNoParseCommands, 470
defun, 472

npBPileDefinition, 188
called by npPrimary1, 164
calls npPileBracketed, 188
calls npPileDefinitionlist, 188
calls npPop1, 188
calls npPush, 188
calls pfListOf, 188
calls pfSequence, 188
defun, 188

npBraced, 186
called by npBracketed, 185
calls npEnclosed, 186
calls pfBraceBar, 186
calls pfBrace, 186
defun, 186

npBracketed, 186
called by npBracketed, 185
calls npEnclosed, 186
calls pfBracketBar, 186
calls pfBracket, 186
defun, 186
npBracketed, 185
called by npBDefinition, 185
calls npAngleBared, 185
calls npBraced, 185
calls npBracketed, 185
calls npParened, 185
defun, 185
npBreak, 174
called by npStatement, 170
calls npEqKey, 174
calls npPush, 174
calls pfBreak, 174
calls pNothing, 174
defun, 174
npBy, 199
called by npForIn, 177
called by npSynthetic, 198
calls npInterval, 199
calls npLeftAssoc, 199
defun, 199
npCategory, 153
called by npCategoryL, 152
calls npPP, 153
calls npSCategory, 153
defun, 153
npCategoryL, 152
called by npSCategory, 153
called by npWith, 150
calls npCategory, 152
calls npPop1, 152
calls npPush, 152
calls pfUnSequence, 152
defun, 152
npCoerceTo, 220
called by npTypeStyle, 219
calls npTypedForm, 220
calls pfCoerceTo, 220
defun, 220
npColon, 217
called by npAssignVariable, 217
called by npPower, 202
calls npAnyNo, 217
calls npTagged, 217
calls npTypified, 217
defun, 217
npColonQuery, 219
called by npTypeStyle, 219
calls npTypedForm, 219
calls pfRetractTo, 219
defun, 219
npComma, 146
called by npQualDef, 145
calls npQualifiedDefinition, 146
calls npTuple, 146
defun, 146
npCommaBackSet, 146
called by npTuple, 146
calls npEqKey, 146
defun, 146
npCompMissing, 151
called by npAdd, 159
called by npForIn, 177
called by npLetQualified, 166
called by npLoop, 175
called by npWith, 150
calls npEqKey, 151
calls npMissing, 151
defun, 151
npConditional, 195
called by npConditionalStatement, 180
called by npWConditional, 195
calls npElse, 195
calls npEqKey, 195
calls npLogical, 195
calls npMissing, 195
calls npTrap, 195
defun, 195
npConditionalStatement, 180
called by npExpress1, 179
calls npConditional, 180
calls npQualifiedDefinition, 180
defun, 180
npConstTok, 184
called by npAtom1, 183
calls npEqPeek, 184
calls npNext, 184
calls npPop1, 184
calls npPrimary1, 184
calls npPush, 184
calls npRestore, 184
calls npState, 184
calls pfSymb, 184
calls tokPosn, 184
calls tokType, 184
uses $stok, 184
defun, 184
npDDInfKey, 208
called by npInfGeneric, 207
calls npEqKey, 208
calls npInfKey, 208
calls npPop1, 208
calls npPush, 208
calls npRestore, 208
calls npState, 208
calls pfSymb, 208
calls tokConstruct, 208
calls tokPart, 208
calls tokPosn, 208
uses $stok, 208
defun, 208
npDecl, 214
called by npVariableName, 214
calls npEqKey, 214
calls npPop1, 214
calls npPop2, 214
calls npPush, 214
calls npTrap, 214
calls npType, 214
calls pfTyped, 214
defun, 214
npDef, 187
called by npDefinitionOrStatement, 167
called by npDefinitionOrStatement, 147
called by npDefn, 187
called by npFix, 166
calls npDefTail, 187
calls npMatch, 187
calls npPop1, 187
calls npPush, 187
calls npTrap, 187
calls pfCheckItOut, 187
calls pfDefinition, 187
calls pfPushBody, 187
defun, 187
npDefaultDecl, 170
called by npDefaultItem, 169
calls npEqKey, 170
calls npPop1, 170
calls npPop2, 170
calls npPush, 170
calls npTrap, 170
calls npType, 170
calls pfParts, 170
calls pfSpread, 170
defun, 170
npDefaultItem, 169
called by npSDefaultItem, 169
calls npDefaultDecl, 169
calls npTrap, 169
calls npTypeVariable, 169
defun, 169
npDefaultItemlist, 168
called by npTyping, 168
calls npPC, 168
calls npPop1, 169
calls npPush, 169
calls npTrap, 169
calls pfUnSequence, 169
defun, 168
npDefaultValue, 194
called by npSCategory, 153
calls npDefinitionOrStatement, 195
calls npEqKey, 194
calls npPop1, 195
calls npPush, 195
calls npTrap, 195
calls pfAdd, 195
calls pfNothing, 195
defun, 194
npDefinition, 167
called by npLetQualified, 166
called by npQualified, 147
calls npDefinitionOrItem, 167
calls npPP, 167
calls npPop1, 167
calls npPush, 167
calls npSequenceToList, 167
defun, 167
npDefinitionOrItem, 167
called by npDefinition, 167
calls npDefn, 168
calls npDef, 167
calls npEqPeek, 167
<table>
<thead>
<tr>
<th>Function</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>npImport</td>
<td>167</td>
</tr>
<tr>
<td>npMacro</td>
<td>167</td>
</tr>
<tr>
<td>npRestore</td>
<td>167</td>
</tr>
<tr>
<td>npStatement</td>
<td>167</td>
</tr>
<tr>
<td>npState</td>
<td>167</td>
</tr>
<tr>
<td>npTrap</td>
<td>168</td>
</tr>
<tr>
<td>npTyping</td>
<td>167</td>
</tr>
<tr>
<td>defun</td>
<td>167</td>
</tr>
<tr>
<td>npDefinitionList</td>
<td>193</td>
</tr>
<tr>
<td>called by npBDefinition</td>
<td>185</td>
</tr>
<tr>
<td>called by npPDefinition</td>
<td>183</td>
</tr>
<tr>
<td>called by npPileDefinitionList</td>
<td>189</td>
</tr>
<tr>
<td>calls npQualDef</td>
<td>193</td>
</tr>
<tr>
<td>calls npSemiListing</td>
<td>193</td>
</tr>
<tr>
<td>defun</td>
<td>193</td>
</tr>
<tr>
<td>npDefinitionOrStatement</td>
<td>147</td>
</tr>
<tr>
<td>called by npAdd</td>
<td>159</td>
</tr>
<tr>
<td>called by npDefTail</td>
<td>194</td>
</tr>
<tr>
<td>called by npDefaultValue</td>
<td>195</td>
</tr>
<tr>
<td>called by npLambda</td>
<td>149</td>
</tr>
<tr>
<td>called by npLet</td>
<td>166</td>
</tr>
<tr>
<td>called by npQualifiedDefinition</td>
<td>147</td>
</tr>
<tr>
<td>calls npBackTrack</td>
<td>147</td>
</tr>
<tr>
<td>calls npDef</td>
<td>147</td>
</tr>
<tr>
<td>calls npGives</td>
<td>147</td>
</tr>
<tr>
<td>defun</td>
<td>147</td>
</tr>
<tr>
<td>npDefn</td>
<td>187</td>
</tr>
<tr>
<td>called by npDefinitionItem</td>
<td>168</td>
</tr>
<tr>
<td>called by npPrimary1</td>
<td>164</td>
</tr>
<tr>
<td>calls npDef</td>
<td>187</td>
</tr>
<tr>
<td>calls npEqKey</td>
<td>187</td>
</tr>
<tr>
<td>calls npPP</td>
<td>187</td>
</tr>
<tr>
<td>defun</td>
<td>187</td>
</tr>
<tr>
<td>npDefTail</td>
<td>194</td>
</tr>
<tr>
<td>called by npDef</td>
<td>187</td>
</tr>
<tr>
<td>called by npMdef</td>
<td>165</td>
</tr>
<tr>
<td>called by npSingleRule</td>
<td>194</td>
</tr>
<tr>
<td>calls npDefinitionOrStatement</td>
<td>194</td>
</tr>
<tr>
<td>calls npEqKey</td>
<td>194</td>
</tr>
<tr>
<td>defun</td>
<td>194</td>
</tr>
<tr>
<td>npDiscrim</td>
<td>197</td>
</tr>
<tr>
<td>called by npDisjand</td>
<td>197</td>
</tr>
<tr>
<td>called by npLeftAssoc</td>
<td>197</td>
</tr>
<tr>
<td>calls npQuiver</td>
<td>197</td>
</tr>
<tr>
<td>defun</td>
<td>197</td>
</tr>
<tr>
<td>npDisjand</td>
<td>197</td>
</tr>
<tr>
<td>called by npLogical</td>
<td>197</td>
</tr>
<tr>
<td>called by npDiscrim</td>
<td>197</td>
</tr>
<tr>
<td>calls npLeftAssoc</td>
<td>197</td>
</tr>
<tr>
<td>defun</td>
<td>197</td>
</tr>
<tr>
<td>npDollar</td>
<td>183</td>
</tr>
<tr>
<td>called by npAtom1</td>
<td>183</td>
</tr>
<tr>
<td>calls npEqPeek</td>
<td>183</td>
</tr>
<tr>
<td>calls npNext</td>
<td>184</td>
</tr>
<tr>
<td>calls npPush</td>
<td>183</td>
</tr>
<tr>
<td>calls tokConstruct</td>
<td>183</td>
</tr>
<tr>
<td>calls tokPosn</td>
<td>183</td>
</tr>
<tr>
<td>uses $stok</td>
<td>184</td>
</tr>
<tr>
<td>defun</td>
<td>183</td>
</tr>
<tr>
<td>npDotted</td>
<td>162</td>
</tr>
<tr>
<td>called by npApplication2</td>
<td>163</td>
</tr>
<tr>
<td>called by npApplication</td>
<td>162</td>
</tr>
<tr>
<td>calls ,</td>
<td>162</td>
</tr>
<tr>
<td>defun</td>
<td>162</td>
</tr>
<tr>
<td>npElse</td>
<td>196</td>
</tr>
<tr>
<td>called by npConditional</td>
<td>195</td>
</tr>
<tr>
<td>calls npBacksetElse</td>
<td>196</td>
</tr>
<tr>
<td>calls npPop1</td>
<td>196</td>
</tr>
<tr>
<td>calls npPop2</td>
<td>196</td>
</tr>
<tr>
<td>calls npPop3</td>
<td>196</td>
</tr>
<tr>
<td>calls npPush</td>
<td>196</td>
</tr>
<tr>
<td>calls npRestore</td>
<td>196</td>
</tr>
<tr>
<td>calls npState</td>
<td>196</td>
</tr>
<tr>
<td>calls npTrap</td>
<td>196</td>
</tr>
<tr>
<td>calls pfIfThenOnly</td>
<td>196</td>
</tr>
<tr>
<td>calls pf1f</td>
<td>196</td>
</tr>
<tr>
<td>defun</td>
<td>196</td>
</tr>
<tr>
<td>npEncAp</td>
<td>182</td>
</tr>
<tr>
<td>called by npPrimary1</td>
<td>164</td>
</tr>
<tr>
<td>called by npPrimary2</td>
<td>158</td>
</tr>
<tr>
<td>calls npAnyNo</td>
<td>182</td>
</tr>
<tr>
<td>calls npEncl</td>
<td>182</td>
</tr>
<tr>
<td>calls npFromdom</td>
<td>182</td>
</tr>
<tr>
<td>defun</td>
<td>182</td>
</tr>
<tr>
<td>npEncl</td>
<td>182</td>
</tr>
<tr>
<td>called by npEncAp</td>
<td>182</td>
</tr>
<tr>
<td>calls npBDefinition</td>
<td>182</td>
</tr>
<tr>
<td>calls npPop1</td>
<td>182</td>
</tr>
<tr>
<td>calls npPop2</td>
<td>182</td>
</tr>
<tr>
<td>calls npPush</td>
<td>182</td>
</tr>
<tr>
<td>calls pfApplication</td>
<td>182</td>
</tr>
<tr>
<td>defun</td>
<td>182</td>
</tr>
</tbody>
</table>
npEnclosed, 211
called by npAngleBared, 186
called by npBraced, 186
called by npBracked, 186
called by npParened, 185
calls npEqKey, 211
calls npMissingMate, 211
calls npPop1, 211
calls npPush, 211
calls pfEnSequence, 211
calls pfListOf, 211
calls pfTuple, 211
uses $stok, 211
defun, 211

npEqKey, 145
called by npAdd, 159
called by npAmpersand, 204
called by npAndOr, 181
called by npAssignment, 217
called by npBacksetElse, 197
called by npBreak, 174
called by npCommaBackSet, 146
called by npCompMissing, 151
called by npConditional, 195
called by npDDInfKey, 208
called by npDecl, 214
called by npDefTail, 194
called by npDefaultDecl, 170
called by npDefaultValue, 194
called by npDefn, 187
called by npEnclosed, 211
called by npExport, 171
called by npFix, 166
called by npForIn, 177
called by npFree, 173
called by npFromdom1, 203
called by npFromdom, 202
called by npInfGeneric, 207
called by npInfixOperator, 160
called by npItem1, 142
called by npItem, 142
called by npIterate, 174
called by npLambda, 149
called by npLetQualified, 166
called by npListAndRecover, 189
called by npList, 155
called by npLocalDecl, 172
called by npLocal, 173
called by npLoop, 175
called by npMoveTo, 191
called by npParenthesize, 215
called by npPileBracketed, 188
called by npPileExit, 216
called by npQualified, 147
called by npReturn, 178
called by npRule, 193
called by npSelector, 163
called by npSemiBackSet, 193
called by npSigDecl, 157
called by npSymbolVariable, 205
called by npTypedForm1, 218
called by npTypedForm, 220
called by npTyping, 168
called by npWith, 150
calls npNext, 145
uses $stok, 145
uses $ttok, 145
defun, 145

npEqPeek, 152
called by npAdd, 159
called by npBackTrack, 148
called by npConstTok, 184
called by npDefinitionItem, 167
called by npDollar, 183
called by npInterval, 199
called by npListAndRecover, 189
called by npMoveTo, 191
called by npPrefixColon, 161
called by npSCategory, 153
called by npSegment, 200
called by npWith, 150
uses $stok, 152
uses $ttok, 152
defun, 152

npExit, 215
called by npGives, 148
calls npAssign, 216
calls npBackTrack, 215
calls npPileExit, 216
defun, 215

npExport, 171
called by npStatement, 170
calls npEqKey, 171
calls npLocalItemlist, 171
calls npPop1, 171
calls npPush, 171
calls npTrap, 171
calls pfExport, 171
defun, 171

npExpress, 179
called by npReturn, 178
called by npStatement, 170
calls npExpress1, 179
calls npIterators, 179
calls npPop1, 179
calls npPop2, 179
calls npPush, 179
calls pfCollect, 179
calls pListOf, 179
defun, 179

npExpress1, 179
called by npExpress, 179
calls npADD, 179
calls npConditionalStatement, 179
defun, 179

npFirstTok, 143
called by npNext, 145
called by npParse, 141
called by npRecoverTrap, 190
called by npRestore, 152
calls tokConstruct, 143
calls tokPart, 143
calls tokPosn, 143
uses $inputStream, 143
uses $stok, 143
uses $ttok, 143
defun, 143

npFix, 166
called by npPrimary1, 164
calls npDef, 166
calls npEqKey, 166
calls npPop1, 166
calls npPush, 166
calls pfFix, 166
defun, 166

npForIn, 177
called by npIterators, 175
called by npIterator, 176
calls npBy, 177
calls npCompMissing, 177
calls npEqKey, 177
calls npPop1, 178
calls npPop2, 178
calls npPush, 178
calls npTrap, 177
calls npVariable, 177
calls pfForin, 178
defun, 177

npFree, 173
called by npStatement, 170
calls npEqKey, 173
calls npLocalItemlist, 173
calls npPop1, 173
calls npPush, 173
calls npTrap, 173
calls pfFree, 173
defun, 173

npFromdom, 202
called by npAmpersandFrom, 202
called by npAtom1, 183
called by npAtom2, 160
called by npEncAp, 182
called by npSegment, 200
calls npApplication, 203
calls npEqKey, 202
calls npFromdom1, 203
calls npPop1, 203
calls npPush, 203
calls npTrap, 203
calls pfFromDom, 203
defun, 202

npFromdom1, 203
called by npFromdom1, 203
called by npFromdom, 203
calls npApplication, 203
calls npEqKey, 203
calls npFromdom1, 203
calls npPop1, 203
calls npPush, 203
calls npTrap, 203
calls pfFromDom, 203
defun, 203

npGives, 148
called by npAssignment, 217
called by npDefinitionOrStatement, 147
calls npBackTrack, 148
calls npExit, 148
calls npLambda, 148
defun, 148

npId, 204
called by npName, 204
called by npSymbolVariable, 205
calls npNext, 204
calls npPush, 204
calls tokConstruct, 204
calls tokPosn, 205
uses $npTokToNames, 205
uses $stok, 205
uses $ttok, 205
defun, 204

defun, 204

npImport, 180
called by npDefinitionItem, 167
called by npStatement, 170
calls npAndOr, 180
calls npQualTypelist, 180
calls pfImport, 180
defun, 180

npInfGeneric, 207
called by npLeftAssoc, 207
called by npRightAssoc, 206
called by npTerm, 201
calls npDDInfKey, 207
calls npEqKey, 207
defun, 207

defun, 207

npInfFixOp, 161
called by npInffixOperator, 160
calls npPushId, 161
uses $stok, 161
uses $ttok, 161
defun, 161

defun, 161

npInffixOperator, 160
called by npAtom2, 159
called by npSignatureDefinee, 156
calls npEqKey, 160
calls npInffixOp, 160
calls npPop1, 160
calls npPush, 160
calls npRestore, 160
calls npState, 160
calls pfSymb, 160
calls tokConstruct, 160
calls tokPart, 160
calls tokPosn, 160
uses $stok, 160
defun, 160

defun, 160

npInfKey, 208
called by npDDInfKey, 208
calls npPushId, 208
uses $stok, 208
uses $ttok, 208
defun, 208

npInline, 174
called by npStatement, 170
calls npAndOr, 174
calls npQualTypelist, 174
calls pfInline, 174
defun, 174

defun, 174

npInterval, 199
called by npBy, 199
calls npArith, 199
calls npEqPeek, 199
calls npPop1, 199
calls npPop2, 199
calls npPush, 199
calls npSegment, 199
calls pfApplication, 199
calls pfInfApplication, 199
defun, 199

defun, 199

npItem, 142
called by npParse, 141
calls npEqKey, 142
calls npItem1, 142
calls npPop1, 142
calls npPush, 142
calls npQualDef, 142
calls pfEnSequence, 142
calls pfNovalue, 142
defun, 142

defun, 142

npItem1, 142
called by npItem1, 142
called by npItem, 142
calls npEqKey, 142
calls npItem1, 142
calls npPop1, 143
calls npQualDef, 142
defun, 142
npListAndRecover, 189
  called by npP Penguin, 210
  called by npPileDefinitionList, 189
  calls npEqKey, 189
  calls npEqPeek, 189
  calls npNext, 189
  calls npPop1, 189
  calls npPush, 189
  calls npRecoverTrap, 189
  calls syGeneralErrorHere, 189
  uses $inputStream, 189
  uses $stack, 189
  catches, 189
defun, 189

npListing, 155
  called by npSDefaultItem, 169
  called by npSLocalItem, 172
  called by npSQualifiedTypelist, 181
  called by npSigItemlist, 154
  called by npTypeVariablelist, 157
  called by npVariablelist, 213
calls npList, 155
calls pfListOf, 155
defun, 155

npListofFun, 221
  called by npSemiListing, 193
  called by npTuple, 146
calls npPop1, 221
calls npPop2, 221
calls npPop3, 221
calls npPush, 221
calls npTrap, 221
  uses $stack, 221
defun, 221

npLocal, 173
  called by npStatement, 170
calls npEqKey, 173
calls npLocalItemlist, 173
calls npPop1, 173
calls npPush, 173
calls npTrap, 173
calls pfLocal, 173
defun, 173

npLocalDecl, 172
  called by npLocalItem, 172
calls npEqKey, 172
calls npPop1, 172
calls npPop2, 172
calls npPush, 172
calls npTrap, 172
calls npType, 172
calls pfNothing, 173
calls pfParts, 172
calls pfSpread, 172
defun, 172

npLocalItem, 172
  called by npSLocalItem, 172
calls npLocalDecl, 172
calls npTypeVariable, 172
defun, 172

npLocalItemlist, 171
  called by npExport, 171
  called by npFree, 173
called by npLocal, 173
calls npPC, 171
calls npPop1, 171
calls npPush, 171
calls npSLocalItem, 171
calls pfUnSequence, 171
defun, 171

npLogical, 197
  called by npConditional, 195
  called by npSuchThat, 176
called by npSuch, 150
called by npWhile, 177
calls npDisjand, 197
calls npLeftAssoc, 197
defun, 197

npLoop, 175
  called by npStatement, 170
calls npAssign, 175
calls npCompMissing, 175
calls npEqKey, 175
calls npIterators, 175
calls npPop1, 175
calls npPop2, 175
calls npPush, 175
calls npTrap, 175
calls pfLoop1, 175
calls pfLp, 175
defun, 175

npMacro, 164
called by npDefinitionItem, 167
called by npPrimary1, 164
calls npMdef, 164
calls npPP, 164
defun, 164

npMatch, 150
called by npDef, 187
called by npType, 149
calls npLeftAssoc, 150
calls npSuch, 150
defun, 150

npMDEF, 165
called by npAssign, 216
calls npBackTrack, 165
calls npMDEFinition, 165
calls npStatement, 165
defun, 165

npMdef, 164
called by npMDEFinition, 165
called by npMacro, 164
calls npDefTail, 165
calls npPop1, 165
calls npPush, 165
calls npQuiver, 164
calls npTrap, 165
calls pfCheckMacroOut, 164
calls pfMacro, 165
calls pfPushMacroBody, 165
defun, 164

npMDEFinition, 165
called by npMDEF, 165
calls npMdef, 165
calls npPP, 165
defun, 165

npMissing, 151
called by npCompMissing, 151
called by npConditional, 195
called by npMissingMate, 215
called by npPileBracketed, 215
calls ncSoftError, 188
calls pfCheckMacroOut, 164
calls pname, 151
calls tokPosn, 151
uses $stok, 151
defun, 151
throws, 151

called by npEnclosed, 211
called by npParenthesize, 215
calls ncSoftError, 215
calls npMissing, 215
calls tokPosn, 215
defun, 215

npMoveTo, 191
called by npMoveTo, 191
called by npRecoverTrap, 190
calls npEqKey, 191
calls npEqPeek, 191
calls npMoveTo, 191
calls npNext, 191
uses $inputStream, 191
defun, 191

npName, 204
called by npAmpersand, 204
called by npAtom1, 183
called by npReturn, 178
called by npSignatureDefinee, 156
called by npVariableName, 213
calls npId, 204
calls npSymbolVariable, 204
defun, 204

npNext, 145
called by npConstTok, 184
called by npDollar, 184
called by npEqKey, 145
called by npId, 204
called by npListAndRecover, 189
called by npMoveTo, 191
called by npPrefixColon, 161
called by npPushId, 209
calls npFirstTok, 145
uses $inputStream, 145
defun, 145

npNull, 333
called by eqpileTree, 339
called by insertpile, 335
called by nextline, 112
called by pileForests, 338
called by pilePlusComments, 336
called by pileTree, 337
calls StreamNull, 333
defun, 333

npParened, 185
called by npBracketed, 185
called by npPP, 209
calls npEnclosed, 185
calls pfParen, 185
defun, 185

npParenthesize, 215
called by npParenthesized, 214
calls npEqKey, 215
calls npMissingMate, 215
calls npPush, 215
uses $stok, 215
defun, 215

npParenthesized, 214
called by npPDefinition, 183
called by npTypeVariable, 156
called by npVariable, 213
calls npParenthesize, 214
defun, 214

npParse, 141
called by intloopEchoParse, 67
called by ncloopParse, 36
calls ncSoftError, 141
calls npFirstTok, 141
calls npItem, 141
calls pfDocument, 141
calls pfListOf, 141
calls pfWrong, 141
calls tokPosn, 141
uses $inputStream, 141
uses $stack, 141
uses $stok, 141
uses $ttok, 141
catches, 141
defun, 141

npPC

called by npDefaultItemList, 168
called by npLocalItemList, 171
called by npQualTypelist, 180

defun, 183
npPDefinition, 183
called by npAtom1, 183
called by npBDefinition, 185
calls npDefinitionList, 183
calls npParenthesized, 183
calls npPop1, 183
calls npPush, 183
calls pfEnSequence, 183
calls npEqKey, 188
calls npMissing, 188
calls npPop1, 188
calls npPush, 188
calls pfNothing, 188
calls pfAppend, 188
defun, 188

npPileDefinition, 189
called by npPileDefinition, 188
calls npDefinitionList, 189
calls npListAndRecover, 189
calls npPop1, 189
calls npPush, 189
calls pfAppend, 189
defun, 189

npPileExit, 216
called by npExit, 216
calls npAssign, 216
calls npEqKey, 216
calls npPop1, 216
calls npPop2, 216
calls npPush, 216
calls npStatement, 216
calls pfExit, 216
defun, 216

npPop1, 144
called by npADD, 158
called by npAdd, 159
called by npAndOr, 182
called by npApplication2, 163
called by npApplication, 162
called by npAssignVariable, 217
called by npAssignment, 217
called by npBpDefinition, 188
called by npCategoryL, 152
called by npConstTok, 184
called by np DDInfKey, 208
called by npDecl, 214
called by npDefaultDecl, 170
called by npDefaultItemList, 169
called by npDefaultValue, 195
called by npDefinition, 167
called by npDef, 187
called by npElse, 196
called by npEnclosed, 211
called by npEnc1, 182
called by npExport, 171
called by npExpression, 179
called by npFix, 166
called by npForIn, 178
called by npFree, 173
called by npFromDom1, 203
called by npFromDom, 203
called by npInfixOperator, 160
called by npInterval, 199
called by npItem1, 143
called by npItem, 142
called by npIterators, 175
called by npLambda, 149
called by npLeftAssoc, 207
called by npLetQualified, 167
called by npListAndRecover, 189
called by npListofFun, 221
called by npList, 155
called by npLocalDecl, 172
called by npLocalItemlist, 171
called by npLocal, 173
called by npLoop, 175
called by npMdef, 165
called by npPDefinition, 183
called by npPpf, 210
called by npPpg, 210
called by npPp, 210
called by npPileBracketed, 188
called by npPileDefinitionlist, 189
called by npPileExit, 216
called by npQualDef, 145
called by npQualTypeList, 180
called by npQualType, 181
called by npQualified, 147
called by npReturn, 178
called by npRightAssoc, 206
called by npSCategory, 153
called by npSDefaultItem, 169
called by npSLocalItem, 172
called by npSQualTypeList, 181
called by npSelector, 163
called by npSigDecl, 157
called by npSigItemlist, 154
called by npSignature, 154
called by npSingleRule, 194
called by npSymbolVariable, 205
called by npSynthetic, 198
called by npTerm, 201
called by npTypeVariable, 156
called by npTypedForm1, 218
called by npTypedForm, 220
called by npType, 149
called by npTyping, 168
called by npVariableName, 214
called by npVariable, 213
called by npWConditional, 195
called by npWith, 150
called by npZeroOrMore, 177
uses $stack, 144
defun, 144

npPop2, 144
called by npAdd, 159
called by npApplication2, 163
called by npApplication, 162
called by npAssignment, 217
called by npDecl, 214
called by npDefaultDecl, 170
called by npElse, 196
called by npEncl, 182
called by npExpress, 179
called by npForIn, 178
called by npInterval, 199
called by npIterators, 175
called by npLambda, 149
called by npLeftAssoc, 207
called by npLetQualified, 167
called by npListAndRecover, 189
called by npListofFun, 221
called by npList, 155
called by npLocalDecl, 172
called by npLocal, 173
called by npLoop, 175
called by npMdef, 165
called by npPDefinition, 183
called by npPpf, 210
called by npPpg, 210
called by npPp, 210
called by npPileBracketed, 188
called by npPileDefinitionlist, 189
called by npPileExit, 216
called by npQualDef, 145
called by npQualTypeList, 180
called by npQualType, 181
called by npQualified, 147
called by npReturn, 178
called by npRightAssoc, 206
called by npSCategory, 153
called by npSDefaultItem, 169
called by npSLocalItem, 172
called by npSQualTypeList, 181
called by npSelector, 163
called by npSigDecl, 157
called by npTypedForm1, 218
called by npTypedForm, 220
called by npWith, 150
called by npZeroOrMore, 177
uses $stack, 144
defun, 144

npPop3, 144
called by npElse, 196
called by npListofFun, 221
called by npList, 155
uses $stack, 144
defun, 144

npPower, 202
called by npProduct, 202
calls npColon, 202
calls npRightAssoc, 202
defun, 202

npPP, 209
called by npCategory, 153
called by npDefinition, 167
called by npDefn, 187
called by npMDEFinition, 165
called by npMacro, 164
called by npRule, 193
calls npPPf, 209
calls npPPg, 210
calls npParened, 209
calls npPileBracketed, 210
calls npPop1, 210
calls npPush, 210
calls pfEnSequence, 210
uses npPParg, 210
defun, 209

npPParg, 209
used by npPP, 210
defvar, 209

npPPf, 211
called by npPPg, 210
called by npPP, 209
calls npPPf, 211
calls npSemiListing, 211
defun, 211

npPPf, 210
called by npPPf, 211
calls npPop1, 210
calls npPush, 210

uses $npPParg, 210
defun, 210

npPPg, 210
called by npPP, 210
calls npListAndRecover, 210
calls npPPf, 210
calls npPop1, 210
calls npPush, 210
calls pfAppend, 210
defun, 210

npPrefixColon, 161
called by npAtom2, 160
called by npSignatureDefinee, 156
calls npEqPeek, 161
calls npNext, 161
calls npPush, 161
calls tokConstruct, 161
calls tokPosn, 161
uses $stok, 161
defun, 161

npPretend, 219
called by npTypeStyle, 219
calls npTypedForm, 219
calls pfPretend, 219
defun, 219

npPrimary, 157
called by npApplication, 162
called by npSCategory, 153
called by npSelector, 163
calls npPrimary1, 157
calls npPrimary2, 158
defun, 157

npPrimary1, 164
called by npApplication2, 163
called by npConstTok, 184
called by npPrimary, 157
calls npAtom1, 157
calls npBPileDefinition, 157
calls npDefn, 164
calls npEncAp, 164
calls npFix, 164
calls npLet, 164
calls npMacro, 164
calls npRule, 164
defun, 164

npPrimary2, 158
called by npPrimary, 158
  calls npAdd, 158
  calls npAtom2, 158
  calls npEncAp, 158
  calls npWith, 158
  calls pfNothing, 158
defun, 158

npProcessSynonym, 473
  called by npsynonym, 473
  calls printSynonyms, 473
  calls processSynonymLine, 473
calls putalist, 473
  calls terminateSystemCommand, 473
uses $CommandSynonymAlist, 473
defun, 473

npProduct, 202
  called by npRemainder, 201
calls npLeftAssoc, 202
calls npPower, 202
defun, 202

npPush, 143
  called by npADD, 158
called by npAdd, 159
called by npAndOr, 182
called by npApplication2, 163
called by npApplication, 162
called by npAssignVariable, 217
called by npAssignment, 217
called by npBPileDefinition, 188
called by npBreak, 174
called by npCategoryL, 152
called by npConstTok, 184
called by npDDInfKey, 208
called by npDecl, 214
called by npDefaultDecl, 170
called by npDefaultItemlist, 169
called by npDefaultValue, 195
called by npDefinition, 167
called by npDef, 187
called by npDollar, 183
called by npElse, 196
called by npEnclosed, 211
called by npEnc, 182
called by npExport, 171
called by npExpress, 179
called by npFix, 166
called by npForIn, 178
called by npFree, 173
called by npFromdom1, 203
called by npFromdom, 203
called by npId, 204
called by npInfixOperator, 160
called by npInterval, 199
called by npItem, 142
called by npIterate, 174
called by npIterators, 175
called by npLambda, 149
called by npLeftAssoc, 207
called by npLetQualified, 167
called by npListAndRecover, 189
called by npListofFun, 221
called by npList, 155
called by npLocalDecl, 172
called by npLocalItemList, 171
called by npLocal, 173
called by npLoop, 175
called by npMdef, 165
called by npPDefinition, 183
called by npPPff, 210
called by npPPg, 210
called by npPP, 210
called by npParenthesize, 215
called by npPileBracketed, 188
called by npPileDefinitionlist, 189
called by npPileExit, 216
called by npPrefixColon, 161
called by npQualDef, 145
called by npQualTypelist, 180
called by npQualType, 181
called by npQualified, 147
called by npRecoverTrap, 190
called by npReturn, 178
called by npRightAssoc, 206
called by npSCategory, 153
called by npSDefaultItem, 169
called by npSLocalItem, 172
called by npSQualTypelist, 181
called by npSelector, 163
called by npSigDecl, 157
called by npSigItemlist, 154
called by npSignature, 154
called by npSingleRule, 194
called by npTerm, 201
calls npLeftAssoc, 201
calls npProduct, 201
defun, 201
npRestore, 152
called by npAdd, 159
called by npBackTrack, 148
called by npConstTok, 184
called by npDDInfKey, 208
called by npDefinitionItem, 167
called by npElse, 196
called by npInfixOperator, 160
called by npRightAssoc, 206
called by npSCategory, 153
called by npSymbolAssoc, 205
called by npWith, 150
calls npFirstTok, 152
uses $InputStream, 152
uses $stack, 152
defun, 152
npRestrict, 220
called by npTypeStyle, 219
calls npTypedForm, 220
calls pfRestrict, 220
defun, 220
npReturn, 178
called by npStatement, 170
calls npEqKey, 178
calls npExpress, 178
calls npName, 178
calls npPop1, 178
calls npPop2, 178
calls npPush, 178
calls npTrap, 178
calls pfNothing, 178
calls pfReturnNoName, 178
calls pfReturn, 178
defun, 178
npRightAssoc, 206
called by npPower, 202
called by npQuiver, 198
called by npRightAssoc, 206
calls npInfGeneric, 206
calls npPop1, 206
calls npPop2, 206
calls npPush, 206
calls npRestore, 206
calls npRightAssoc, 206
calls npState, 206
calls pfApplication, 206
calls pfInfApplication, 206
defun, 206
npRule, 193
called by npPrimary1, 164
calls npEqKey, 193
calls npPP, 193
calls npSingleRule, 193
defun, 193
npSCategory, 153
called by npCategory, 153
calls npApplication, 153
calls npCategoryL, 153
calls npDefaultValue, 153
calls npEqPeek, 153
calls npPop1, 153
calls npPrimary, 153
calls npPush, 153
calls npRestore, 153
calls npSignature, 153
calls npState, 153
calls npTrap, 153
calls npWConditional, 153
calls pfAttribute, 153
defun, 153
npSDefaultItem, 169
called by npDefaultItemList, 169
calls npDefaultItem, 169
calls npListing, 169
calls npPop1, 169
calls npPush, 169
calls pfAppend, 169
calls pfParts, 169
defun, 169
npSegment, 200
called by npInterval, 199
calls npEqPeek, 200
calls npFromdom, 200
calls npPushId, 200
defun, 200
npSelector, 163
calls npEqKey, 163
calls npPop1, 163
calls npPop2, 163
calls npPrimary, 163
calls npPush, 163
calls npTrap, 163
calls pfApplication, 163
defun, 163

npSemiBackSet, 193
calledby npSemiListing, 193
calls npEqKey, 193
defun, 193

npSemiListing, 193
calledby npDefinitionlist, 193
calledby npPPf, 211
calls npListoffun, 193
calls npSemiBackSet, 193
calls pfAppend, 193
defun, 193

npSigDecl, 157
calledby npSigItem, 156
calls npEqKey, 157
calls npPop1, 157
calls npPop2, 157
calls npPush, 157
calls npTrap, 157
calls npType, 157
calls pfParts, 157
calls pfSpread, 157
defun, 157

npSigItem, 156
calledby npSigItemlist, 154
calls npEqKey, 157
calls npPop1, 157
calls npPush, 157
calls npTrap, 156
calls npTypeVariable, 156
defun, 156

npSigItemlist, 154
calledby npSignature, 154
calls npListing, 154
calls npPop1, 154
calls npPush, 154
calls npSigItem, 154
calls pfAppend, 154
calls pfListof, 154
calls pfParts, 154
defun, 154

npSignature, 154
calledby npSCategory, 153
calledby npPop1, 154
calls npPush, 154
calls npSigItemlist, 154
calls pfNothing, 154
calls pfWDec, 154
defun, 154

npSignatureDefinee, 156
calledby npTypeVariablelist, 157
calledby npTypeVariable, 156
calls npInfixOperator, 156
calls npName, 156
calls npPrefixColon, 156
defun, 156

npSingleRule, 194
calledby npRule, 193
calls npDefTail, 194
calls npPop1, 194
calls npPop2, 194
calls npPush, 194
calls npQuiver, 194
calls npTrap, 194
calls pfRule, 194
defun, 194

npSLocalItem, 172
calledby npLocalItemlist, 171
calls npListing, 172
calls npLocalItem, 172
calls npPop1, 172
calls npPush, 172
calls pfAppend, 172
calls pfParts, 172
defun, 172

npSQualTypelist, 181
calledby npQualTypelist, 180
calls npListing, 181
calls npPop1, 181
calls npPush, 181
calls npQuiver, 181
calls pfParts, 181
defun, 181

npState, 212
calledby npAdd, 159
calledby npBackTrack, 148
calledby npConstTok, 184
calledby npDDInfKey, 208
calledby npDefinitionItem, 167
called by npElse, 196
called by npInfixOperator, 160
called by npRightAssoc, 206
called by npSCategory, 153
called by npSymbolVariable, 205
called by npWith, 150
uses $inputStream, 212
uses $stack, 212
defun, 212

npStatement, 170
called by npDefinitionItem, 167
called by npMDEF, 165
called by npPileExit, 216
called by npVoid, 179
calls npBreak, 170
calls npExport, 170
calls npExpress, 170
calls npFree, 170
calls npImport, 170
calls npInline, 170
calls npIterate, 170
calls npLocal, 170
calls npLoop, 170
calls npReturn, 170
calls npTyping, 170
calls npVoid, 170
defun, 170

npSuch, 150
called by npMatch, 150
calls npLeftAssoc, 150
calls npLogical, 150
defun, 150

npSuchThat, 176
called by npIterator, 176
calls npAndOr, 176
calls npLogical, 176
calls pfSuchThat, 176
defun, 176

npSum, 201
called by npArith, 200
calls npLeftAssoc, 201
calls npTerm, 201
defun, 201

npSymbolVariable, 205
called by npName, 204
calls npEqKey, 205
calls npId, 205
calls npPop1, 205
calls npPush, 205
calls npRestore, 205
calls npState, 205
calls tokConstruct, 205
calls tokPart, 205
calls tokPosn, 205
defun, 205

defun, 205

defun, 473

called by handleNoParseCommands, 470
calls npProcessSynonym, 473
defun, 473

npSynthetic, 198
called by npRelation, 198
calls npAmpersandFrom, 198
calls npBy, 198
calls npPop1, 198
calls npPop2, 198
calls npPush, 198
calls pfApplication, 198
calls pfInfApplication, 198
defun, 198

npSystem, 472
called by handleNoParseCommands, 470
calls sayKeyedMsg, 472
defun, 472

npTagged, 218
called by npColon, 217
calls npTypedForm1, 218
calls pfTagged, 218
defun, 218

npTerm, 201
called by npSum, 201
calls npInfGeneric, 201
calls npPop1, 201
calls npPop2, 201
calls npPush, 201
calls npRemainder, 201
calls pfApplication, 201
defun, 201

npTrap, 212
called by npAdd, 159
called by npAmpersand, 204
called by npAndOr, 182
called by npAssignment, 217
INDEX

calls npColonQuery, 219
calls npPretend, 219
calls npRestrict, 219
defun, 219

npTypeVariable, 156
called by npDefaultItem, 169
called by npLocalItem, 172
called by npSigItem, 156
calls npParenthesized, 156
calls npPop1, 156
calls npPush, 156
calls npSignatureDefinee, 156
calls npTypeVariablelist, 156
calls pListOf, 156
defun, 156

npTypeVariablelist, 157
called by npTypeVariable, 156
calls npListing, 157
called by npSignatureDefinee, 157
defun, 157

npTypified, 218
called by npColon, 217
calls npAnyNo, 218
calls npApplication, 218
calls npTypeStyle, 218
defun, 218

npTyping, 168
called by npDefinitionItem, 167
called by npStatement, 170
calls npDefaultItemlist, 168
calls npEqKey, 168
calls npPop1, 168
calls npPush, 168
calls npTrap, 168
calls pTyping, 168
defun, 168

npVariable, 213
called by npAdd, 159
called by npForIn, 177
called by npLambda, 148
called by npWith, 150
calls npParenthesized, 213
calls npPop1, 213
calls npPush, 213
calls npVariableName, 213
calls npVariablelist, 213
calls pListOf, 213
defun, 213

npVariablelist, 213
called by npVariable, 213
calls npListing, 213
calls npVariableName, 213
defun, 213

npVariableName, 213
called by npVariablelist, 213
called by npVariable, 213
calls npDecl, 214
calls npName, 213
calls npPop1, 214
calls npPush, 214
calls pNothing, 214
calls pTyped, 214
defun, 213

npVoid, 179
called by npStatement, 170
calls npAndOr, 179
calls npStatement, 179
calls pNothing, 179
defun, 179

npWConditional, 195
called by npSCategory, 153
calls npConditional, 195
calls npPop1, 195
calls npPush, 195
calls pTweakIf, 195
defun, 195

npWhile, 177
called by npIterators, 175
called by npIterator, 176
calls npAndOr, 177
calls npLogical, 177
calls pWhile, 177
defun, 177

npWith, 150
called by npPrimary2, 158
called by npType, 149
calls npCategoryL, 150
calls npCompMissing, 150
calls npEqKey, 150
calls npEqPeek, 150
calls npPop1, 150
calls npPop2, 150
calls npPush, 150
calls npRestore, 150
calls npState, 150
calls npTrap, 150
calls npVariable, 150
calls pfNothing, 150
calls pfWith, 150
defun, 150

npZeroOrMore, 177
calledby npIterators, 175
calls npPop1, 177
calls npPop2, 177
calls npPush, 177
uses $stack, 177
defun, 177

nremove

calledby changeToNamedInterpreterFrame, 560

nreverse0

calledby augmentHasArgs, 1365
calledby bcInputExplicitMatrix, 1186
calledby bcMakeEquations, 1234
calledby bcMakeLinearEquations, 1234
calledby reduceAlistForDomain, 1362
calledby reportOpsForDomain, 1362
calledby reportOpsFromUnitDirectly, 822

NRTevalDomain, 1079
calledby compiledLookup, 1076
calls evalDomain, 1079
calls eval, 1079
calls qcar, 1079
defun, 1079

nsubst

calledby ruleLhsTran, 322
calledby zeroOneTran, 66

obey

calledby copyright, 522
calledby editFile, 545
calledby newHelpSpad2Cmd, 572
calledby summary, 830

object2Identifier

calledby fetchKeyedMsg, 328
calledby fixObjectForPrinting, 456
calledby frameSpad2Cmd, 566
calledby pathnameTypeId, 1041
calledby readHiFi, 601
calledby saveHistory, 596
calledby selectOptionLC, 479
calledby setHistoryCore, 585
calledby writeHiFi, 602

object2String

calledby clearCmdExcept, 505
calledby displayMacro, 454
calledby displaySetOptionInformation, 655
calledby displaySetVariableSettings, 657
calledby getTraceOption, 854
calledby makePathname, 1042
calledby set1, 809
calledby setExposeAddGroup, 699
calledby setLinkerArgs, 728
calledby setNagHost, 755
calledby setOutputAlgebra, 763
calledby setOutputFormula, 790
calledby setOutputFortran, 770
calledby setOutputHtml, 781
calledby setOutputMathml, 776
calledby setOutputOpenMath, 786
calledby setOutputTex, 797
calledby setStreamsCalculate, 801

objMode

calledby displayType, 460
calledby displayValue, 459
calledby getAndEvalConstructorArgument, 958
calledby interpret2, 53
calledby isDomainValuedVariable, 959
calledby printTypeAndTimeNormal, 57
calledby processInteractive1, 50
calledby retract, 1064
calledby showInOut, 600
calledby transTraceItem, 863

objNew

calledby interpret1, 52
calledby interpret2, 53
calledby retract, 1064

objNewWrap

calledby coerceSpadArgs2E, 865
calledby coerceSpadFunValue2E, 868
calledby coerceTraceArgs2E, 864
calledby coerceTraceFunValue2E, 867
INDEX

objValUnwrap
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1185
called by bcInputMatrixByFormula, 1183
called by coerceSpadArgs2E, 865
called by coerceSpadFunValue2E, 868
called by coerceTraceArgs2E, 864
called by coerceTraceFunValue2E, 867
called by displayValue, 459
called by evaluateType1, 918
called by isDomainValuedVariable, 959
called by processInteractive1, 50
called by showInOut, 600

ofCategory, 419
called by evalCategory, 959
called by ofCategory, 419
calls hasCaty, 419
calls identp, 419
calls ofCategory, 419
local def $Subst, 419
local def $hope, 419
defun, 419

oldCompLookup, 1079
called by basicLookup, 1076
calls lookupInDomainVector, 1079
local def $lookupDefaults, 1079
defun, 1079

oldHistFileName, 580
called by initHist, 581
calls makeHistFileName, 580
uses $oldHistoryFileName, 580
defun, 580

om-bindTCP, 1141
defun, 1148
om-getSymbol, 1148
defun, 1148
om-getType, 1148
defun, 1148
om-getVar, 1148
defun, 1148
om-listCDs, 1138
defun, 1138
om-listSymbols, 1138
defun, 1138
om-makeConn, 1140
defun, 1140
om-openFileDev, 1139
defun, 1139
om-openStringDev, 1140
defun, 1140
om-putApp, 1149
defun, 1149
om-putAtp, 1149
defun, 1149
om-putAttr, 1149
defun, 1149
om-putBind, 1149
defun, 1149
om-putBVar, 1150
defun, 1150
om-putByteArray, 1150
defun, 1150
om-putEndApp, 1150
defun, 1150
om-putEndAtp, 1150
defun, 1150
om-putEndAttr, 1151
defun, 1151
om-putEndBind, 1151
defun, 1151
om-putEndBVar, 1151
defun, 1151
om-putEndError, 1151
defun, 1151
om-putEndObject, 1152
defun, 1152
om-putError, 1152
defun, 1152
om-putFloat, 1152
defun, 1152
om-putInt, 1152
defun, 1152
om-putObject, 1153
defun, 1153
om-putObject, 1153
defun, 1153
om-putString, 1153
defun, 1153
om-putSymbol, 1153
defun, 1153
om-putVar, 1153
defun, 1153
om-Read, 1137
defun, 1137
om-setDevEncoding, 1139
defun, 1139
om-stringPtrToString, 1154
defun, 1154
om-stringToStringPtr, 1154
defun, 1154
om-supportsCD, 1138
defun, 1138
om-supportsSymbol, 1138
defun, 1138
openOutputLibrary, 666
called by setOutputLibrary, 664
calls dropInputLibrary, 666
uses input-libraries, 666
uses output-library, 666
defun, 666
openserver, 987
called by restart, 15
defun, 987
operationOpen, 1007
uses *operation-hash*, 1007
uses *operation-stream*, 1007
uses *operation-stream-stamp*, 1007
uses $spadroot, 1007
defun, 1007
operationopen
called by resehashTables, 1000
called by restart0, 16
opIsHasCat
called by basicLookup, 1076
opOf
called by abbreviationsSpad2Cmd, 483
called by augmentHasArgs, 1365
called by conOpPage1, 1375
called by dbAddDocTable, 1383
called by dbSearchOrder, 1356
called by dbSelectCon, 1394
called by dbShowConditions, 1394
called by dbShowCons1, 1389
called by dbShowConsDoc, 1392
called by displaySpad2Cmd, 535
called by domainToSpad2Cmd, 861
called by hasCaty, 420
called by isDomainOrPackage, 875
called by kArgumentCheck, 1371
called by kcPage, 1357
called by kcaPage1, 1363
called by kccPage, 1364
called by kcdePage, 1366
called by kcnPage, 1369
called by kcpPage, 1361
called by kcuPage, 1367
called by kePage, 1351
called by reportOperations, 816
called by spadTrace, 876
optionError, 449
called by readSpad2Cmd, 642
calls commandErrorMessage, 449
defun, 449
optionUserLevelError, 450
calls userLevelErrorMessage, 450
defun, 450
opTran, 323
called by pf2Sex1, 300
called by pfApplication2Sex, 305
uses $dotdot, 323
defun, 323
orderBySlotNumber, 893
calls assocright, 893
calls exit, 893
calls orderList, 893
calls seq, 893
defun, 893
origin
called by inclmsgIfSyntax, 95
called by inclmsgPreliminaryEOF, 86
called by inclmsgPreliminaryFin, 94
originsInOrder, 1382
called by dbDocTable, 1382
called by originsInOrder, 1382
calls ancestorsof, 1382
calls assocleft, 1382
calls getdatabase, 1382
calls insert, 1382
calls originsInOrder, 1382
calls parentsOf, 1382
defun, 1382
output
called by recordAndPrint, 54
output-library, 665
used by openOutputLibrary, 666
defun, 665
outputFormat
called by displayValue, 459
parallel, 399
syntax, 399
parentsOf
called by kcpPage, 1361
called by originsInOrder, 1382
parse-integer
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1185
called by bcInputMatrixByFormula, 1183
parseAndEval, 1294
defun, 1294
parseAndEval1, 1294
defun, 1294
parseAndInterpret, 46
called by executeQuietCommand, 46
called by serverReadLine, 43
uses $interactiveFrame, 46
uses $interactiveMode, 46
uses $boot, 46
uses $e, 46
uses $spad, 46
defun, 46
parseFromString, 46
called by parseSystemCmd, 469
calls StreamNull, 47
calls incString, 47
calls lineoftoks, 47
calls macroExpanded, 47
calls ncloopParse, 47
calls next, 46
calls pf2Sex, 47
defun, 46
parseNoMacroFromString, 1373
calledby mkConform, 1374
calls StreamNull, 1374
calls function, 1373
calls incString, 1374
calls lineoftoks, 1374
calls ncloopParse, 1373
calls next, 1373
calls pf2Sex, 1374
defun, 1373
parseSystemCmd, 469
calledby handleParsedSystemCommands, 468
calls dumbTokenize, 469
calls parseFromString, 469
calls stripSpaces, 469
calls tokTran, 469
defun, 469
parseWord, 1312
defun, 1312
pathname, 1042
calledby deleteFile, 1043
calledby editFile, 545
calledby editSpad2Cmd, 544
calledby getDependentsOfConstructor, 1367
calledby getUsersOfConstructor, 1368
calledby makePathname, 1042
calledby namestring, 1040
calledby pathnameDirectory, 1041
calledby pathnameName, 1040
calledby pathnameType, 1040
calledby pathname, 1042
calledby readSpad2Cmd, 642
calledby reportOpsFromLisplib1, 817
calledby reportOpsFromUnitDirectly1, 825
calledby setExposeAddGroup, 699
calledby setExpose, 697
calledby updateSourceFiles, 546
calledby workfilesSpad2Cmd, 949
calledby make-filename, 1042
calledby pathname, 1042
defun, 1042
pathnameDirectory, 1041
calledby editSpad2Cmd, 544
calledby loadLib, 1035
calledby mergePathnames, 1041
calledby setOutputAlgebra, 763
calledby setOutputFormula, 790
calledby setOutputFortran, 770
calledby setOutputHtml, 781
calledby setOutputMathml, 776
calledby setOutputOpenMath, 786
calledby setOutputTex, 797
calledby pathname, 1041
defun, 1041
pathnameName, 1040
calledby editSpad2Cmd, 544
calledby mergePathnames, 1041
calledby readSpad2Cmd, 642
calledby setOutputAlgebra, 763
calledby setOutputFormula, 790
calledby setOutputFortran, 770
calledby setOutputHtml, 781
calledby setOutputMathml, 776
calledby setOutputOpenMath, 786
calledby setOutputTex, 797
calledby updateSourceFiles, 546
calls pathname, 1040
defun, 1040
PathnameString

calledby editSpad2Cmd, 544
calledby mergePathnames, 1041
calledby pathnameDirectory, 1041
calledby pathnameType, 1041
calledby pathname, 1042
calledby setOutputAlgebra, 763
calledby setOutputFormula, 790
calledby setOutputFortran, 770
calledby setOutputHtml, 781
calledby setOutputMathml, 776
calledby setOutputOpenMath, 786
calledby setOutputTex, 797
calledby updateSourceFiles, 546
calls pathname, 1040
defun, 1040

calledby pfname, 89
pathnameType, 1040
pathnameTypeId, 1041
called by readSpad2Cmd, 642
called by updateSourceFiles, 546
calls object2Identifier, 1041
calls pathnameType, 1041
calls upcase, 1041
defun, 1041

patternVarsOf, 321
called by ruleLhsTran, 322
called by rulePredicateTran, 319
calls patternVarsOf1, 321
defun, 321

patternVarsOf1, 321
called by patternVarsOf, 321
called by patternVarsOf, 321
calls patternVarsOf1, 321
defun, 321

pcounters, 860
called by trace1, 848
calls bright, 860
calls concat, 860
calls sayBrightly, 860
uses /countlist, 860
defun, 860

peekTimedName
called by interpretTopLevel, 51

pf0ApplicationArgs, 237
called by macApplication, 223
calls pf0FlattenSyntacticTuple, 237
calls pfApplicationArg, 237
defun, 237

pf0AssignLhsItems, 256
called by pf2Sex1, 301
calls pfAssignLhsItems, 256
calls pfParts, 256
defun, 256

pf0DefinitionLhsItems, 262
called by pfDefinition2Sex, 315
calls pfDefinitionLhsItems, 262
calls pfParts, 262
defun, 262

pf0FlattenSyntacticTuple, 237
called by pf0ApplicationArgs, 237
called by pf0FlattenSyntacticTuple, 237
calls pf0FlattenSyntacticTuple, 237
calls pf0TupleParts, 237
calls pfTuple?, 237
defun, 237

pf0ForinLhs, 267
called by pf2Sex1, 300
calls pfForinLhs, 267
calls pfParts, 267
defun, 267

pf0FreeItems, 266
called by pf2Sex1, 301
calls pfFreeItems, 266
calls pfParts, 266
defun, 266

pf0LambdaArgs, 274
called by macLambdaParameterHandling, 231
called by pfLambdaTran, 316
calls pfLambdaArgs, 274
calls pfParts, 274
defun, 274

pf0LocalItems, 275
called by pf2Sex1, 301
calls pfLocalItems, 275
calls pfParts, 275
defun, 275

pf0LoopIterators, 276
called by pf0LoopIterators, 276
called by pf2Sex1, 300
calls pf0LoopIterators, 276
calls pfParts, 276
defun, 276

pf0MLambdaArgs, 278
called by mac0MLambdaApply, 223
called by macLambdaParameterHandling, 231
calls pfParts, 278
defun, 278

pf0SequenceArgs, 287
called by pfSequence2Sex, 310
called by pfUnSequence, 293
calls pfParts, 287
calls pfSequenceArgs, 287
defun, 287

pf0TupleParts, 293
called by pf0FlattenSyntacticTuple, 237
called by pf2Sex1, 300
called by pfApplication2Sex, 306
called by pfCheckArg, 241
called by pfCheckItOut, 239
called by pfCollectVariable1, 242
called by pfSexpr, strip, 250
called by pfSuchThat2Sex, 307
called by pfTransformArg, 244
calls pfParts, 293
calls pfTupleParts, 293
defun, 293
pf0WhereContext, 295
called by pf2Sex1, 301
calls pfParts, 295
calls pfWhereContext, 295
defun, 295
pf2Sex, 290
called by intInterpretPform, 66
called by parseFromString, 47
called by parseNoMacroFromString, 1374
called by pfApplication2Sex, 306
called by pfSuchThat2Sex, 307
calls pf2Sex1, 299
uses $QuietCommand, 299
uses $insideApplication, 299
uses $insideRule, 299
uses $insideSEQ, 299
defun, 299
pf2Sex1, 300
called by loopIters2Sex, 312
called by pf2Sex1, 300
called by pf2Sex, 299
called by pfApplication2Sex, 306
called by pfCollect2Sex, 314
called by pfDefinition2Sex, 315
called by pfLambdaTran, 316
called by pfLhsRule2Sex, 318
called by pfOp2Sex, 308
called by pfRhsRule2Sex, 319
called by pfSequence2Sex, 310
called by pfSuchThat2Sex, 307
calls keyedSystemError, 301
calls loopIters2Sex, 300
calls opTran, 300
calls pf0AssignLhsItems, 301
calls pf0ForinLhs, 300
calls pf0FreeItems, 301
calls pf0LocalItems, 301
calls pf0LoopIterators, 300
calls pf0TupleParts, 300
calls pf0WhereContext, 301
calls pf2Sex1, 300
calls pfAbSynOp, 302
calls pfAnd?, 301
calls pfAndLeft, 301
calls pfAndRight, 301
calls pfApplication2Sex, 300
calls pfApplication?, 300
calls pfAssign?, 301
calls pfAssignRhs, 301
calls pfBreak?, 301
calls pfBreakFrom, 301
calls pfCoerceto?, 300
calls pfCoercetoExpr, 300
calls pfCoercetoType, 300
calls pfCollect2Sex, 300
calls pfCollect?, 300
calls pfDefinition2Sex, 301
calls pfDefinition?, 301
calls pfDo?, 301
calls pfDoBody, 301
calls pfExit?, 300
calls pfExitCond, 300
calls pfExitExpr, 300
calls pfForin?, 300
calls pfForinWhole, 300
calls pfFree?, 301
calls pfFromdom?, 300
calls pfFromdomDomain, 300
calls pfFromdomWhat, 300
calls pfIdSymbol, 300
calls pfIf?, 300
calls pfIfCond, 300
calls pfIfElse, 300
calls pfIfThen, 300
calls pfIterate?, 301
calls pfLambda2Sex, 301
calls pfLambda?, 301
calls pfLiteral2Sex, 300
calls pfLiteral?, 300
calls pfLocal?, 301
calls pfLoop?, 300
calls pfMLambda?, 301
calls pfMacro?, 301
<table>
<thead>
<tr>
<th>Function</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls pfNot?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfNotArg,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfNothing?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfNoValue?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfNoValueExpr,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfOr?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfOrLeft,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfOrRight,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfPretend?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfPretendExpr,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfRestrict?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfRestrictExpr,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfRestrictType,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfReturn?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfReturnExpr,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfRule2Sex,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfRule?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfSequence2Sex,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfSequence?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfSuchthat?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfSuchthatCond,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfSymbol?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfSymbolSymbol,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfTagged?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfTaggedExpr,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfTaggedTag,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfTuple?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfTyped?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfTypedId,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfTypedType,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfWhere?,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfWhereExpr,</td>
<td>301</td>
</tr>
<tr>
<td>calls pfWhile?,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfWhileCond,</td>
<td>300</td>
</tr>
<tr>
<td>calls pfWrong?,</td>
<td>301</td>
</tr>
<tr>
<td>calls spadThrow,</td>
<td>301</td>
</tr>
<tr>
<td>calls tokPart,</td>
<td>302</td>
</tr>
<tr>
<td>uses $QuietCommand,</td>
<td>302</td>
</tr>
<tr>
<td>uses $insideRule,</td>
<td>302</td>
</tr>
<tr>
<td>uses $insideSEQ,</td>
<td>302</td>
</tr>
<tr>
<td>defun</td>
<td>231</td>
</tr>
<tr>
<td>calledby pf2Sex1,</td>
<td>302</td>
</tr>
<tr>
<td>calledby pfCopyWithPos,</td>
<td>236</td>
</tr>
<tr>
<td>calledby pfLeaf?,</td>
<td>247</td>
</tr>
<tr>
<td>calledby pfLiteral?,</td>
<td>248</td>
</tr>
<tr>
<td>calledby pfLiteralClass,</td>
<td>248</td>
</tr>
<tr>
<td>calledby pfMapParts,</td>
<td>237</td>
</tr>
<tr>
<td>calledby pfSexpr.strip,</td>
<td>250</td>
</tr>
<tr>
<td>called ifcar,</td>
<td>412</td>
</tr>
<tr>
<td>defun</td>
<td>412</td>
</tr>
<tr>
<td>called by intloopProcess,</td>
<td>62</td>
</tr>
<tr>
<td>called by pfAnd?,</td>
<td>254</td>
</tr>
<tr>
<td>called by pfApplication?,</td>
<td>255</td>
</tr>
<tr>
<td>called by pfAssign?,</td>
<td>256</td>
</tr>
<tr>
<td>called by pfBreak?,</td>
<td>258</td>
</tr>
<tr>
<td>called by pfCoerce,</td>
<td>259</td>
</tr>
<tr>
<td>called by pfCollect?,</td>
<td>260</td>
</tr>
<tr>
<td>called by pfDefinition?,</td>
<td>261</td>
</tr>
<tr>
<td>called by pfDo?,</td>
<td>262</td>
</tr>
<tr>
<td>called by pfExit?,</td>
<td>263</td>
</tr>
<tr>
<td>called by pfForIn?,</td>
<td>266</td>
</tr>
<tr>
<td>called by pfFree?,</td>
<td>265</td>
</tr>
<tr>
<td>called by pfFromdom?,</td>
<td>268</td>
</tr>
<tr>
<td>called by pfIfid?,</td>
<td>246</td>
</tr>
<tr>
<td>called by pff?,</td>
<td>269</td>
</tr>
<tr>
<td>called by pfIterate?,</td>
<td>271</td>
</tr>
<tr>
<td>called by pfLambda?,</td>
<td>273</td>
</tr>
<tr>
<td>called by pfLam,</td>
<td>272</td>
</tr>
<tr>
<td>called by pfLocal?,</td>
<td>274</td>
</tr>
<tr>
<td>called by pfLoop?,</td>
<td>276</td>
</tr>
<tr>
<td>called by pfMLambda?,</td>
<td>278</td>
</tr>
<tr>
<td>called by pfMacro?,</td>
<td>277</td>
</tr>
<tr>
<td>called by pfNot?,</td>
<td>279</td>
</tr>
<tr>
<td>called by pfNothing?,</td>
<td>245</td>
</tr>
<tr>
<td>called by pfNoValue?,</td>
<td>280</td>
</tr>
<tr>
<td>called by pfOr?,</td>
<td>280</td>
</tr>
<tr>
<td>called by pfPretend?,</td>
<td>282</td>
</tr>
<tr>
<td>called by pfRestrict?,</td>
<td>283</td>
</tr>
<tr>
<td>called by pfReturn?,</td>
<td>284</td>
</tr>
<tr>
<td>called by pfRule?,</td>
<td>286</td>
</tr>
<tr>
<td>called by pfSequence?,</td>
<td>287</td>
</tr>
<tr>
<td>called by pfSuchthat?,</td>
<td>288</td>
</tr>
<tr>
<td>called by pfSymbol?,</td>
<td>252</td>
</tr>
<tr>
<td>called by pfTagged?,</td>
<td>289</td>
</tr>
</tbody>
</table>

pfAbSynOp?, 412
called by intloopProcess, 62
called by pfAnd?, 254
called by pfApplication?, 255
called by pfAssign?, 256
called by pfBreak?, 258
called by pfCoerce?, 259
called by pfCollect?, 260
called by pfDefinition?, 261
called by pfDo?, 262
called by pfExit?, 263
called by pfForIn?, 266
called by pfFree?, 265
called by pfFromdom?, 268
called by pfIfid?, 246
called by pff?, 269
called by pfIterate?, 271
called by pfLambda?, 273
called by pfLam, 272
called by pfLocal?, 274
called by pfLoop?, 276
called by pfMLambda?, 278
called by pfMacro?, 277
called by pfNot?, 279
called by pfNothing?, 245
called by pfNoValue?, 280
called by pfOr?, 280
called by pfPretend?, 282
called by pfRestrict?, 283
called by pfReturn?, 284
called by pfRule?, 286
called by pfSequence?, 287
called by pfSuchthat?, 288
called by pfSymbol?, 252
called by pfTagged?, 289
called by pfTuple?, 292
called by pfTyped?, 291
called by pfWhere?, 294
called by pfWhile?, 296
called by pfWrong?, 297
calls eqcar, 412
defun, 412

pfAdd, 252
called by npAdd, 159
called by npDefaultValue, 195
calls pfNothing, 252
calls pfTree, 252
defun, 252

pfAnd, 253
called by pfInfApplication, 271
calls pfTree, 253
defun, 253

pfAnd?, 254
called by pf2Sex1, 301
calls pfAbSynOp?, 254
defun, 254

pfAndLeft, 254
called by pf2Sex1, 301
defun, 254

pfAndRight, 255
called by pf2Sex1, 301
defun, 255

pfAppend, 255
called by npPPg, 210
called by npPileDefinitionlist, 189
called by npSDefaultItem, 169
called by npSLocalItem, 172
called by npSemiListing, 193
called by npSigItemlist, 154
called by pfUnSequence, 293
defun, 255

pfApplication, 253
called by npApplication2, 163
called by npApplication, 162
called by npEncl, 182
called by npInterval, 199
called by npLeftAssoc, 207
called by npRightAssoc, 206
called by npSelector, 163
called by npSynthetic, 198
called by npTerm, 201
called by pfBraceBar, 257
called by pfBrace, 257
called by pfBracketBar, 257
called by pfBracket, 257
called by pfFix, 265
called by pfFromDom, 267
called by pfInfApplication, 271
calls pfTree, 253
defun, 253

pfApplication2Sex, 305
called by pf2Sex1, 300
calls hasOptArgs?, 306
calls opTran, 305
calls pf0TupleParts, 306
calls pf2Sex1, 306
calls pf2Sex, 306
calls pfApplicationArg, 306
calls pfApplicationOp, 305
calls pfOp2Sex, 305
calls pfSuchThat2Sex, 306
calls pfTuple?, 306
uses $insideApplication, 306
uses $insideRule, 306
defun, 305

pfApplication?, 255
called by macExpand, 222
called by pf2Sex1, 300
called by pfCheckItOut, 239
called by pfCheckMacroOut, 240
called by pfCollect1?, 242
called by pfFlattenApp, 241
called by pfFromDom, 267
called by pfSexpr,strip, 250
calls pfAbSynOp?, 255
defun, 255

pfApplicationArg, 254
called by pf0ApplicationArgs, 237
called by pfApplicationArg, 237
called by pfCollectVariable1, 242
called by pfFlattenApp, 242
called by pfFromDom, 267
called by pfSexpr,strip, 250
defun, 254

pfApplicationOp, 254
called by macApplication, 223
called by pfApplication2Sex, 305
INDEX

called by pfCollect1?, 242
called by pfFlattenApp, 242
called by pfFromDom, 267
called by pfSexpr, strip, 250
defun, 254

pfAssign, 255
called by npAssignment, 217
calls pfTree, 255
defun, 255

pfAssign?, 256
called by pf2Sex1, 301
calls pfAbSynOp?, 256
defun, 256

pfAssignLhsItems, 256
called by pf0AssignLhsItems, 256
defun, 256

pfAssignRhs, 256
called by pf2Sex1, 301
defun, 256

pfAttribute, 253
called by npSCategory, 153
calls pfTree, 253
defun, 253

pfBrace, 257
called by npBraced, 186
calls pfApplication, 257
calls pfIdPos, 257
calls tokPosn, 257
defun, 257

pfBraceBar, 257
called by npBraced, 186
calls pfApplication, 257
calls pfIdPos, 257
calls tokPosn, 257
defun, 257

pfBracket, 257
called by npBracked, 186
calls pfApplication, 257
calls pfIdPos, 257
calls tokPosn, 257
defun, 257

pfBracketBar, 257
called by npBracked, 186
calls pfApplication, 257
calls pfIdPos, 258
calls tokPosn, 258
defun, 257
calls pfTransformArg, 239
calls pfTuple?, 239
calls pfTyped, 239
defun, 239
pfCheckMacroOut, 240
called by npMdef, 164
calls npTrapForm, 240
calls pfApplication?, 240
calls pfCheckArg, 240
calls pfCheckId, 240
calls pfFlattenApp, 240
calls pfId?, 240
defun, 240
pfCoerceto, 259
called by npCoerceTo, 220
calls pfTree, 259
defun, 259
pfCoerceto?, 259
called by pf2Sex1, 300
calls pfAbSynOp?, 259
defun, 259
pfCoercetoExpr, 259
called by pf2Sex1, 300
defun, 259
pfCoercetoType, 259
called by pf2Sex1, 300
defun, 259
pfCollect, 260
called by npExpress, 179
calls pfTree, 260
defun, 260
pfCollect1?, 242
called by pfCheckItOut, 239
called by pfFlattenApp, 242
called by pfTaggedToTyped1, 244
calls pfApplication?, 242
calls pfApplicationOp, 242
calls pfId?, 242
calls pfIdSymbol, 242
defun, 242
pfCollect2Sex, 314
called by pf2Sex1, 300
calls loopIters2Sex, 314
calls pf2Sex1, 314
calls pfCollectBody, 314
calls pfCollectIterators, 314
calls pfParts, 314
defun, 314
pfCollect?, 260
called by pf2Sex1, 300
called by pfCollectArgTran, 317
calls pfAbSynOp?, 260
defun, 260
pfCollectArgTran, 317
called by pfLambdaTran, 316
calls pf2Sex1, 317
calls pfCollect?, 317
calls pfCollectBody, 317
calls pfCollectIterators, 317
calls pfParts, 317
defun, 317
pfCollectBody, 260
called by pfCollect2Sex, 314
called by pfCollectArgTran, 317
defun, 260
pfCollectIterators, 260
called by pfCollect2Sex, 314
called by pfCollectArgTran, 317
defun, 260
pfCollectVariable1, 242
called by pfCheckItOut, 239
called by pfTaggedToTyped1, 244
calls pf0TupleParts, 244
calls pfApplicationArg, 242
calls pfSuch, 242
calls pfTaggedToTyped, 242
calls pfTypedId, 242
calls pfTypedType, 243
calls pfTyped, 242
defun, 242
pfCopyWithPos, 236
called by macId, 227
called by pfCopyWithPos, 236
calls pfAbSynOp, 236
calls pfCopyWithPos, 236
calls pfLeaf?, 236
calls pfLeaf, 236
calls pfParts, 236
calls pfTree, 236
calls tokPart, 236
defun, 236
pfDefinition, 261
INDEX

called by npDef, 187
calls pfTree, 261
defun, 261

pfDefinition2Sex, 315
called by pf2Sex1, 301
calls pf0DefinitionLhsItems, 315
calls pf2Sex1, 315
calls pfDefinitionRhs, 315
calls pLambdaTran, 315
calls systemError, 315
uses $insideApplication, 315
defun, 315

pfDefinition?, 261
called by pf2Sex1, 301
called by pfCheckItOut, 239
called by pfTaggedToTyped1, 244
calls pfAbSynOp?, 261
defun, 261

pfDefinitionLhsItems, 261
called by pf0DefinitionLhsItems, 262
defun, 261

pfDefinitionRhs, 261
called by pfDefinition2Sex, 315
defun, 261

pfDo, 262
called by pfLoop1, 275
called by pfLp, 276
calls pfTree, 262
defun, 262

pfDo?, 262
called by pf2Sex1, 301
calls pfAbSynOp?, 262
defun, 262

pfDoBody, 262
called by pf2Sex1, 301
defun, 262

pfDocument, 246
called by npParse, 141
called by npRecoverTrap, 190
calls pfLeaf, 246
defun, 246

pfEnSequence, 263
called by npEnclosed, 211
called by npItem, 142
called by npPDefinition, 183
called by npPP, 210
calls pfListOf, 263
calls pfSequence, 263
calls pfTuple, 263
defun, 263

pfExit, 263
called by npPileExit, 216
calls pfTree, 263
defun, 263

pfExit?, 263
called by pf2Sex1, 300
calls pfAbSynOp?, 263
defun, 263

pfExitCond, 263
called by pf2Sex1, 300
defun, 263

pfExitExpr, 264
called by pf2Sex1, 300
defun, 264

pfExport, 264
called by npExport, 171
calls pfTree, 264
defun, 264

pfExpression, 264
called by pfSymb, 251
calls ifcar, 264
calls pfLeaf, 264
defun, 264

pfFileName, 236
called by ppos, 357
calls poFileName, 236
defun, 236

pfFirst, 264
called by pfLam, 272
called by pfSourceStok, 243
defun, 264

pfFix, 265
called by npFix, 166
calls pfApplication, 265
calls pfId, 265
defun, 265

pfFlattenApp, 241
called by pfCheckItOut, 239
called by pfCheckMacroOut, 240
called by pfFlattenApp, 242
calls pfApplication?, 241
calls pfApplicationArg, 242
calls pfApplicationOp, 242
calls pfCollect1?, 242
calls pfFlattenApp, 242
defun, 241
pfForin, 266
called by npForIn, 178
calls pfTree, 266
defun, 266
pfForin?, 266
called by pf2Sex1, 300
calls pfAbSynOp?, 266
defun, 266
pfForinLhs, 267
called by pf0ForinLhs, 267
defun, 267
pfForinWhole, 267
called by pf2Sex1, 300
defun, 267
pfFree, 265
called by npFree, 173
calls pfTree, 265
defun, 265
pfFree?, 265
called by pf2Sex1, 301
calls pfAbSynOp?, 265
defun, 265
pfFreeItems, 266
called by pf0FreeItems, 266
defun, 266
pfFromDom, 267
called by npFromdom1, 203
called by npFromdom, 203
calls pfApplication?, 267
calls pfApplicationArg, 267
calls pfApplicationOp, 267
calls pfApplication, 267
calls pfFromdom, 267
defun, 267
pfFromdom, 268
called by pfFromDom, 267
calls pfTree, 268
defun, 268
pfFromdom?, 268
called by pf2Sex1, 300
calls pfAbSynOp?, 268
defun, 268
pfFromdomDomain, 269
called by pf2Sex1, 300
defun, 269
pfFromdomWhat, 268
called by pf2Sex1, 300
defun, 268
pfGlobalLinePosn, 235
called by syIgnoredFromTo, 191
calls poGlobalLinePosn, 235
defun, 235
pfHide, 269
called by npAngleBared, 186
calls pfTree, 269
defun, 269
pfId, 246
called by pfFix, 265
called by pfSuch, 244
called by pfTaggedToTyped, 289
calls pfLeaf, 246
defun, 246
pfId?, 246
called by mac0MLambdaApply, 224
called by mac0SubstituteOuter, 231
called by macExpand, 222
called by macMacro, 229
called by pfCheckId, 241
called by pfCheckItOut, 239
called by pfCheckMacroOut, 240
called by pfCollect1?, 242
called by pfSexr,strip, 250
called by pfTaggedToTyped, 289
calls pfAbSynOp?, 246
defun, 246
pfIdPos, 246
called by pfBraceBar, 257
called by pfBrace, 257
called by pfBracketBar, 258
called by pfBracket, 257
calls pfLeaf, 246
defun, 246
pfIdSymbol, 247
called by macId, 227
called by macLambdaParameterHandling, 231
called by macMacro, 229
called by macSubstituteId, 232
INDEX

called by pf2Sex1, 300
called by pfCollect1?, 242
called by pfInfApplication, 271
called by pfSexpr, strip, 250
calls tokPart, 247
defun, 247

pfIf, 269
called by npElse, 196
called by pfIfThenOnly, 270
calls pfTree, 269
defun, 269

pfIf?, 269
called by pf2Sex1, 300
calls pfAbSynOp?, 269
defun, 269

pfIfCond, 270
called by pf2Sex1, 300
called by pfTweakIf, 290
defun, 270

pfIfElse, 270
called by pf2Sex1, 300
called by pfTweakIf, 290
defun, 270

pfIfThen, 270
called by pf2Sex1, 300
called by pfTweakIf, 290
defun, 270

pfIfThenOnly, 270
called by npElse, 196
calls pfIf, 270
calls pfTree, 270
defun, 270
calls pfNothing, 270
defun, 270

called by ppos, 357

called by pfImport, 271
called by npImport, 180
calls pfTree, 271
defun, 271

pfInfApplication, 271
called by npInterval, 199
called by npLeftAssoc, 207
called by npRightAssoc, 206
called by npSynthetic, 198
called by pfSuch, 244
called by pfTaggedToTyped, 290
calls pfAnd, 271
calls pfApplication, 271
calls pfIdSymbol, 271
calls pfListOf, 271
calls pfOr, 271
calls pfTuple, 271
defun, 271

called by npInline, 174
calls pfTree, 272
defun, 272

pfIterate, 271
called by npIterate, 174
calls pfTree, 271
defun, 271

pfIterate?, 271
called by pf2Sex1, 301
calls pfAbSynOp?, 271
defun, 271

pfLam, 272
called by npLambda, 149
calls pfAbSynOp?, 272
calls pfFirst, 272
calls pfLambda, 272
calls pfNothing, 272
calls pfSecond, 272
defun, 272

pfLambda, 273
called by pfLam, 272
called by pfPushBody, 249
calls pfTree, 273
defun, 273

pfLambda2Sex, 317
called by pf2Sex1, 301
calls pfLambdaTran, 317
defun, 317

pfLambda?, 273
called by mac0SubstituteOuter, 231
called by macExpand, 222
called by macLambdaParameterHandling, 231
called by pf2Sex1, 301
called by pfLambdaTran, 316
calls pfAbSynOp?, 273
defun, 273

pfLambdaArgs, 274
called by pf0LambdaArgs, 274
calls float2Sex, 304
  calls keyedSystemError, 304
  calls pfLeafToken, 304
  calls pfLiteralClass, 304
  calls pfLiteralString, 304
  calls pfSymbolSymbol, 304
  uses $insideRule, 304
defun, 304
pfLiteral?, 248
called by pf2Sex1, 300
called by pfSexr, strip, 250
calls pfAbSynOp, 248
defun, 248
pfLiteralClass, 248
called by pfLiteral2Sex, 304
calls pfAbSynOp, 248
defun, 248
pfLiteralString, 249
called by pfLiteral2Sex, 304
called by pfSexr, strip, 250
calls tokPart, 249
defun, 249
pfLocal, 274
called by npLocal, 173
calls pfTree, 274
defun, 274
pfLocal?, 274
called by pf2Sex1, 301
called by pfAbSynOp?, 274
defun, 274
pfLocalItems, 275
called by pf0LocalItems, 275
defun, 275
pfLoop, 275
called by pfLoop1, 275
called by pfLp, 276
calls pfTree, 275
defun, 275
pfLoop1, 275
called by npLoop, 175
calls pfDo, 275
calls pfListOf, 275
calls pfLoop, 275
defun, 275
pfLoop?, 276
called by pf2Sex1, 300
calls pfAbSynOp?, 276
defun, 276
pfLoopIterators, 276
defun, 276
pfLp, 276
called by npLoop, 175
calls pfDo, 276
calls pfListOf, 276
calls pfLoop, 276
defun, 276
pfMacro, 277
called by macMacro, 229
called by npMdef, 165
calls pfTree, 277
defun, 277
pfMacro?, 277
called by macExpand, 222
called by pf2Sex1, 301
calls pfAbSynOp?, 277
defun, 277
pfMacroLhs, 277
called by macMacro, 229
defun, 277
pfMacroRhs, 277
called by macMacro, 229
defun, 277
pfMapParts, 236
called by macApplication, 223
called by macExpand, 222
called by macLambda,mac, 229
called by macWhere,mac, 228
calls pfAbSynOp, 237
calls pfLeaf?, 236
calls pfParts, 236
calls pfTree, 237
defun, 236
pfMLambda, 278
called by pfPushMacroBody, 243
calls pfTree, 278
defun, 278
pfMLambda?, 278
called by macApplication, 223
called by macLambdaParameterHandling, 231
called by macMacro, 229
called by pf2Sex1, 301
calls pfAbSynOp?, 278
  defun, 278
pfMLambdaArgs, 278
defun, 278
pfMLambdaBody, 279
calledby mac0GetName, 226
calledby mac0MLambdaApply, 223
defun, 279
pfname, 89
calledby thefname, 89
calls PathnameString, 89
defun, 89
pfNoPosition, 414
calledby pfLeaf, 247
calledby tokPosn, 413
calls poNoPosition, 414
defun, 414
pfNoPosition?, 412
calledby ppos, 357
calledby tokConstruct, 411
calls poNoPosition?, 412
defun, 412
pfNot?, 279
calledby pf2Sex1, 301
calls pfAbSynOp?, 279
defun, 279
pfNotArg, 279
calledby pf2Sex1, 301
defun, 279
pfNothing, 245
calledby macMacro, 229
calledby npAdd, 159
calledby npBreak, 174
calledby npDefaultValue, 195
calledby npIterate, 174
calledby npLocalDecl, 173
calledby npPileBracketed, 188
calledby npPrimary2, 158
calledby npQualType, 181
calledby npReturn, 178
calledby npSignature, 154
calledby npVariableName, 214
calledby npWith, 150
called by pfAdd, 252
called by pfCheckItOut, 239
called by pfIfThenOnly, 270
called by pfLam, 272
called by pfPushBody, 249
called by pfReturnNoName, 285
called by pfTaggedToTyped1, 244
called by pfTaggedToTyped, 289
calls pfTree, 245
defun, 245
pfNothing?, 245
called by macMacro, 229
called by pf2Sex1, 300
called by pfLambdaTran, 316
called by pfTweakIf, 290
calls pfAbSynOp?, 245
defun, 245
pfNoValue, 279
called by npItem, 142
called by npVoid, 179
calls pfTree, 279
defun, 279
pfNoValue?, 280
called by pf2Sex1, 301
calls pfAbSynOp?, 280
defun, 280
pfNoValueExpr, 280
called by pf2Sex1, 301
defun, 280
pfOp2Sex, 308
called by pfApplication2Sex, 305
calls pf2Sex1, 308
calls pfSymbol?, 308
calls pmDontQuote?, 308
uses $insideRule, 308
uses $quotedOpList, 308
defun, 308
pfOr, 280
called by pfInfApplication, 271
calls pfTree, 280
defun, 280
pfOr?, 280
called by pf2Sex1, 301
calls pfAbSynOp?, 280
defun, 280
pfOrLeft, 281
called by pf2Sex1, 301
defun, 281
INDEX

called by mac0InfiniteExpansion, 225
called by mac0MLambdaApply, 224
called by macMacro, 229
called by phMacro, 221
pfOrRight, 281
called by pf2Sex1, 301
defun, 281

pfParen, 281
called by npParened, 185
defun, 281

pfParts, 249
called by mac0SubstituteOuter, 231
called by macLambdaParameterHandling, 231
called by npDefaultDecl, 170
called by npLocalDecl, 172
called by npSDefaultItem, 169
called by npSLocalItem, 172
called by npSQualTypelist, 181
called by npSigDecl, 157
called by npSigItemlist, 154
called by pf0AssignLhsItems, 256
called by pf0DefinitionLhsItems, 262
called by pf0ForinLhs, 267
called by pf0FreeItems, 266
called by pf0LambdaArgs, 274
called by pf0LocalItems, 275
called by pf0LoopIterators, 276
called by pf0MLambdaArgs, 278
called by pf0SequenceArgs, 287
called by pf0TupleParts, 293
called by pf0WhereContext, 295
called by pfCollect2Sex, 314
called by pfCollectArgTran, 317
called by pfCopyWithPos, 236
called by pfMapParts, 236
called by pfSexpr,strip, 250
called by pfSourcePosition, 238
called by pfSourceStok, 243
called by pfWDec, 294
defun, 249

pfPile, 249
called by npPileBracketed, 188
defun, 249

pfPretend, 281
called by npPretend, 219
calls pfTree, 281
defun, 281

pfPretend?, 282
called by pf2Sex1, 300
calls pfAbSynOp?, 282
defun, 282

pfPretendExpr, 282
called by pf2Sex1, 300
defun, 282

pfPretendType, 282
called by pf2Sex1, 300
defun, 282

pfPrintSrcLines
called by displayParserMacro, 464
called by npDef, 187
called by pfPushBody, 249
calls pfLambda, 249
calls pfNothing, 249
calls pfPushBody, 249
defun, 249

pfPushBody, 249
called by npDef, 187
called by pfPushMacroBody, 243
calls pfMLambda, 243
calls pfPushMacroBody, 243
defun, 243

pfQualType, 282
called by npQualType, 181
calls pfTree, 282
defun, 282

pfRestrict, 283
called by npRestrict, 220
calls pfTree, 283
defun, 283

pfRestrict?, 283
called by pf2Sex1, 301
calls pfAbSynOp?, 283
defun, 283

pfRestrictExpr, 283
called by pf2Sex1, 301
defun, 283

pfRestrictType, 283
called by pf2Sex1, 301
defun, 283

pfRetractTo, 284
called by npColonQuery, 219
calls pfTree, 284
defun, 284
pfReturn, 284
called by npReturn, 178
called by pfReturnNoName, 285
calls pfTree, 284
defun, 284
pfReturn?, 284
called by pf2Sex1, 301
calls pfAbSynOp?, 284
defun, 284
pfReturnExpr, 284
called by pf2Sex1, 301
defun, 284
pfReturnNoName, 285
called by npReturn, 178
calls pfNothing, 285
calls pfReturn, 285
defun, 285
pfReturnTyped, 285
called by npLambda, 149
calls pfTree, 285
defun, 285
pfRhsRule2Sex, 319
called by pfRule2Sex, 318
calls pf2Sex1, 319
uses $insideRule, 319
defun, 319
pfRule, 285
called by npSingleRule, 194
calls pfTree, 285
defun, 285
pfRule2Sex, 318
called by pf2Sex1, 301
calls pfLhsRule2Sex, 318
calls pfRhsRule2Sex, 318
calls pfRuleLhsItems, 318
calls pfRuleRhs, 318
calls ruleLhsTran, 318
calls rulePredicateTran, 318
uses $multiVarPredicateList, 318
uses $predicateList, 318
uses $quotedOpList, 318
defun, 318
pfRule?, 286
called by pf2Sex1, 301
calls pfAbSynOp?, 286
defun, 286
pfRuleLhsItems, 286
called by pfRule2Sex, 318
defun, 286
pfRuleRhs, 286
called by pfRule2Sex, 318
defun, 286
pfSecond, 286
called by pfLam, 272
defun, 286
pfSequence, 287
called by npBPileDefinition, 188
called by pfEnSequence, 263
calls pfTree, 287
defun, 287
pfSequence2Sex, 310
called by pf2Sex1, 300
calls pf0SequenceArgs, 310
calls pf2Sex1, 310
uses $insideSEQ, 310
defun, 310
pfSequence2Sex0, 310
called by pfSequence2Sex0, 310
calls pfSequence2Sex0, 310
defun, 310
pfSequence?, 287
called by pf2Sex1, 300
called by pfSequenceToList, 238
called by pfUnSequence, 293
calls pfAbSynOp?, 287
defun, 287
pfSequenceArgs, 287
called by pf0SequenceArgs, 287
called by pfSequenceToList, 238
defun, 287
pfSequenceToList, 238
called by npDefinition, 167
calls pfListOf, 239
calls pfSequence?, 238
calls pfSequenceArgs, 238
defun, 238
pfSexpr, 250
called by pfSymb, 251
calls pfSexpr,strip, 250
INDEX

defun, 250
pfSexpr,strip, 250

called by pfSexpr,strip, 250
called by pfSexpr, 250
calls pf0TupleParts, 250
calls pfAbSynOp, 250
calls pfApplication?, 250
calls pfApplicationArg, 250
calls pfApplicationOp, 250
calls pfId?, 250
calls pfIdSymbol, 250
calls pfLeaf?, 250
calls pfLiteral?, 250
calls pfLiteralString, 250
calls pfParts, 250
calls pfSexpr,strip, 250
calls pfTuple?, 250
calls tokPart, 250
defun, 250

pfSourcePosition, 238

called by mac0ExpandBody, 224
called by mac0MLambdaApply, 223
called by macId, 227
called by macMacro, 229
called by pfSourcePosition, 238
calls pfLeaf?, 238
calls pfLeafPosition, 238
calls pfParts, 238
calls pfSourcePosition, 238
calls poNoPosition?, 238
uses $nopos, 238
defun, 238

pfSourceStok, 243

called by npTrapForm, 212
called by pfSourceStok, 243
calls pfFirst, 243
calls pfLeaf?, 243
calls pfParts, 243
calls pfSourceStok, 243
defun, 243

pfSpread, 239

called by npDefaultDecl, 170
called by npLocalDecl, 172
called by npSigDecl, 157
calls pfTyped, 239
defun, 239

pfSuch, 244

called by pfCollectVariable1, 242
called by pfTaggedToTyped, 289
calls pfId, 244
calls pfInfApplication, 244
defun, 244

pfSuchthat, 288

called by npSuchThat, 176
calls pfTree, 288
defun, 288

pfSuchthat2Sex, 307

called by pfApplication2Sex, 306
calls pf0TupleParts, 307
calls pf2Sex1, 307
calls pf2Sex, 307
uses $predicateList, 307
defun, 307

pfSuchthat?, 288

called by pf2Sex1, 301
calls pfAbSynOp?, 288
defun, 288

pfSuchthatCond, 288

called by pf2Sex1, 301
defun, 288

pfSymb, 251

called by npConstTok, 184
called by npDDInfKey, 208
called by npInffixOperator, 160
calls ifcar, 251
calls pfExpression, 251
calls pfLeaf?, 251
calls pfSexpr, 251
calls pfSymbol, 251
calls tokPart, 251
defun, 251

pfSymbol, 251

called by pfSymb, 251
calls ifcar, 251
calls pfLeaf, 251
defun, 251

pfSymbol?, 252

called by pf2Sex1, 300
called by pfOp2Sex, 308
calls pfAbSynOp?, 252
defun, 252

pfSymbolSymbol, 252
called by pf2Sex1, 300
called by pfLiteral2Sex, 304
calls tokPart, 252
defun, 252
pfTagged, 288
called by npTagged, 218
calls pfTree, 288
defun, 288
pfTagged?, 289
called by pf2Sex1, 300
called by pfCheckItOut, 239
called by pfTaggedToTyped, 289
calls pfAbSynOp?, 289
defun, 289
pfTaggedExpr, 289
called by pf2Sex1, 300
called by pfCheckItOut, 239
called by pfTaggedToTyped, 289
defun, 289
pfTaggedTag, 289
called by pf2Sex1, 300
called by pfCheckItOut, 239
called by pfTaggedToTyped, 289
defun, 289
pfTaggedToTyped, 289
called by pfCheckItOut, 239
called by pfCollectVariable1, 242
called by pfTaggedToTyped1, 244
calls pfId?, 289
calls pfId, 289
calls pfInfApplication, 290
calls pfNothing, 289
calls pfSuch, 289
calls pfTagged?, 289
calls pfTaggedExpr, 289
calls pfTaggedTag, 289
calls pfTyped, 289
defun, 289
pfTaggedToTyped1, 244
called by pfCheckItOut, 239
called by pfTransformArg, 244
calls pfCollect1?, 244
calls pfCollectVariable1, 244
calls pfDefinition?, 244
calls pfNothing, 244
calls pfTaggedToTyped, 244
calls pfTyped, 244
defun, 244
pfTransformArg, 244
called by pfCheckItOut, 239
calls pf0TupleParts, 244
calls pfListOf, 244
calls pfTaggedToTyped1, 244
calls pfTuple?, 244
defun, 244
pfTree, 252
called by pfAdd, 252
called by pfAnd, 253
called by pfApplication, 253
called by pfAssign, 255
called by pfAttribute, 253
called by pfBreak, 258
called by pfCoerceTo, 259
called by pfCollect, 260
called by pfCopyWithPos, 236
called by pfDefinition, 261
called by pfDo, 262
called by pfExit, 263
called by pfExport, 264
called by pfForIn, 266
called by pfFree, 265
called by pfFromDom, 268
called by pfHide, 269
called by pfIf, 269
called by pfImport, 271
called by pfInline, 272
called by pffterate, 271
called by pfLambda, 273
called by pfListOf, 245
called by pfLocal, 274
called by pfLoop, 275
called by pfLambda, 278
called by pfMacro, 277
called by pfMapParts, 237
called by pfNothing, 245
called by pfNothing, 279
called by pfOr, 280
called by pfPretend, 281
called by pfQualType, 282
called by pfRestrict, 283
called by pfRetractTo, 284
called by pfReturnTyped, 285
INDEX

1627
called by pfReturn, 284
called by pfRule, 285
called by pfSequence, 287
called by pfSuchthat, 288
called by pfTagged, 288
called by pfTuple, 292
called by pfTweakIf, 290
called by pfTyped, 290
called by pfTyping, 291
called by pfWDeclare, 294
called by pfWhere, 294
called by pfWhile, 295
called by pfWith, 296
called by pfWrong, 296
defun, 252
pfTuple, 292
called by npEnclosed, 211
called by pfEnSequence, 263
called by pfInfApplication, 271
called by pfTupleListOf, 292
calls pfTree, 292
defun, 292
pfTuple?, 292
called by pf0FlattenSyntacticTuple, 237
called by pf2Sex1, 300
called by pfApplication2Sex, 306
called by pfCheckArg, 241
called by pfCheckItOut, 239
called by pfSexpr, strip, 250
called by pfTransformArg, 244
calls pfAbSynOp?, 292
defun, 292
pfTupleListOf, 292
called by pfTuple, 146
calls pfListOf, 292
calls pfTuple, 292
defun, 292
pfTupleParts, 293
called by pf0TupleParts, 293
defun, 293
pfTweakIf, 290
called by pfWConditional, 195
calls pfIfCond, 290
calls pfIfElse, 290
calls pfIfThen, 290
calls pfListOf, 290
calls pfNothing?, 290
calls pfTree, 290
defun, 290
pfTyped, 290
called by npDecl, 214
called by npVariableName, 214
called by pfCheckItOut, 239
called by pfCollectVariable1, 242
called by pfSpread, 239
called by pfTaggedToTyped1, 244
called by pfTaggedToTyped, 289
calls pfTree, 290
defun, 290
pfTyped?, 291
called by pf2Sex1, 301
called by pfLambdaTran, 316
calls pfAbSynOp?, 291
defun, 291
pfTypedId, 291
called by macLambdaParameterHandling, 231
called by pf2Sex1, 301
called by pfCollectVariable1, 242
called by pfLambdaTran, 316
defun, 291
pfTypedType, 291
called by pf2Sex1, 301
called by pfCollectVariable1, 243
called by pfLambdaTran, 316
defun, 291
pfTyping, 291
called by npTyping, 168
calls pfTree, 291
defun, 291
pfUnSequence, 293
called by npCategoryL, 152
called by npDefaultItemlist, 169
called by npLocalItemlist, 171
called by npQualTypelist, 180
calls pf0SequenceArgs, 293
calls pfAppend, 293
calls pfListOf, 293
calls pfSequence?, 293
defun, 293
pfWDec, 293
called by npSignature, 154
calls pfParts, 294
calls pfWDeclare, 293
defun, 293
pfWDeclare, 294
called by pfWDec, 293
calls pfTree, 294
defun, 294
pfWhere, 294
called by npLetQualified, 167
called by npQualified, 147
calls pfTree, 294
defun, 294
pfWhere?, 294
called by macExpand, 222
called by pf2Sex1, 301
calls pfAbSynOp?, 294
defun, 294
pfWhereContext, 295
called by pf0WhereContext, 295
defun, 295
pfWhereExpr, 295
called by pf2Sex1, 301
defun, 295
pfWhile, 295
called by npWhile, 177
calls pfTree, 295
defun, 295
pfWhile?, 296
called by pf2Sex1, 300
calls pfAbSynOp?, 296
defun, 296
pfWhileCond, 296
called by pf2Sex1, 300
defun, 296
pfWith, 296
called by npWith, 150
calls pfTree, 296
defun, 296
pfWrong, 296
called by npParse, 141
called by npRecoverTrap, 190
calls pfTree, 296
defun, 296
pfWrong?, 297
called by pf2Sex1, 301
calls pfAbSynOp?, 297
defun, 297
phInterpret, 65
calls intInterpretPform, 65
calls ncElQ, 65
calls ncPutQ, 65
defun, 65
phIntReportMsgs, 64
calls ncElQ, 64
calls ncPutQ, 65
calls processMsgList, 65
uses $erMsgToss, 65
defun, 64
phMacro, 221
calls macroExpanded, 221
calls ncElQ, 221
calls ncPutQ, 221
calls pform, 221
defun, 221
phParse, 64
calls ncPutQ, 64
defun, 64
pileCforest, 340
called by insertpile, 335
called by pileCtree, 340
calls enPile, 340
calls separatePiles, 340
calls tokPart, 340
defun, 340
pileColumn, 337
called by eqpileTree, 339
called by pileTree, 337
calls tokPosn, 337
defun, 337
pileCtree, 340
called by pileForests, 338
calls dqAppend, 340
calls pileCforest, 340
defun, 340
pileForest, 338
called by pileForests, 337
calls pileForest1, 337
calls pileTree, 338
defun, 338
pileForest1, 338
called by pileForest1, 339
called by pileForest, 338
INDEX

calls eqpileTree, 338
calls pileForest1, 339
defun, 338
pileForests, 337
calledby eqpileTree, 339
calledby pileForests, 338
calledby pileTree, 337
calls npNull, 338
calls pileCtree, 338
calls pileForests, 338
calls pileForest, 337
defun, 337
pilePlusComment, 336
calledby insertpile, 335
calledby pilePlusComments, 336
calls tokType, 336
defun, 336
pilePlusComments, 336
calledby insertpile, 335
calledby pilePlusComments, 336
calls npNull, 336
calls pilePlusComments, 336
calls pilePlusComment, 336
defun, 336
pileTree, 337
calledby insertpile, 335
calledby pileForest, 338
calls npNull, 337
calls pileColumn, 337
calls pileForests, 337
defun, 337
placep
calledby unwritable?, 606
calledby writify, writifyInner, 607
pluralize
calledby dbConsHeading, 1395
calledby dbShowConditions, 1395
calledby kePageDisplay, 1354
pluralSay
calledby kePage, 1352
PLUSCOMMENT, 106
defvar, 106
pmDontQuote?, 309
calledby pfOp2Sex, 308
defun, 309
pmTransFilter

calledby dbShowCons, 1387
calledby koaPageFilterByName, 1380
pname, 1045
calledby clearCmdParts, 505
calledby coerceTraceArgs2E, 864
calledby coerceTraceFunValue2E, 867
calledby displayType, 460
calledby displayValue, 459
calledby fixObjectForPrinting, 456
calledby gensymInt, 616
calledby getAliasIfTracedMapParameter, 890
calledby getStFromMsg, 354
calledby hasOption, 451
calledby isSharpVarWithNum, 886
calledby kcnPage, 1369
calledby letPrint2, 887
calledby letPrint3, 888
calledby letPrint, 885
calledby mac0InfiniteExpansion, name, 226
calledby mkConArgSublis, 1405
calledby newHelpSpad2Cmd, 573
calledby npMissing, 151
calledby reportUndo, 928
calledby rwrite, 604, 605
calledby selectOption, 480
calledby setFortDir, 726
calledby setFortTmpDir, 724
calledby setOutputCharacters, 767
calledby stupidIsSpadFunction, 906
calledby undo, 922
defun, 1045
poCharPosn, 377
calledby compareposns, 370
calledby pfCharPosn, 235
calledby posPointers, 378
calledby processChPosesForOneLine, 376
calledby putFTText, 379
defun, 377
poFileName, 360
calledby decideHowMuch, 359
calledby pfFileName, 236
calls lnFileName, 360
calls poGetLineObject, 360
defun, 360
poGetLineObject, 361
called by poFileName, 360
called by poGlobalLinePosn, 70
called by poLinePosn, 361
called by poPosImmediate?, 360
defun, 361
poGlobalLinePosn, 70
called by compareposns, 370
called by listDecideHowMuch, 361
called by makeMsgFromLine, 371
called by nloopDQlines, 70
called by pfGlobalLinePosn, 235
called by processMsgList, 369
called by thisPosIsEqual, 374
called by thisPosIsLess, 374
calls lnGlobalNum, 70
calls ncBug, 70
calls poGetLineObject, 70
defun, 70
poLinePosn, 361
called by decideHowMuch, 359
called by makeMsgFromLine, 371
called by pfLinePosn, 235
calls lnLocalNum, 361
calls poGetLineObject, 361
defun, 361
poNoPosition, 414
called by pfNoPosition, 414
uses $nopos, 414
defun, 414
poNoPosition?, 413
called by pfNoPosition?, 412
called by pfSourcePosition, 238
calls eqcar, 413
defun, 413
poPosImmediate?, 360
called by decideHowMuch, 359
called by listDecideHowMuch, 361
calls lnImmediate?, 360
calls poGetLineObject, 360
calls poNopos?, 360
defun, 360
porigin, 87
called by inclmsgFileCycle, 91
called by ppos, 357
calls stringp, 87
defun, 87
posend, 129
called by scanW, 128
defun, 129
position
called by dbSubConform, 1386
called by kArgPage, 1346
posPointers, 378
called by processChPosesForOneLine, 376
calls IFCAR, 378
calls getMsgPos2, 378
calls getMsgPos, 378
calls insertPos, 378
calls poCharPosn, 378
calls poNoPosition, 414
calls poPosImmediate?, 360
calls poGetLineObject, 360
calls porigin, 357
poundsign
called by dewritify, dewritifyInner, 612
called by displaySetVariableSettings, 657
called by getTraceOptions, 852
called by isDomainOrPackage, 875
called by newHelpSpad2Cmd, 573
called by set1, 809
called by setOpenLibrary, 664
called by trace1, 848
called by traceReply, 899
pp
called by mkConform, 1374
called by reportUndo, 928
pp2Cols
called by filterAndFormatConstructors, 944
ppos, 357
called by getPosStL, 356
calls pfCharPosn, 357
calls pfFileName, 357
calls pfImmediate?, 357
calls pfLinePosn, 357
calls pfNoPosition?, 357
calls porigin, 357
uses $outputLines, 57
uses $printTimeIfTrue, 57
uses $printTypeIfTrue, 57
defun, 57
printTypeAndTimeSaturn, 58
calledby printTypeAndTime, 56
calls devaluate, 58
calls form2StringAsTeX, 58
calls printAsTeX, 58
calls $interpreterTimedClasses, 58
uses $interpreterTimedNames, 58
uses $printTimeIfTrue, 58
uses $printTypeIfTrue, 58
defun, 58
prior-token
usedby token-stack-show, 971
probeName, 984
defun, 984
processChPosesForOneLine, 376
calledby queueUpErrors, 372
calls getMsgFTTag?, 376
calls getMsgPos, 376
calls getMsgPrefix, 376
calls makeLeaderMsg, 376
calls poCharPosn, 376
calls posPointers, 376
calls putFTText, 376
calls setMsgPrefix, 376
calls size, 376
uses $preLength, 376
defun, 376
processInteractive, 48
calledby intInterpretPform, 65
calledby kisValidType, 1372
calledby topLevelInterpEval, 1372
calls clrhash, 48
calls initializeTimedNames, 48
calls processInteractive1, 48
calls qcar, 48
calls reportInstantiations, 48
calls updateHist, 48
calls writeHistModesAndValues, 48
uses $Coerce, 48
uses $ProcessInteractiveValue, 48
uses $StreamFrame, 48
uses $analyzingMapList, 48
uses $compErrorMessageStack, 48
uses $compilingMap, 48
uses $compilingLoop, 48
uses $declaredMode, 48
uses $defaultFortVar, 48
uses $domainVar, 48
uses $fortVar, 48
uses $freeVars, 48
uses $inRetract, 48
uses $instantCanCoerceCount, 48
uses $instantCoerceCount, 48
uses $instantMmCondCount, 48
uses $instantRecord, 48
uses $interpOnly, 48
uses $interpreterTimedClasses, 48
uses $interpreterTimedNames, 48
uses $lastLineInSEQ, 48
uses $mapList, 48
uses $minivectorCode, 48
uses $minivectorNames, 48
uses $minivector, 48
uses $op, 48
uses $reportInstantiations, 48
uses $timeGlobalName, 48
uses $whereCacheList, 48
defun, 48
processInteractive1, 50
calledby processInteractive, 48
calls interpretTopLevel, 50
calls objMode, 50
calls objValUnwrap, 50
calls recordAndPrint, 50
calls recordFrame, 50
calls startTimingProcess, 50
calls stopTimingProcess, 50
uses $InteractiveFrame, 50
uses $ProcessInteractiveValue, 50
uses $e, 50
defun, 50
processKeyedError, 353
calledby ncBug, 368
calledby ncHardError, 352
calledby ncSoftError, 351
calls CallerName, 353
calls getMsgKey, 353
calls getMsgPrefix?, 353
calls getMsgTag?, 353
calls msgImPr?, 353
calls msgOutputter, 353
calls sayBrightly, 353
uses $ncMsgList, 353
defun, 353
processMsgList, 369
called by phIntReportMsgs, 65
calls erMsgSort, 369
calls getMsgPos, 369
calls listOutputter, 369
calls makeMsgFromLine, 369
calls queueUpErrors, 369
uses $noRepList, 369
uses $outputList, 369
defun, 369
processSynonymLine, 835
called by npProcessSynonym, 473
called by synonymSpad2Cmd, 832
processSynonymLine, removeKeyFromLine, 834
defun, 835
processSynonymLine, removeKeyFromLine, 834
called by processSynonymLine, 835
calls dropLeadingBlanks, 834
calls maxindex, 834
defun, 834
processSynonyms, 31
called by doSystemCommand, 446
called by intProcessSynonyms, 31
called by processSynonyms, 32
calls concat, 32
calls lassoc, 31
calls processSynonyms, 32
calls rplacstr, 32
calls size, 32
calls strconc, 32
calls string2id-n, 31
calls strpos, 31
calls substring, 31
uses $CommandSynonymAlist, 32
uses line, 32
defun, 31
protectedEVAL, 45
called by serverReadLine, 43
calls resetStackLimits, 45
calls sendHTErrorSignal, 45
defun, 45
prTraceNames, 898
calls exit, 898
calls prTraceNames, 898
defun, 898
prTraceNames, fn, 898
called by prTraceNames, 898
calls devaluate, 898
calls exit, 898
calls isDomainOrPackage, 898
calls qcar, 898
calls qcdr, 898
calls seq, 898
defun, 898
ptimers, 859
called by trace1, 848
calls bright, 859
calls concat, 859
calls float, 859
calls quotient, 859
calls sayBrightly, 859
uses /timerlist, 859
defun, 859
punctuation?, 118
called by scanToken, 114
defun, 118
put, 1039
defun, 1039
putalist
called by interpFunctionDepAlists, 465
called by npProcessSynonym, 473
called by synonymSpad2Cmd, 832
putDatabaseStuff, 365
called by msgCreate, 348
  calls getMsgInfoFromKey, 365
  calls setMsgText, 365
  calls setMsgUnforcedAttrList, 365
defun, 365

putFTText, 379
  called by processChPosesForOneLine, 376
  calls getMsgFTTag?, 379
  calls getMsgPos2, 379
called by getMsgPos, 379
called by getMsgText, 379
called by poCharPosn, 379
called by setMsgText, 379
defun, 379

putHist, 590
  called by importFromFrame, 563
called by recordAndPrint, 54
called by restoreHistory, 597
called by undoChanges, 593
called by undoFromFile, 594
called by undoInCore, 593
called by writeHistModesAndValues, 603
called by get, 590
called by putIntSymTab, 591
called by putTarget
  called by putHist, 591
putTarget
called by evaluateType1, 918
called by interpret1, 52

called by pvarCondList, 1281
defun, 1281
called by pvarCondList1, 1281
defun, 1281
called by pvarPredTran, 322
defun, 322
called by rulePredicateTran, 319
called by pvarsOfPattern, 1282
defun, 1282

qcaar
called by getConstructorDocumentation, 1394

called by dbSpecialExpandIfNecessary, 1400
called by getConstructorDocumentation, 1394

called by /tracereply, 894
called by ?t, 903
called by NRTevalDomain, 1079
called by ScanOrPairVec, ScanOrInner, 615
called by abbreviationsSpad2Cmd, 483
called by gbGetDocTable, hh, 1384
called by dbSpecialExpandIfNecessary, 1400
called by dewritify, dewritifyInner, 612
called by displayProperties, 461
called by domainDescendantsOf, 1349
called by evaluateType1, 918
called by evaluateType, 916
called by frameSpad2Cmd, 566
called by funfind, LAM, 874
called by getConstructorDocumentation, 1394
called by getTraceOption, 854
called by hasPair, 891
called by kcPage, 1357
called by kccPage, 1364
called by kcnPage, 1368
called by kcpPage, 1361
called by mkEvalable, 913
called by ncAlist, 415
called by ncTag, 415
called by prTraceNames, fn, 898
called by printTypeAndTimeNormal, 57
called by processInteractive, 48
called by reportOperations, 815
called by reportOpsFromUnitDirectly, 822
called by reportSpadTrace, 892
called by restoreHistory, 597
called by retract, 1064
called by selectOption, 480
<table>
<thead>
<tr>
<th>Function Call</th>
<th>Called By</th>
</tr>
</thead>
<tbody>
<tr>
<td>called by setExposeAddConstr</td>
<td>700</td>
</tr>
<tr>
<td>called by setExposeAddGroup</td>
<td>699</td>
</tr>
<tr>
<td>called by setExposeAdd</td>
<td>698</td>
</tr>
<tr>
<td>called by setExposeDropConstr</td>
<td>703</td>
</tr>
<tr>
<td>called by setExposeDropGroup</td>
<td>702</td>
</tr>
<tr>
<td>called by setExposeDrop</td>
<td>701</td>
</tr>
<tr>
<td>called by setExpose</td>
<td>697</td>
</tr>
<tr>
<td>called by setInputLibrary</td>
<td>667</td>
</tr>
<tr>
<td>called by setOutputAlgebra</td>
<td>763</td>
</tr>
<tr>
<td>called by setOutputCharacters</td>
<td>767</td>
</tr>
<tr>
<td>called by setOutputFormula</td>
<td>790</td>
</tr>
<tr>
<td>called by setOutputFortran</td>
<td>770</td>
</tr>
<tr>
<td>called by setOutputHtml</td>
<td>781</td>
</tr>
<tr>
<td>called by setOutputMathml</td>
<td>776</td>
</tr>
<tr>
<td>called by setOutputOpenMath</td>
<td>786</td>
</tr>
<tr>
<td>called by setOutputTex</td>
<td>796</td>
</tr>
<tr>
<td>called by spadClosure?</td>
<td>611</td>
</tr>
<tr>
<td>called by spadReply, printName</td>
<td>894</td>
</tr>
<tr>
<td>called by trace1</td>
<td>848</td>
</tr>
<tr>
<td>called by traceReply</td>
<td>899</td>
</tr>
<tr>
<td>called by traceSpad2Cmd</td>
<td>847</td>
</tr>
<tr>
<td>called by transOnlyOption</td>
<td>860</td>
</tr>
<tr>
<td>called by undoSteps</td>
<td>930</td>
</tr>
<tr>
<td>called by undo</td>
<td>922</td>
</tr>
<tr>
<td>called by untraceDomainConstructor, keepTrace</td>
<td>882</td>
</tr>
<tr>
<td>called by writeSpad2Cmd</td>
<td>939</td>
</tr>
<tr>
<td>called by writify, writifyInner</td>
<td>607</td>
</tr>
</tbody>
</table>

**qcdr**
- called by dbSpecialExpandIfNecessary | 1400
- called by getConstructorDocumentation | 1394

**qceseq**
- called by dbShowCons1 | 1389
- called by dbShowCons1 | 1389
called by writify, writifyInner, 607

qrplacd
called by dewritify, dewritifyInner, 612
called by writify, writifyInner, 607

qsabsval, 1069
defmacro, 1069

qsadd1, 1068
defmacro, 1068

qsdifference, 1067
defmacro, 1067

qsDot26432, 1063
defmacro, 1063

qsDot2Mod6432, 1063
defmacro, 1063

qsetvelt
called by dewritify, dewritifyInner, 612
called by writify, writifyInner, 607

qslessp, 1068
called by trace1, 848
defmacro, 1068

qsmax, 1070
defmacro, 1070

qsmin, 1070
defmacro, 1070

qsmminus, 1068
defmacro, 1068

qsmul6432, 1062
defmacro, 1062

qsmul6432, 1063
defmacro, 1063

qsmulAdd6432, 1062
defmacro, 1062

qsmulAddMod6432, 1063
defmacro, 1063

qsoddp, 1069
defmacro, 1069

qsplus, 1069
defmacro, 1069

qsquotient, 1067
defun, 1067

qsremainder, 1067
defun, 1067

qsclrbl, 1068
defmacro, 1068

qstimes, 1069
defmacro, 1069

qszerop, 1070
defmacro, 1070

queryClients, 510
called by close, 511
calls sockGetInt, 510
calls sockSendInt, 510
uses $QueryClients, 510
uses $SessionManager, 510
defun, 510

queryUserKeyedMsg
called by close, 510
called by historySpad2Cmd, 582
called by importFromFrame, 563
called by listConstructorAbbreviations, 484
called by quitSpad2Cmd, 638
called by yesanswer, 537

QUESTION, 107
defvar, 107

queueUpErrors, 372
called by processMsgList, 369
calls processChPosesForOneLine, 372
uses $outputList, 372
defun, 372

quickAnd
called by domainDescendantsOf, 1349

quit, 638
calls quitSpad2Cmd, 638
defun, 638

quit help page, 637
manpage, 637

quitSpad2Cmd, 638
called by pquitSpad2Cmd, 634
called by quit, 638
calls leaveScratchpad, 638
calls queryUserKeyedMsg, 638
calls sayKeyedMsg, 638
calls string2id-n, 638
calls tersyscommand, 638
calls upcase, 638
uses $quitCommandType, 638
defun, 638

quoteString, 1297
defun, 1297

QUOTIENT
INDEX

called by Else?, 78
called by Elseif?, 78

quotient
called by If?, 77
called by Top?, 77
called by ptimers, 859

quotient2, 1088
defun, 1088

qvelt
called by dewritify,dewritifyInner, 612
called by writify,writifyInner, 607
qvlenU16, 1055
defmacro, 1055
qvlenU32, 1056
defmacro, 1056
qvlenU8, 1054
defmacro, 1054
qvmaxindex
called by dewritify,dewritifyInner, 612
called by writify,writifyInner, 607

RADIXCHAR, 106
defvar, 106
random, 1088
defun, 1088

rassoc
called by rassocSub, 871
called by saveMapSig, 853
rassocSub, 871
called by ?', 903
called by isSubForRedundantMapName, 873
called by traceReply, 899
calls rassoc, 871
defun, 871
rdefinestream, 1074
calls rdefinestream, 1074
defun, 1074
rdefiostream, 1074
called by rdefinestream, 1074
called by readHiFi, 601
called by saveHistory, 595
called by setHistoryCore, 585
called by writeHiFi, 602
rdefoutstream, 1075
calls rdefiostream, 1075
defun, 1075
rdigit?, 133
calls strpos, 133
defun, 133
read, 642
called by undo, 922
calls readSpad2Cmd, 642
defun, 642
read help page, 641
manpage, 641
read-line
called by serverReadLine, 42
readHiFi, 601
called by fetchOutput, 600
called by restoreHistory, 597
called by setHistoryCore, 585
called by showInOut, 600
called by showInput, 599
called by undoFromFile, 594
called by undoInCore, 593
called by writeInputLines, 587
calls assoc, 601
calls histFileName, 601
calls keyedSystemError, 601
calls object2Identifier, 601
calls qcdr, 601
calls rdefiostream, 601
calls rshut, 601
calls spadrread, 601
uses $internalHistoryTable, 601
uses $useInternalHistoryTable, 601
defun, 601
readLibPathFast
called by getDependentsOfConstructor, 1367
called by getUsersOfConstructor, 1368
readSpad2Cmd, 642
called by read, 642
calls /read, 642
calls findfile, 642
calls makePathname, 642
calls member, 642
calls mergePathnames, 642
calls namestring, 642
calls optionError, 642
calls pathnameName, 642
calls pathnameTypeId, 642
calls pathname, 642
  calls selectOptionLC, 642
  calls throwKeyedMsg, 642
  calls upcase, 642
  uses /editfile, 643
  uses $InteractiveMode, 642
  uses $UserLevel, 642
  uses $findfile, 642
  uses $options, 642
  defun, 642
readSpadProfileIfThere, 961
  calledby restart, 15
  uses /editfile, 961
  defun, 961
reclaim, 37
  calledby clearCmdCompletely, 502
  defun, 37
recordAndPrint, 54
  calledby processInteractive1, 50
  calls mkCompanionPage, 54
  calls objNewWrap, 54
  calls output, 54
  calls printStatisticsSummary, 54
  calls printStorage, 54
  calls printTypeAndTime, 54
  calls putHist, 54
  calls recordAndPrintTest, 54
  uses $InteractiveFrame, 923
  uses $frameRecord, 923
  uses $previousBindings, 923
  uses $undoFlag, 923
  defun, 923
recordNewValue, 591
  calledby clearCmdParts, 505
  calledby putHist, 591
  calledby undoFromFile, 594
  calls recordNewValue0, 591
  calls startTimingProcess, 591
  calls stopTimingProcess, 591
  defun, 591
recordNewValue0, 591
  calledby recordNewValue, 591
  calls assq, 591
  uses $HistRecord, 591
  defun, 591
recordOldValue, 592
  calledby clearCmdParts, 505
  calledby putHist, 590
  calledby undoFromFile, 594
  calls recordOldValue0, 592
  calls startTimingProcess, 592
  calls stopTimingProcess, 592
  defun, 592
recordOldValue0, 592
  calledby recordOldValue, 592
  calls assq, 592
  uses $HistList, 592
  defun, 592
reduce-stack-clear
  calledby ioclear, 972
reduceAlistForDomain, 1362
  calledby kccPage, 1364
  calls nreverse0, 1362
  calls simpHasPred, 1362
  calls sublis, 1362
defun, 1362
redundant, 374
calls msgNoRep?, 374
calls sameMsg?, 374
uses $noRepList, 374
defun, 374
refvecp
calledby isDomainOrPackage, 875
calledby spadTrace, 876
regress help page, 645
manpage, 645
remainder
calledby KeepPart?, 79
calledby SkipEnd?, 78
calledby SkipPart?, 79
remainder2, 1087
defun, 1087
remalist
calledby clearParserMacro, 453
rendup
calledby clearCmdParts, 505
calledby dbConsHeading, 1395
calledby dbShowCons1, 1389
calledby dbShowConsDoc, 1392
calledby displayMacros, 538
calledby displayOperationsFromLisplib, 820
calledby displayProperties, 461
calledby reportOpsFromLisplib, 818
calledby reportOpsFromUnitDirectly, 822
remFile, 357
calledby getPosStL, 356
calls IFCDR, 357
defun, 357
remLine, 362
calledby getPosStL, 356
calls IFCAR, 357
defun, 362
removeAttributes
calledby getMsgInfoFromKey, 366
removeOption, 861
calledby spadTrace, 876
defun, 861
remover, 897
calledby remover, 897
calledby spadUntrace, 896
calls remover, 897
defun, 897
removeTracedMapSigs, 864
calledby untrace, 862
uses $tracedMapSignatures, 864
defun, 864
removeUndoLines, 933
calls charPosition, 933
calls concat, 934
calls exit, 933
calls maxindex, 934
calls seq, 933
calls spaddifference, 934
calls stringPrefix?, 933
calls substring, 933
calls trimString, 933
calls undoCount, 934
uses $IOindex, 934
uses $currentLine, 934
defun, 933
removeZeroOneDestructively
calledby reportOperations, 816
remprop
calledby loadLib, 1036
rempropI
calledby restoreHistory, 597
renamePatternVariables, 1278
defun, 1278
renamePatternVariables1, 1278
defun, 1278
rep, 371
calledby makeMsgFromLine, 371
defun, 371
repeat, 402
syntax, 402
replaceFile, 985
defun, 985
replacePercentByDollar, 1256
defun, 1256
replacePercentByDollar,fn, 1255
defun, 1255
replaceSharps, 958
calledby evaluateType1, 917
calls subCopy, 958
local ref $FormalMapVariableList, 958
local ref $TriangleVariableList, 958
defun, 958
reportAO, 1348
called by reportCategory, 1347
calls bcConform, 1348
calls escapeSpecialChars, 1348
calls form2HtString, 1348
calls htSay, 1348
calls satDownLink, 1348
defun, 1348
reportCategory, 1347
calls bcConPredTable, 1347
calls bcConform, 1347
calls bcPred, 1347
calls categoryParts, 1347
calls htSay, 1347
calls reportAO, 1347
defun, 1347
reportInstantiations
called by processInteractive, 48
reportinstantiations, 1053
defun, 1053
reportOperations, 815
called by showSpad2Cmd, 814
calls bright, 815
calls evaluateType, 816
calls isDomainValuedVariable, 815
calls isNameOfType, 815
calls isType, 816
calls mkAtree, 816
calls opOf, 816
calls qcar, 815
calls removeZeroOneDestructively, 816
calls reportOpsFromLisplib0, 816
calls reportOpsFromUnitDirectly0, 816
calls sayBrightly, 815
calls sayKeyedMsg, 815
calls unabbrev, 816
uses $doNotAddEmptyModeIfTrue, 816
uses $env, 816
uses $eval, 816
uses $genValue, 816
uses $quadSymbol, 816
defun, 815
reportOpsFromLisplib, 818
called by reportOpsFromLisplib0, 817
called by reportOpsFromLisplib1, 817
calls bright, 818
calls centerAndHighlight, 818
calls concat, 818
calls constructor?, 818
calls dc1, 818
calls displayOperationsFromLisplib, 818
calls eqsubstlist, 818
calls form2StringWithWhere, 818
calls formatAttribute, 818
calls getConstructorSignature, 818
calls getdatabase, 818
calls isExposedConstructor, 818
calls kdr, 818
calls msort, 818
calls namestring, 818
calls remrev, 818
calls remdup, 818
calls say2PerLine, 818
calls sayBrightly, 818
calls sayKeyedMsg, 818
calls selectOptionLC, 818
calls specialChar, 818
calls strconc, 818
uses $FormalMapVariableList, 818
uses $linelength, 818
uses $options, 818
uses $showOptions, 818
defun, 818
reportOpsFromLisplib0, 817
called by reportOperations, 816
called by reportOpsFromLisplib0, 816
called by reportOpsFromLisplib1, 817
called by reportOpsFromLisplib0, 816
called by reportOpsFromLisplib1, 817
calls cost, 817
calls defio, 817
calls editFile, 817
calls erase, 817
calls pathname, 817
calls reportOpsFromLisplib, 817
calls sayShowWarning, 817
calls shut, 817
uses $erase, 817
uses $sayBrightlyStream, 817
defun, 817
reportOpsFromUnitDirectly, 821
called by: displayOperationsFromLispLib, 820
called by: reportOpsFromUnitDirectly0, 821
called by: reportOpsFromUnitDirectly1, 825
calls bright, 822
calls centerAndHighlight, 822
calls evalDomain, 822
calls formatAttribute, 822
calls formatOperation, 822
calls getOpListForConstructorForm, 822
calls getDatabase, 822
calls getl, 822
calls isExposedConstructor, 822
calls member, 821
calls msort, 822
calls namestring, 822
calls nreverse0, 822
calls qcar, 822
calls remdup, 822
calls say2PerLine, 822
calls sayBrightly, 822
calls specialChar, 822
calls strconc, 822
calls systemErrorHere, 822
uses $CategoryFrame, 822
uses $commentedOps, 822
uses $linelength, 822
uses $options, 822
uses $showOptions, 822
defun, 821
reportOpsFromUnitDirectly0, 821
called by: reportOperations, 816
called by: reportOpsFromUnitDirectly1, 821
called by: reportOpsFromUnitDirectly, 821
uses $useEditorForShowOutput, 821
defun, 821
reportOpsFromUnitDirectly1, 825
called by: reportOpsFromUnitDirectly0, 821
called by: reportOpsFromUnitDirectly1, 825
calls defostream, 825
calls editFile, 825
calls erase, 825
calls pathName, 825
calls reportOpsFromUnitDirectly, 825
calls sayShowWarning, 825
calls shut, 825
uses $erase, 826
uses $sayBrightlyStream, 825
defun, 825
reportOpSymbol
called by: displayOperations, 537
reportSpadTrace, 892
called by: ?t, 903
called by: spadTrace.isTraceable, 876
called by: spadTrace, 877
calls qcar, 892
calls sayBrightly, 892
uses $traceNoisely, 892
defun, 892
reportUndo, 927
called by: diffAlist, 925
called by: concat, 928
called by: exit, 928
called by: lassoc, 928
called by: pname, 928
calls pp, 928
calls sayBrightlyNT, 928
calls sayBrightly, 928
calls seq, 928
defun, 927
reportundo
uses $InteractiveFrame, 928
reportWhatOptions, 941
called by: whatSpad2Cmd, 940
calls sayBrightly, 941
uses $whatOptions, 941
defun, 941
reroot, 38
called by: initroot, 33
called by: inFile, 33
calls make-absolute-fileName, 39
uses $current-directory, 39
uses $defaultMsgDatabaseName, 39
uses $directory-list, 39
uses $library-directory-list, 39
uses $msgDatabaseName, 39
uses $relative-directory-list, 39
uses $relative-library-directory-list, 39
uses $spadroot, 39
<table>
<thead>
<tr>
<th>Function</th>
<th>Calls</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>reset-stack-limits</td>
<td>called by resetStackLimits, 19</td>
<td></td>
</tr>
<tr>
<td>resetCounters, 858</td>
<td>called by trace1, 848</td>
<td>called by reset-stack-limits, 19</td>
</tr>
<tr>
<td></td>
<td>calls concat, 858</td>
<td>calls /countlist, 858</td>
</tr>
<tr>
<td></td>
<td>defun, 858</td>
<td></td>
</tr>
<tr>
<td>resetHashTable, 1000</td>
<td>calls browsecopen, 1000</td>
<td>uses <em>allconstructors</em>, 1001</td>
</tr>
<tr>
<td></td>
<td>calls categoryopen, 1000</td>
<td>uses <em>browse-stream</em>, 1000</td>
</tr>
<tr>
<td></td>
<td>calls initial-getdatabase, 1000</td>
<td>uses <em>category-stream</em>, 1000</td>
</tr>
<tr>
<td></td>
<td>calls interopen, 1000</td>
<td>uses <em>category-stream-stamp</em>, 1000</td>
</tr>
<tr>
<td></td>
<td>calls operationopen, 1000</td>
<td>uses <em>hascategory-hash</em>, 1001</td>
</tr>
<tr>
<td></td>
<td>uses <em>interp-stream</em>, 1000</td>
<td>uses <em>interp-stream-stamp</em>, 1001</td>
</tr>
<tr>
<td></td>
<td>uses <em>operation-stream</em>, 1000</td>
<td>uses <em>operation-stream-stamp</em>, 1000</td>
</tr>
<tr>
<td></td>
<td>uses <em>sourcefiles</em>, 1000</td>
<td>defun, 1000</td>
</tr>
<tr>
<td>resetInCoreHist, 588</td>
<td>called by clearCmdAll, 503</td>
<td></td>
</tr>
<tr>
<td></td>
<td>called by historySpad2Cmd, 582</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $HistListAct, 588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $HistListLen, 588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $HistList, 588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defun, 588</td>
<td></td>
</tr>
<tr>
<td>resetSpacers, 858</td>
<td>called by trace1, 848</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls concat, 858</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /spacelist, 858</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defun, 858</td>
<td></td>
</tr>
<tr>
<td>resetStackLimits, 19</td>
<td>called by intloop, 24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>called by protectedEVAL, 45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>called by runspad, 19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls reset-stack-limits, 19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defun, 19</td>
<td></td>
</tr>
<tr>
<td>resetTimers, 858</td>
<td>called by trace1, 848</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls concat, 858</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /countlist, 858</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defun, 858</td>
<td></td>
</tr>
<tr>
<td>resetWorkspaceVariables, 654</td>
<td>calls copy, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls initializeSetVariables, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /countlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /editfile, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /pretty, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /sourcefiles, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /spacelist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses /timerlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $CommandSynonymAlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $IOindex, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $InitialCommandSynonymAlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $UserAbbreviationsAlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $boot, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $coerceIntByMapCounter, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $compileMapFlag, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $dependeeClosureAlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $echoLineStack, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $env, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $existingFiles, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $e, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $functionTable, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $msgAlist, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $msgDatabaseName, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $msgDatabase, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $operationNameList, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $setOptions, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $slamFlag, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses $sourceFiles, 654</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defun, 654</td>
<td></td>
</tr>
<tr>
<td>resolveTT</td>
<td>called by unifyStructVar, 429</td>
<td></td>
</tr>
<tr>
<td>Rest, 75</td>
<td>called by incLudel, 80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defmacro, 75</td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>calls get-current-directory, 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls init-memory-config, 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls initHist, 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls initializeInterpreterFrameRing, 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls initroot, 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls makeInitialModemapFrame, 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calls opserver, 15</td>
<td></td>
</tr>
</tbody>
</table>
calls readSpadProfileIfThere, 15
 calls restart0, 15
 calls spadStartUpMsgs, 15
 calls spad, 15
 calls statisticsInitialization, 15
 uses $IOindex, 16
 uses $InteractiveFrame, 16
 uses $SpadServerName, 15
 uses $SpadServer, 16
 uses $current-directory, 16
 uses $currentLine, 16
 uses $displayStartMsgs, 16
 uses $openServerIfTrue, 15
 uses $printLoadMsgs, 16

 restart function, 14
 restart0, 16
 calledby restart, 15
 calls browseopen, 17
 calls categoryopen, 17
 calls getEnv, 17
 calls interpopen, 16
 calls operationopen, 16
 defun, 16

 restoreHistory, 597
 calledby historySpad2Cmd, 582
 calls $fcopy, 597
 calls clearCmdSortedCaches, 597
 calls clearSpad2Cmd, 597
 calls disableHist, 597
 calls get, 597
 calls histFileErase, 597
 calls histFileName, 597
 calls identp, 597
 calls makeHistFileName, 597
 calls makeInputFilename, 597
 calls namestring, 597
 calls putHist, 597
 calls qcar, 597
 calls qcdr, 597
 calls readHiFi, 597
 calls rempropl, 597
 calls rkeyids, 597
 calls sayKeyedMsg, 597
 calls throwKeyedMsg, 597
 calls updateInCoreHist, 597
 uses $HiFiAccess, 597
 uses $InteractiveFrame, 597
 uses $e, 597
 uses $internalHistoryTable, 597
 uses $oldHistoryFileName, 597
 uses $options, 597
 uses $useInternalHistoryTable, 597
 defun, 597

 retract, 1064
 calledby printTypeAndTimeNormal, 57
 calls isWrapped, 1064
 calls objMode, 1064
 calls objNew, 1064
 calls objVal, 1064
 calls qcar, 1064
 calls retract1, 1064
 local ref $EmptyMode, 1064
 defun, 1064

 retract1
 calledby retract, 1064
 rkeyids
 calledby restoreHistory, 597
 calledby setHistoryCore, 585
 rplac
 calledby spadTrace, 877
 calledby spadUntrace, 896
 rplacstr
 calledby processSynonyms, 32
 rread, 605
 calledby SPADRREAD, 605
 calledby getDependentsOfConstructor, 1367
 calledby getUsersOfConstructor, 1368
 calledby rread, 605
 calls rread, 605
 defun, 605

 rshut
 calledby getDependentsOfConstructor, 1367
 calledby getUsersOfConstructor, 1368
 calledby readHiFi, 601
 calledby saveDependentsHashTable, 1022
 calledby saveHistory, 596
 calledby saveUsersHashTable, 1023
 calledby setHistoryCore, 585
 calledby writeHiFi, 602

 ruleLhsTran, 322
 calledby pfRule2Sex, 318
 calls nsubst, 322
calls patternVarsOf, 322
uses $multiVarPredicateList, 322
uses $predicateList, 322
defun, 322
rulePredicateTran, 319
calledby pfRule2Sex, 318
calls patternVarsOf, 319
calls pvarPredTran, 319
uses $multiVarPredicateList, 319
defun, 319
runspad, 19
calledby spad, 18
calls exit, 19
calls ncTopLevel, 19
calls resetStackLimits, 19
calls seq, 19
uses $quitTag, 19
catches, 19
defun, 19
rwrite, 604
calledby rwrite, 604
calledby saveDependentsHashTable, 1022
calledby saveUsersHashTable, 1022
calledby spadrwrite0, 604
calls identp, 605
calls pname, 604, 605
calls rwrite, 604
defun, 604
safeWritify, 606
calledby spadrwrite0, 604
calls writify, 607
catches, 606
defun, 606
sameMsg?, 376
calledby redundant, 374
calls getMsgArgL, 376
calls getMsgKey, 376
defun, 376
sameUnionBranch, 59
calledby printTypeAndTimeNormal, 57
defun, 59
satBreak
calledby kcPage, 1358
satDownLink
calledby reportAO, 1348
satisfiesRegularExpressions, 943
calledby filterListOfStringsWithFn, 943
calledby filterListOfStrings, 942
calls strpos, 943
defun, 943
satisfiesUserLevel, 451
calledby commandsForUserLevel, 448
calledby displaySetVariableSettings, 657
calledby set1, 809
uses $UserLevel, 451
defun, 451
save-system
calledby spad-save, 990
saveDependentsHashTable, 1022
calledby make-databases, 1019
calls erase, 1022
calls hget, 1022
calls hkeys, 1022
calls msort, 1022
calls rshut, 1022
calls rwrite, 1022
calls writeLib1, 1022
local ref $depTb, 1022
local ref $erase, 1022
defun, 1022
saveHistory, 595
calledby historySpad2Cmd, 582
calls histFileErase, 595
calls histFileName, 595
calls histInputFileName, 595
calls makeHistFileName, 595
calls namestring, 596
calls object2Identifier, 596
calls rdefostream, 595
calls rshut, 596
calls sayKeyedMsg, 595
calls spadrwrite0, 595
calls throwKeyedMsg, 595
calls writeInputLines, 595
uses $HiFiAccess, 596
uses $internalHistoryTable, 596
uses $seen, 596
uses $useInternalHistoryTable, 596
defun, 595
saveMapSig, 853
INDEX

called by trace1, 848
calls addassoc, 853
calls getMapSig, 853
calls rassoc, 853
uses $mapSubNameAlist, 853
uses $tracedMapSignatures, 853
defun, 853
saveSystem, 650
calls helpSpad2Cmd, 650
calls spad-save, 650
defun, 650
saveSystem help page, 649
manpage, 649
saveUsersHashTable, 1022
called by make-databases, 1019
calls erase, 1022
calls hget, 1022
calls hkeys, 1022
calls msort, 1022
calls rshut, 1023
calls rwrite, 1022
calls writeLib1, 1022
local ref $erase, 1023
local ref $usersTb, 1023
defun, 1022
say
called by displaySetVariableSettings, 657
called by newHelpSpad2Cmd, 573
called by trace1, 848
called by whatCommands, 942
called by workfilesSpad2Cmd, 949
say2PerLine
called by displayOperationsFromLisplib, 821
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
sayAsManyPerLineAsPossible
called by apropos, 945
called by displayWorkspaceNames, 454
called by setExposeAddGroup, 699
called by setOutputCharacters, 767
called by whatCommands, 942
sayBrightly
called by ?t, 903
called by bcError, 1247
called by break, 906
called by describeFortPersistence, 757
called by describeInputLibraryArgs, 668
called by describeOutputLibraryArgs, 665
called by describeSetFortDir, 727
called by describeSetFortTmpDir, 725
called by describeSetLinkerArgs, 729
called by describeSetNagHost, 755
called by describeSetOutputAlgebra, 765
called by describeSetOutputFormula, 792
called by describeSetOutputFortran, 772
called by describeSetOutputHtml, 783
called by describeSetOutputMathml, 778
called by describeSetOutputOpenMath, 788
called by describeSetOutputTex, 798
called by displayCondition, 465
called by displayMacros, 538
called by displayMacro, 454
called by displayModemap, 466
called by displayMode, 466
called by displaySetOptionInformation, 655
called by displaySetVariableSettings, 657
called by displayWorkspaceNames, 454
called by finalExactRequest, 1241
called by libConstructorSig, 1407
called by linearFinalRequest, 1241
called by msgOutputter, 353
called by pcounters, 860
called by printLabelledList, 474
called by processKeyedError, 353
called by pspacers, 859
called by ptimers, 859
called by reportOperations, 815
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
called by reportSpadTrace, 892
called by reportUndo, 928
called by reportWhatOptions, 941
called by sayShowWarning, 826
called by setFortDir, 726
called by setFortTmpDir, 724
called by setOutputCharacters, 766
called by traceReply, 899
called by workfilesSpad2Cmd, 950
called by zsystemdevelopment1, 954
sayBrightly1, 1049
called by sayMSG2File, 331
defun, 1049
saybrightly1
   called by saymsg, 331
calls brightprint-0, 1049
calls brightprint, 1050
sayBrightlyLength
   called by traceReply, 899
sayBrightlyNT
   called by bcError, 1247
called by letPrint, 885
called by mkConform, 1374
called by reportUndo, 928
sayExample, 539
defun, 539
sayexample
   calls cleanupline, 540
calls sayNewLine, 540
sayKeyedMsg, 329
   called by abbreviationsSpad2Cmd, 483
called by apropos, 945
called by changeHistListLen, 589
called by clearCmdAll, 503
called by clearCmdCompletely, 502
called by clearCmdParts, 505
called by clearSpad2Cmd, 500
called by commandAmbiguityError, 452
called by commandErrorMessage, 450
called by describeSetStreamsCalculate, 802
called by displayExposedConstructors, 705
called by displayExposedGroups, 705
called by displayFrameNames, 563
called by displayHiddenConstructors, 705
called by displayOperations, 537
called by displayProperties, 461
called by handleNoParseCommands, 470
called by helpSpad2Cmd, 572
called by historySpad2Cmd, 582
called by history, 582
called by importFromFrame, 563
called by listConstructorAbbreviations, 485
called by loadLibNoUpdate, 1037
called by loadLib, 1035
called by load, 629
called by localdatabase, 1014
called by localrllib, 1016
called by newHelpSpad2Cmd, 573
called by npsystem, 472
called by printStatisticsSummary, 56
called by printTypeAndTimeNormal, 57
called by quitSpad2Cmd, 638
called by reportOperations, 815
called by reportOpsFromLispLib, 818
called by restoreHistory, 597
called by saveHistory, 595
called by set1, 809
called by setExposeAddConstr, 700
called by setExposeAddGroup, 699
called by setExposeAdd, 698
called by setExposeDropConstr, 703
called by setExposeDropGroup, 702
called by setExposeDrop, 701
called by setExpose, 697
called by setHistoryCore, 585
called by setOutputAlgebra, 763
called by setOutputFormula, 790
called by setOutputFortran, 770
called by setOutputHtml, 781
called by setOutputMathml, 776
called by setOutputOpenMath, 786
called by setOutputTex, 797
called by showSpad2Cmd, 814
called by spadStartUpMsgs, 17
called by trace1, 848
called by undoInCore, 593
called by userLevelErrorMessage, 451
called by whatCommands, 942
called by whatSpad2Cmd, 940
called by workfilesSpad2Cmd, 949
called by writeInputLines, 587
called by writifyComplain, 606
calls sayKeyedMsgLocal, 329
uses $texFormatting, 329
defun, 329
uses $displayMsgNumber, 330
called by spadUntrace, 895
called by traceDomainLocalOps, 880
called by traceReply, 899
called by untraceDomainLocalOps, 880
called by spadUntrace, 895
uses $linepos, 134
called by scanNumber, 132
called by commandAmbiguityError, 452
called by displayProperties, sayFunctionDeps, 458
called by displayProperties, 462
called by displaySetOptionInformation, 655
called by displayType, 460
called by displayValue, 459
called by evalDomain, 913
called by getAndSay, 461
called by initializeSetVariables, 653
called by sayKeyedMsgLocal, 330
called by scanComment, 116
called by set1, 809
called by setExposeAddGroup, 699
called by setExposeAdd, 698
called by setExposeDropConstr, 704
called by setExposeDropGroup, 702
called by setExposeDrop, 701
called by setExpose, 697
called by showInOut, 600
called by showInput, 599
called by spadStartUpMsgs, 17
sayMSG, 331
saymsg
sayMSG2File, 331
sayNewLine
sayexample, 540
sayShowWarning, 826
called by reportOpsFromLisplib1, 817
called by reportOpsFromUnitDirectly1, 825
called by sayBrightly, 826
called by scanCheckRadix, 134
called by scanNumber, 132
called by scanComment, 116
called by scanToken, 114
called by scanKeyTr, 120
called by scanToken, 114
called by scanKeyTr, 120
called by scanToken, 114
calls hkeys, 137
defun, 137
scanError, 135
called by scanPunct, 118
called by scanToken, 114
calls lfeerror, 135
calls lnExtraBlanks, 135
calls ncSoftError, 135
uses $linepos, 135
uses $ln, 135
uses $n, 135
defun, 135
scanEsc, 123
called by scanEscape, 135
called by scanEsc, 124
called by scanS, 131
called by scanW, 128
called by spleI1, 122
calls nextline, 124
calls qenum, 124
calls scanEsc, 124
calls startsComment?, 124
calls startsNegComment?, 124
calls strposl, 124
uses $ln, 124
uses $n, 124
uses $r, 124
uses $sz, 124
defun, 123
scanEscape, 135
called by scanToken, 114
calls scanEsc, 135
calls scanWord, 135
uses $n, 135
defun, 135
scanExponent, 126
called by scanNumber, 132
called by scanPossFloat, 121
calls concat, 126
calls digit?, 126
calls lffloat, 126
calls qenum, 126
calls spleI, 126
uses $ln, 126
uses $n, 126
uses $sz, 126
defun, 126
scanIgnoreLine, 113
called by lineoftoks, 111
calls incPrefix?, 113
calls qenum, 113
defun, 113
scanInsert, 138
calls qenum, 138
defun, 138
scanKeyTable, 136
defvar, 136
scanKeyTableCons, 136
defun, 136
scanKeyTr, 120
called by scanPunct, 118
calls keyword, 120
calls lfkey, 120
calls scanCloser?, 120
calls scanPossFloat, 120
uses $floatok, 120
defun, 120
scanKeyWords, 108
defvar, 108
scanNegComment, 117
called by scanToken, 114
calls lfnegcomment, 117
calls substring, 117
uses $ln, 117
uses $n, 117
uses $sz, 117
defun, 117
scanNumber, 132
called by scanToken, 114
calls concat, 132
calls lffinteger, 132
calls lfrinteger, 132
calls qenum, 132
calls scanCheckRadix, 132
calls scanExponent, 132
calls spleI1, 132
calls spleI, 132
uses $floatok, 132
uses $ln, 132
uses $n, 132
uses $sz, 132
defun, 132
ScanOrPairVec, 616
  called by dewritify, 615
  called by writify, 610
  calls ScanOrPairVec,ScanOrInner, 616
  uses $seen, 616
  catches, 616
  defun, 616

ScanOrPairVec,ScanOrInner, 615
  called by ScanOrPairVec,ScanOrInner, 615
  called by ScanOrPairVec, 616
  calls ScanOrPairVec,ScanOrInner, 615
  calls hget, 615
  calls hput, 615
  calls qcar, 615
  calls qcdr, 615
  calls vcep, 615
  uses $seen, 615
  defun, 615
  throws, 615

scanPossFloat, 121
  called by scanKeyTr, 120
  calls digit?, 121
  calls lfkey, 121
  calls scanExponent, 121
  calls spleI, 121
  uses $ln, 121
  uses $n, 121
  uses $sz, 121
  defun, 121

scanPun, 139
  defvar, 139

scanPunCons, 139
  calls hkeys, 139
  defun, 139

scanPunct, 118
  called by scanToken, 114
  calls scanError, 118
  calls scanKeyTr, 118
  calls subMatch, 118
  uses $ln, 118
  uses $n, 118
  defun, 118

scanS, 131
  called by scanString, 130
  called by scanS, 131
  calls concat, 131
  calls lnExtraBlanks, 131
  calls ncSoftError, 131
  calls scanEsc, 131
  calls scanS, 131
  calls scanTransform, 131
  calls strpos, 131
  calls substring, 131
  uses $linepos, 131
  uses $ln, 131
  uses $n, 131
  uses $sz, 131
  defun, 131

scanSpace, 129
  called by scanToken, 114
  calls lfspaces, 129
  calls strposl, 129
  uses $noatok, 129
  uses $ln, 129
  uses $n, 129
  defun, 129

scanString, 130
  called by scanToken, 114
  calls lfstring, 130
  calls scanS, 130
  uses $noatok, 130
  uses $n, 130
  defun, 130

scanToken, 114
  calls constoken, 114
  calls digit?, 114
  calls dqUnit, 114
  calls lfid, 114
  calls lnExtraBlanks, 114
  calls punctuation?, 114
  calls qenum, 114
  calls scanComment, 114
  calls scanError, 114
  calls scanEscape, 114
  calls scanNegComment, 114
  calls scanNumber, 114
  calls scanPunct, 114
  calls scanSpace, 114
  calls scanString, 114
  calls scanWord, 114
  calls startsComment?, 114
  calls startsId?, 114
calls startsNegComment?, 114
uses $linepos, 114
uses $in, 114
uses $n, 114
defun, 114
scanTransform, 132
calledby scanS, 131
defun, 132
scanW, 128
calledby scanWord, 126
calledby scanW, 128
calls concat, 128
calls idChar?, 128
calls posend, 128
calls qenum, 128
calls scanEsc, 128
calls scanW, 128
calls substring, 128
uses $ln, 128
uses $n, 128
uses $sz, 128
defun, 128
scanWord, 126
calledby scanEscape, 135
calledby scanToken, 114
calls keyword?, 126
calls lfd, 126
calls lfkey, 126
calls scanW, 126
uses $floatok, 126
defun, 126
search, 964
calledby getProplist, 964
calls searchCurrentEnv, 964
calls searchTailEnv, 964
defun, 964
searchCurrentEnv, 964
calledby search, 964
calls assq, 964
calls kdr, 964
defun, 964
searchTailEnv, 965
calledby search, 964
calls assq, 965
calls kdr, 965
defun, 965
sec, 1104
defun, 1104
sech, 1106
defun, 1106
segmentKeyedMsg, 330
calledby getMsgInfoFromKey, 366
calledby msgText, 60
calledby sayKeyedMsgLocal, 330
calls string2Words, 330
defun, 330
selectOption, 479
calledby selectOptionLC, 479
calledby set1, 809
calledby systemCommand, 448
calledby unAbbreviateKeyword, 469
calls identp, 479
calls member, 479
calls pname, 480
calls qcar, 480
calls qcdr, 480
calls stringPrefix?, 479
defun, 479
selectOptionLC, 479
calledby abbreviationsSpad2Cmd, 483
calledby clearCmdParts, 505
calledby clearSpad2Cmd, 500
calledby close, 510
calledby describeSpad2Cmd, 529
calledby displaySpad2Cmd, 536
calledby frameSpad2Cmd, 566
calledby getTraceOption, 854
calledby historySpad2Cmd, 582
calledby newHelpSpad2Cmd, 573
calledby readSpad2Cmd, 642
calledby reportOpsFromLisplib, 818
calledby reportOpsFromUnitDirectly, 822
calledby setExposeAdd, 698
calledby setExposeDrop, 702
calledby setExpose, 697
calledby setInputLibrary, 667
calledby synonymsForUserLevel, 833
calledby systemCommand, 448
calledby trace1, 848
calledby unAbbreviateKeyword, 469
calledby whatSpad2Cmd, 940
calledby workfilesSpad2Cmd, 949
called by `zsystemdevelopment1`, 954
calls downcase, 479
calls object2Identifier, 479
calls selectOption, 479
defun, 479

`sendHTErrorSignal`
called by `protectedEVAL`, 45

`separatePiles`, 341
called by `pileCforest`, 340
called by `separatePiles`, 341
calls `dqConcat`, 341
calls `dqUnit`, 341
calls `lastTokPosn`, 341
calls `separatePiles`, 341
calls `tokConstruct`, 341
defun, 341

`SEQ`
called by `funfind,LAM`, 874

`seq`
called by `/tracereply`, 894
called by `abbreviationsSpad2Cmd`, 483
called by `apropos`, 945
called by `clearCmdParts`, 505
called by `coerceSpadArgs2E`, 865
called by `dewritify,dewritifyInner`, 612
called by `difAlist`, 925
called by `displayMacros`, 538
called by `displayProperties, sayFunctionDeps`, 458
called by `displayProperties`, 462
called by `flattenOperationAlist`, 883
called by `getAliasIfTracedMapParameter`, 890
called by `getBpiNameIfTracedMap`, 890
called by `getPreviousMapSubNames`, 870
called by `getTraceOption,ln`, 853
called by `getTraceOptions`, 852
called by `getTraceOption`, 854
called by `getWorkspaceNames`, 455
called by `historySpad2Cmd`, 583
called by `importFromFile`, 563
called by `isListOfIdentifiersOrStrings`, 869
called by `isListOfIdentifiers`, 868
called by `orderBySlotNumber`, 893
called by `prTraceNames, fn`, 898
called by `prTraceNames`, 898
called by `recordFrame`, 923
called by `removeUndoLines`, 933
called by `reportUndo`, 928
called by `runspad`, 19
called by `set1`, 809
called by `spadReply, printName`, 894
called by `spadReply`, 895
called by `spadTrace, isTraceable`, 875
called by `spadTrace`, 876
called by `spadUntrace`, 896
called by `subTypes`, 866
called by `trace1`, 848
called by `traceDomainConstructor`, 880
called by `traceReply`, 899
called by `undoFromFile`, 594
called by `undoLocalModemapHack`, 933
called by `undoSingleStep`, 931
called by `untraceDomainConstructor, keepTraced?`, 882

called by `untraceDomainConstructor`, 883
called by `whatConstructors`, 945
called by `whatSpad2Cmd`, 940
called by `writify, writifyInner`, 607

called by `intloopReadConsole`, 28
called by `addNewInterpreterFrame`, 42
called by `changeToNamedInterpreterFrame`, 42
called by `executeQuietCommand`, 43
called by `is-console[9]`, 43
called by `lassoc`, 42
called by `mkprompt`, 42
called by `parseAndInterpret`, 43
called by `protectedEVAL`, 43
called by `read-line`, 42
called by `serverSwitch`, 43
called by `sockGetInt`, 42
called by `sockGetString`, 42
called by `sockSendInt`, 42
called by `sockSendString`, 42
called by `unescapeStringsInForm`, 43
uses *eof*, 43
uses $CallInterp`, 43
uses $CreateFrameAnswer`, 43
uses $CreateFrame`, 43
uses $EndOfOutput`, 43
uses $EndServerSession`, 43
uses $EndSession, 43
uses $KillLispSystem, 43
uses $LispCommand, 43
uses $MenuServer, 43
uses $NeedsToSignalSessionManager, 43
uses $NonSmanSession, 43
uses $QuietSpadCommand, 43
uses $SessionManager, 43
uses $SpadCommand, 43
uses $SpadServer, 43
uses $SwitchFrames, 43
uses $currentFrameNum, 43
uses $frameAlist, 43
uses $frameNumber, 43
uses $sockBufferLength, 43
uses in-stream, 43
catches, 42
defun, 42
serverSwitch
called by serverReadLine, 43
set, 808
called by browse, 493
calls set1, 808
uses $setOptions, 808
defun, 808
set help page, 651
manpage, 651
set-hole-size
called by init-memory-config, 33
set-restart-hook, 13
defun, 13
set1, 808
called by set1, 809
called by set, 808
calls bright, 809
calls displaySetOptionInformation, 809
calls displaySetVariableSettings, 809
calls downcase, 809
calls exit, 809
calls kdr, 809
calls lassoc, 809
calls literals, 809
calls object2String, 809
calls poundsign, 809
calls satisfiesUserLevel, 809
calls sayKeyedMsg, 809
calls sayMSG, 809
calls sayMessage, 809
calls selectOption, 809
calls seq, 809
calls set1, 809
calls translateYesNo2TrueFalse, 809
calls tree, 809
calls use-fast-links, 809
calls $UserLevel, 809
uses $displaySetValue, 809
uses $setOptionNames, 809
defun, 808
setAref2U16, 1059
defmacro, 1059
setAref2U32, 1061
defmacro, 1061
setAref2U8, 1058
defmacro, 1058
setCurrentLine, 40
called by intloopEchoParse, 67
called by intloopProcessString, 36
called by intloopProcess, 62
called by intloopReadConsole, 28
uses $currentLine, 40
defun, 40
setdatabase, 1009
called by abbreviationsSpad2Cmd, 483
calls make-database, 1009
defun, 1009
setdifference
called by displayWorkspaceNames, 454
called by untraceMapSubNames, 873
setq
called by setExposeAddConstr, 700
called by setExposeAddGroup, 699
called by setExposeDropConstr, 704
called by setExposeDropGroup, 702
setqU16, 1056
defmacro, 1056
setqU32, 1057
defmacro, 1057
setqU8, 1055
defmacro, 1055
setqExpose, 697
called by setExpose, 697
calls displayExposedConstructors, 697
calls displayExposedGroups, 697
calls displayHiddenConstructors, 697
calls namestring, 697
calls pathname, 697
calls qcar, 697
calls qcdr, 697
calls sayKeyedMsg, 697
calls sayMSG, 697
calls selectOptionLC, 697
calls setExposeAdd, 697
calls setExposeDrop, 697
defun, 697
setExposeAdd, 698
calledby setExposeAdd, 698
calledby setExpose, 697
calls centerAndHighlight, 698
calls displayExposedConstructors, 698
calls displayExposedGroups, 698
calls qcar, 698
calls qcdr, 698
calls sayKeyedMsg, 698
calls sayMSG, 698
calls selectOptionLC, 698
calls setExposeAddConstr, 698
calls setExposeAddGroup, 698
calls setExposeAdd, 698
calls specialChar, 698
uses $linelength, 698
defun, 698
setExposeAddConstr, 700
calledby localnrlib, 1015
calledby setExposeAdd, 698
calls centerAndHighlight, 701
calls clearClams, 701
calls delete, 700
calls displayExposedConstructors, 701
calls getdatabase, 700
calls member, 700
calls msort, 700
calls qcar, 700
calls sayKeyedMsg, 700
calls setelt, 700
calls specialChar, 701
calls unabbrev, 700
uses $interpreterFrameName, 701
uses $linelength, 701
uses $localExposureData, 701
defun, 700
setExposeAddGroup, 699
calledby setExposeAdd, 698
calls centerAndHighlight, 699
calls clearClams, 699
calls displayExposedConstructors, 699
calls displayExposedGroups, 699
calls displayHiddenConstructors, 699
calls getalist, 699
calls member, 699
calls msort, 699
calls namestring, 699
calls object2String, 699
calls pathname, 699
calls qcar, 699
calls sayAsManyPerLineAsPossible, 699
calls sayKeyedMsg, 699
calls sayMSG, 699
calls setelt, 699
calls specialChar, 699
uses $globalExposureGroupAlist, 699
uses $interpreterFrameName, 699
uses $linelength, 699
uses $localExposureData, 699
defun, 699
setExposeDrop, 701
calledby setExposeDrop, 702
calledby setExpose, 697
calls centerAndHighlight, 701
calls displayHiddenConstructors, 701
calls qcar, 701
calls qcdr, 701
calls sayKeyedMsg, 701
calls sayMSG, 701
calls selectOptionLC, 702
calls setExposeDropConstr, 702
calls setExposeDropGroup, 702
calls setExposeDrop, 702
calls specialChar, 701
uses $linelength, 702
defun, 701
setExposeDropConstr, 703
calledby setExposeDrop, 702
calls centerAndHighlight, 704
calls clearClams, 704
calls delete, 704
calls displayExposedConstructors, 704
calls displayHiddenConstructors, 704
calls getdatabase, 703
calls member, 704
calls msort, 704
calls qcar, 703
calls sayKeyedMsg, 703
calls sayMSG, 704
calls setelt, 704
calls specialChar, 704
uses $interpreterFrameName, 704
uses $linelength, 704
uses $localExposureData, 704
defun, 703
setExposeDropGroup, 702
calledby setExposeDrop, 702
calls centerAndHighlight, 702
calls clearClams, 702
calls delete, 702
calls displayExposedConstructors, 702
calls displayExposedGroups, 702
calls displayHiddenConstructors, 702
calls getalist, 702
calls member, 702
calls qcar, 702
calls sayKeyedMsg, 702
calls sayMSG, 702
calls setelt, 702
calls specialChar, 702
uses $globalExposureGroupAlist, 703
uses $interpreterFrameName, 703
uses $linelength, 702
uses $localExposureData, 703
defun, 702
setFortDir, 726
calls bright, 726
calls describeSetFortDir, 726
calls pname, 726
calls sayBrightly, 726
calls validateOutputDirectory, 726
uses $fortranDirectory, 726
defun, 726
setFortPers, 756
calls bright, 756
calls describeFortPersistence, 756
calls sayMessage, 756
calls terminateSystemCommand, 756
uses $fortPersistence, 756
defun, 756
setFortTmpDir, 724
calls bright, 724
calls describeSetFortTmpDir, 724
calls pname, 724
calls sayBrightly, 724
calls validateOutputDirectory, 724
uses $fortranTmpDir, 724
defun, 724
setFunctionsCache, 708
defun, 708
setHistoryCore, 584
calledby historySpad2Cmd, 582
calls boot-equal, 585
calls disableHist, 585
calls histFileErase, 585
calls histFileName, 585
calls object2Identifier, 585
calls rdefiostream, 585
calls readHiFi, 585
calls rkeyids, 585
calls rshut, 585
calls sayKeyedMsg, 585
calls spadrwrite, 585
uses $HiFiAccess, 585
uses $IOindex, 585
uses $internalHistoryTable, 585
uses $useInternalHistoryTable, 585
defun, 584
setInputLibrary, 667
calledby setInputLibrary, 667
calls addInputLibrary, 667
calls describeInputLibraryArgs, 667
calls dropInputLibrary, 667
calls qcar, 667
calls qcdr, 667
calls selectOptionLC, 667
calls setInputLibrary, 667
uses input-libraries, 667
defun, 667
setIOindex, 599
INDEX

uses $IOindex, 599
defun, 599
setletprintflag
called by breaklet, 905
called by spadTrace, 877
called by spaduntrace, 896
called by tracelet, 904
setLinkerArgs, 728
calls describeSetLinkerArgs, 728
calls object2String, 728
uses $fortranLibraries, 728
defun, 728
setMsgCatlessAttr, 365
called by setMsgforcedAttr, 364
called by setMsgunforcedAttr, 367
calls ifcdr, 365
calls ncAlist, 365
calls ncPutQ, 365
calls qassq, 365
defun, 365
setMsgForcedAttr, 364
called by setMsgForcedAttrList, 364
calls ncPutQ, 364
calls setMsgCatlessAttr, 364
defun, 364
setMsgForcedAttrList, 364
called by msgCreate, 364
calls msgCreate, 364
calls setMsgForcedAttr, 364
calls whichCat, 364
defun, 364
setMsgPrefix, 349
called by processChPosesForOneLine, 349
defun, 349
setMsgText, 349
called by putDatabaseStuff, 349
called by putFTText, 349
defun, 349
setMsgUnforcedAttr, 367
called by initInPr, 367
called by initToWhere, 368
called by setMsgUnforcedAttrList, 366
calls ncAlist, 367
calls ncPutQ, 367
calls qassq, 367
calls setMsgCatlessAttr, 367
defun, 367
setMsgUnforcedAttrList, 366
called by putDatabaseStuff, 365
calls setMsgUnforcedAttr, 366
calls whichCat, 366
defun, 366
setNagHost, 755
calls describeSetNagHost, 755
calls object2String, 755
uses $nagHost, 755
defun, 755
setOutputAlgebra, 763
called by describeSetOutputAlgebra, 765
called by spad, 18
calls $filep, 763
calls concat, 763
calls defiostream, 763
calls describeSetOutputAlgebra, 763
calls make-outstream, 763
calls member, 763
calls object2String, 763
calls pathnameDirectory, 763
calls pathnameName, 763
calls pathnameType, 763
calls qcar, 763
calls qcdr, 763
calls sayKeyedMsg, 763
calls shut, 763
calls upcase, 763
uses $algebraFormat, 763
uses $algebraOutputFile, 763
uses $algebraOutputStream, 763
uses $filep, 763
defun, 763
setOutputCharacters, 766
called by setOutputCharacters, 767
called by spad, 18
calls bright, 766
calls concat, 766
calls downcase, 767
calls pname, 767
calls qcar, 767
calls qcdr, 767
calls sayAsManyPerLineAsPossible, 767
calls sayBrightly, 766
calls sayMessage, 766
calls setOutputCharacters, 767
calls specialChar, 767
uses $RTspecialCharacters, 767
uses $plainRTspecialCharacters, 767
uses $specialCharacterAlist, 767
uses $specialCharacters, 767
defun, 766

setOutputFormula, 790
calledby describeSetOutputFormula, 792
calls $filep, 790
calls concat, 790
calls defiostream, 790
calls describeSetOutputFormula, 790
calls make-outstream, 790
calls member, 790
calls object2String, 790
calls pathnameDirectory, 790
calls pathnameName, 790
calls pathnameType, 790
calls qcar, 790
calls qcdr, 790
calls sayKeyedMsg, 790
calls shut, 790
calls upcase, 790
uses $filep, 790
uses $formulaFormat, 791
uses $formulaOutputFile, 790
uses $formulaOutputStream, 790
defun, 790

setOutputFortran, 770
calledby describeSetOutputFortran, 772
calls $filep, 770
calls concat, 770
calls defiostream, 770
calls describeSetOutputFortran, 770
calls make-stream, 770
calls member, 770
calls object2String, 770
calls pathnameDirectory, 770
calls pathnameName, 770
calls pathnameType, 770
calls qcar, 770
calls qcdr, 770
calls sayKeyedMsg, 770
calls shut, 770
calls upcase, 770
uses $filep, 770
uses $fortranFormat, 770
uses $fortranOutputFile, 770
uses $fortranOutputStream, 770
defun, 770

setOutputHtml, 781
calledby describeSetOutputHtml, 783
calls $filep, 781
calls concat, 781
calls defiostream, 781
calls describeSetOutputHtml, 781
calls make-outstream, 781
calls member, 781
calls object2String, 781
calls pathnameDirectory, 781
calls pathnameName, 781
calls pathnameType, 781
calls qcar, 781
calls qcdr, 781
calls sayKeyedMsg, 781
calls shut, 781
calls upcase, 781
uses $filep, 781
uses $htmlFormat, 781
uses $htmlOutputFile, 781
uses $htmlOutputStream, 781
defun, 781

setOutputLibrary, 664
calledby describeOutputLibraryArgs, 664
calls $filep, 664
calls openOutputLibrary, 664
calls poundsign, 664
uses $outputLibraryName, 664
defun, 664

setOutputMathml, 776
calledby describeSetOutputMathml, 778
calls $filep, 776
calls concat, 776
calls defiostream, 776
calls describeSetOutputMathml, 776
calls make-outstream, 776
calls member, 776
calls object2String, 776
calls pathnameDirectory, 776
calls pathnameName, 776
calls pathnameType, 776
calls qcar, 776
calls qcdr, 776
INDEX

1657

calls sayKeyedMsg, 776
calls shut, 776
calls upcase, 776
uses $filep, 777
uses $mathmlFormat, 777
uses $mathmlOutputStream, 776
defun, 776

setOutputOpenMath, 785
calledby describeSetOutputOpenMath, 788
calls $filep, 786
calls concat, 785
calls defostream, 785
calls describeSetOutputOpenMath, 785
calls make-outstream, 786
calls member, 786
calls object2String, 786
calls pathnameDirectory, 786
calls pathnameName, 786
calls pathnameType, 786
calls qcar, 786
calls qcdr, 785
calls sayKeyedMsg, 786
calls shut, 786
calls upcase, 786
uses $filep, 786
uses $openMathFormat, 786
uses $openMathOutputStream, 786
defun, 785

setOutputTex, 796
calledby describeSetOutputTex, 798
calls $filep, 797
calls concat, 796
calls defostream, 796
calls describeSetOutputTex, 796
calls make-outstream, 797
calls member, 797
calls object2String, 797
calls pathnameDirectory, 797
calls pathnameName, 797
calls pathnameType, 797
calls qcar, 796
calls qcdr, 796
calls sayKeyedMsg, 797
calls shut, 797
calls upcase, 797
uses $filep, 797
uses $texFormat, 797
uses $texOutputFile, 797
uses $texOutputStream, 797
defun, 796

setStreamsCalculate, 801
calls bright, 802
calls describeSetStreamsCalculate, 801
calls object2String, 801
calls sayMessage, 801
calls terminateSystemCommand, 802
uses $streamCount, 802
defun, 801

setUpDefault, 1275
defun, 1275

shortenForPrinting, 891
calledby letPrint, 885
calls devaluate, 891
calls terminateSystemCommand, 891
defun, 891

show, 814
calls showSpad2Cmd, 814
defun, 814

show help page, 813
manpage, 813

showdatabase, 1008
calls getdatabase, 1008
defun, 1008

showHistory
calledby historySpad2Cmd, 582

showInOut, 600
calls assq, 600
calls disableHist, 600
calls objMode, 600
calls readHiFi, 600
calls sayMSG, 600
calls spadPrint, 600
defun, 600

showInput, 599
calls disableHist, 599
calls readHiFi, 599
calls sayMSG, 599
calls tab, 599
defun, 599
showMsgPos?, 357
called by getPosStL, 356
calls leader?, 358
calls msgImPr?, 357
uses $erMsgToss, 358
defun, 357

showSpad2Cmd, 814
called by show, 814
calls helpSpad2Cmd, 814
calls member, 814
calls qcar, 814
calls reportOperations, 814
calls sayKeyedMsg, 814
uses $InteractiveFrame, 815
uses $env, 815
uses $e, 815
uses $options, 815
uses $showOptions, 815
defun, 814

shut, 982
called by reportOpsFromLisplib1, 817
called by reportOpsFromUnitDirectly1, 825
called by sayMSG2File, 331
called by setOutputAlgebra, 763
called by setOutputFormula, 790
called by setOutputFortran, 770
called by setOutputHtml, 781
called by setOutputMathml, 776
called by setOutputOpenMath, 786
called by setOutputTex, 797
called by writeInputLines, 587
called by zsystemdevelopment1, 954
calls is-console[9], 982
defun, 982

simpCatPredicate
called by dbSearchOrder, 1356

simpHasPred
called by domainDescendantsOf, 1349
called by kTestPred, 1386
called by kePage, 1352
called by reduceAlistForDomain, 1362

defun, 1362

size, 1045
called by abbreviationsSpad2Cmd, 483
called by getPreStL, 355
called by isgenvar, 886
called by makeMsgFromLine, 371
called by processChPosesForOneLine, 376
called by processSynonyms, 32
called by substringMatch, 119
called by writeInputMatch, 587
defun, 1045

SkipEnd?, 78
called by incLude1, 80
calls remainder, 78
defvar, 78

SkipPart?, 79
called by incLude1, 80
calls remainder, 79
defvar, 79

Skipping?, 79
called by incLude1, 80
calls KeepPart?, 79
defvar, 79

sockGetInt
called by queryClients, 510
called by serverReadLine, 42
sockGetString

called by executeQuietCommand, 46
called by serverReadLine, 42
sockSendInt
called by close, 510
called by queryClients, 510
called by serverReadLine, 42
sockSendString
called by serverReadLine, 42

sortby
called by workfilesSpad2Cmd, 949

SPACE, 105
defvar, 105

spad, 18
called by restart, 15
calls runspad, 18
calls setOutputAlgebra, 18
uses $PrintCompilerMessageIfTrue, 18
defun, 18

spad-error-loc, 970
called by spad-long-error, 969
defun, 970

spad-long-error, 969
called by spad-syntax-error, 969
calls iostat, 969
calls spad-error-loc, 969
uses out-stream, 969
defun, 969
spad-save, 989
called by savesystem, 650
calls save-system, 990
uses $SpadServer, 990
uses $openServerIfTrue, 990
defun, 989
spad-short-error, 970
called by spad-synt-ex-error, 969
calls line-past-end-p, 970
calls line-print, 970
uses $current-line, 970
defun, 970
spad-syntax-error, 969
calls bumperrorcount, 969
calls consoleinputpt, 969
calls ioclear, 969
calls spad-long-error, 969
calls spad-short-error, 969
uses debugmode, 969
defun, 969
throws, 969
spad2BootCoerce, 1066
defun, 1066
spadcall
called by basicLookupCheckDefaults, 1078
called by basicLookup, 1076
called by clearCmdSortedCaches, 501
called by letPrint3, 888
called by lookupInDomainVector, 1078
spadClosure?, 611
called by writify,writifyInner, 607
calls bpiname, 611
calls qcar, 611
calls qcdr, 611
calls vceps, 611
defun, 611
spadConstant, 1155
defmacro, 1155
spaddifference
called by changeHistListLen, 589
called by charDigitVal, 617
called by dewritify,dewritifyInner, 612
called by displaySetVariableSettings, 657
called by fetchOutput, 600
called by getAliasIfTracedMapParameter, 889
called by removeUndoLines, 934
called by undoCount, 929
called by undoInCore, 592
called by undoSteps, 930
called by undo, 922
called by writeInputLines, 587
spaderrorstream
used by init-boot/spad-reader, 968
used by spad-long-error, 969
SpadInterpretStream, 25
called by intloop, 24
calls intloopInclude, 27
calls intloopReadConsole, 27
calls mkprompt, 27
uses $erMsgToss, 27
uses $fn, 27
uses $inclAssertions, 27
uses $lastPos, 27
uses $libQuiet, 27
uses $ncMsgList, 27
uses $newcompErrorCount, 27
uses $nopos, 27
uses $okToExecuteMachineCode, 27
uses $promptMsg, 27
uses $systemCommandFunction, 27
defun, 25
spadPrint
called by showInOut, 600
spadReply, 895
called by spadTrace, 877
called by spadUntrace, 896
calls exit, 895
calls seq, 895
calls spadReply,printName, 895
uses /tracenames, 895
defun, 895
spadReply,printName, 894
called by spadReply, 895
calls devaluate, 894
calls exit, 894
calls isDomainOrPackage, 894
calls qcar, 894
calls seq, 894
defun, 894
SPADRREAD
calls dewritify, 605
calls rread, 605
spadrread, 605
calledby readHiFi, 601
defun, 605
spadrwrite, 605
calledby setHistoryCore, 585
calledby writeHiFi, 602
calls spadrwrite0, 605
calls throwKeyedMsg, 605
defun, 605
spadrwrite0, 604
calledby saveHistory, 595
calledby spadrwrite, 605
calls rwrite, 604
calls safeWritify, 604
defun, 604
spadStartUpMsgs, 17
calledby restart, 15
calls fillerSpaces, 17
calls sayKeyedMsg, 17
calls sayMSG, 17
calls specialChar, 17
uses *build-version*, 17
uses *yearweek*, 17
uses $linelength, 17
uses $msgAlist, 17
uses $opSysName, 17
defun, 17
spadsysnamep
calledby coerceTraceArgs2E, 864
calledby coerceTraceFunValue2E, 867
spadThrow
calledby pf2Sex1, 301
calledby tersyscommand, 452
calledby throwEvalTypeMsg, 919
spadTrace, 876
calledby traceDomainConstructor, 881
calls aldrTrace, 876
calls as-insert, 876
calls assoc, 876
calls bpiname, 877
calls bptrace, 877
calls constructSubst, 877
calls exit, 876
calls flattenOperationAlist, 876
calls getOperationAlistFromLispLib, 876
calls getOption, 876
calls isDomainOrPackage, 876
calls kdr, 876
calls opOf, 876
calls printDashedLine, 877
calls refvecp, 876
calls removeOption, 876
calls reportSpadTrace, 877
calls rplac, 877
calls seq, 876
calls setletprintflag, 877
calls spadReply, 877
calls spadTrace,g, 876
calls spadTrace,isTraceable, 876
calls spadTraceAlias, 877
calls subTypes, 877
calls userError, 876
uses /tracenames, 877
uses $fromSpadTrace, 877
uses $letAssoc, 877
uses $reportSpadTrace, 877
uses $traceNoisely, 877
uses $tracedModemap, 877
defun, 876
spadTrace,g, 875
calledby spadTrace, 876
defun, 875
spadTrace,isTraceable, 875
calledby spadTrace, 876
calls bpiname, 876
calls exit, 876
calls gensymp, 876
calls reportSpadTrace, 876
calls seq, 875
defun, 875
spadTraceAlias, 891
calledby spadTrace, 877
calls internl, 891
defun, 891
spadUntrace, 895
calls assoc, 895
calls bpiname, 896
calls bpiuntrace, 896
INDEX

1661

calls bright, 896
calls delasc, 896
calls devaluate, 895
calls exit, 896
calls getOption, 895
calls isDomainOrPackage, 895
calls prefix2String, 896
calls remover, 896
calls rplac, 896
calls sayMSG, 895
calls seq, 896
calls setletprintAg, 896
calls spadReply, 896
calls userError, 895
uses /tracenames, 896
uses $letAssoc, 896
defun, 895
specialChar, 980
calledby displayOperationsFromLisplib, 820
calledby displaySetOptionInformation, 655
spool help page, 827
calledby displaySetVariableSettings, 657
manpage, 827
calledby filterAndFormatConstructors, 944

calledby printSynonyms, 474
calledby doSystemCommand, 446

calledby reportOpsFromLisplib, 818

calledby reportOpsFromUnitDirectly, 822
calledby setExposéAddConstr, 701
calledby setExposéAddGroup, 699
calledby setExposéAdd, 698
calledby setExposéDropConstr, 704

calledby setExposéDropGroup, 702

calledby setExposéDrop, 701
calledby setOutputCharacters, 767

calledby spadStartUpMsgs, 17
calledby whatCommands, 942
calledby workfilesSpad2Cmd, 949
calls assq, 980
calls ifcdr, 980
uses $specialCharacterAlist, 980
uses $specialCharacters, 980
defun, 980

spleI, 122
calledby scanExponent, 126
calledby scanNumber, 132
calledby scanPossFloat, 121
calls spleI1, 122
defun, 122

calledby scanNumber, 132
calledby spleI1, 122
calledby spleI, 122
calls concat, 122
calls qenum, 122
calls scanEsc, 122
calls spleI1, 122
calls substring, 122
uses $ln, 122
uses $n, 123
uses $sz, 123
defun, 122

splitConTable
calledby dbShowConditions, 1394
splitIntoOptionBlocks, 447
calledby doSystemCommand, 446
calls stripSpaces, 447
defun, 447

stackTraceOptionError, 861
calledby getTraceOption, 854
calledby getTraceOption, 854

calledby traceOptionError, 857
calledby transOnlyOption, 860
uses $traceErrorStack, 861
defun, 861

startsComment?, 115
calledby scanEsc, 124
calledby scanToken, 114
calls qenum, 115
uses $ln, 115
uses $n, 115
uses $sz, 115
defun, 115

startsId?, 1044
calledby scanEsc, 124
calledby scanToken, 114
defmacro, 1044

startsNegComment?, 117
calledby scanEsc, 124
calledby scanToken, 114
calls qenum, 117
uses $ln, 117
uses $n, 117
uses $sz, 117
defun, 117

startTimingProcess
called by evalDomain, 913
called by loadLib, 1035
called by localnrlib, 1015
called by processInteractive1, 50
called by recordNewValue, 591
called by recordOldValue, 592
called by updateHist, 589

statisticsInitialization, 1035
called by restart, 15
calls gbe-time, 1035
defun, 1035

statisticsSummary
called by printStatisticsSummary, 56

startTimingProcess
called by evalDomain, 913
called by interpretTopLevel, 51
called by loadLibNoUpdate, 1037
called by loadLib, 1036
called by processInteractive1, 50
called by recordNewValue, 591
called by recordOldValue, 592
called by updateHist, 589

strconc
called by bcComplexLimitGen, 1225
called by bcDefiniteIntegrateGen, 1192
called by bcDifferentiateGen, 1196
called by bcDraw2DSolveGen, 1203
called by bcDraw2DfunGen, 1199
called by bcDrawDparGen, 1201
called by bcDraw3DfunGen, 1205
called by bcDraw3DparGen, 1210
called by bcDrawIt, 1247
called by bcGenEquations, 1242
called by bcGen, 1244, 1245
called by bcIndefiniteIntegrateGen, 1190
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1186
called by bcInputMatrixByFormulaGen, 1185
called by bcLinearMatrixGen, 1240
called by bcLinearSolveMatrixInhomo, 1238

called by bcMakeEquations, 1234
called by bcMakeLinearEquations, 1234
called by bcMatrixGen, 1188
called by bcProductGen, 1195
called by bcRealLimitGen1, 1223
called by bcSeriesExpansionGen, 1212
called by bcSeriesGen, 1219
called by bcSolveEquations, 1236
called by bcSumGen, 1194
called by bccwds2liststring, 1246
called by displayMacro, 454
called by displayValue, 459
called by editFile, 545
called by fixObjectForPrinting, 456
called by htMkName, 1248
called by htStringPad, 1248
called by makeMsgFromLine, 371
called by processChPosesForOneLine, 376
called by processSynonyms, 32
called by reportOpsFromLisplib, 818
called by reportOpsFromUnitDirectly, 822
called by whatCommands, 942

streamChop, 70
called by ncloopDQlines, 70
called by streamChop, 70
calls StreamNull, 70
calls ncloopPrefix?, 70
calls streamChop, 70
defun, 70

StreamNil, 102
used by incrgen1, 102
defvar, 102

StreamNull, 333
called by incrAppend1, 85
called by incLude1, 79
called by incZip1, 72
called by intloopProcess, 62
called by ncloopDQlines, 70
called by next1, 37
called by npNull, 333
called by parseFromString, 47
called by parseNoMacroFromString, 1374
called by streamChop, 70
calls eqcar, 333
defun, 333

called by string-concatenate
called by concat, 1047
string2id-n
called by close, 511
called by historySpad2Cmd, 582
called by importFromFrame, 563
called by listConstructorAbbreviations, 484
called by processSynonyms, 31
called by quitSpad2Cmd, 638
called by synonymnsForUserLevel, 833
called by yesanswer, 537
string2Integer
called by dbGetDocTable, 1385
string2pint-n
called by getAliasIfTracedMapParameter, 889
string2Words
called by segmentKeyedMsg, 330
STRINGCHAR, 105
defvar, 105
stringimage
called by bcInputEquations, 1231
called by bcInputExplicitMatrix, 1186
called by bcInputMatrixByFormulaGen, 1185
called by bcLinearSolveMatrixInhomo, 1238
called by bcMatrixGen, 1188
called by bcUnixTable, 1397
called by conPageFastPath, 1344
called by constructSubst, 434
called by dbConsHeading, 1395
called by dbGetDocTable, 1385
called by dbShowConditions, 1395
called by dbShowCons, 1388
called by htMkName, 1248
called by htStringPad, 1248
called by kArgumentCheck, 1371
called by kpPageInfo, 1345
called by kePageDisplay, 1354
called by koPageInputAreaUnchanged?, 1370
called by koaPageFilterByName, 1380
stringize, 1262
defun, 1262
stringList2String, 1248
defun, 1248
stringMatches?, 1090
calls basicMatch?, 1090
defun, 1090
stringp
called by porigin, 87
stringPrefix?
called by clearCmdExcept, 504
called by hasOption, 451
called by removeUndoLines, 933
called by selectOption, 479
called by undo, 922
stringSpaces
called by getFirstWord, 469
StringToDir, 1091
called by fnameMake, 1090
calls laste, 1091
defun, 1091
stripLisp, 471
called by handleNoParseCommands, 470
defun, 471
stripSpaces, 471
called by dumbTokenize, 467
called by handleNoParseCommands, 470
called by parseSystemCmd, 469
called by splitIntoOptionBlocks, 447
strpos
called by getSystemCommandLine, 833
called by processSynonyms, 31
called by rdigit?, 133
called by satisfiesRegularExpressions, 943
called by scanS, 131
called by stupidIsSpadFunction, 906
defun, 1045
strposl, 1046
called by nextline, 112
called by scanEsc, 124
called by scanSpace, 129
defun, 1046
stupidIsSpadFunction, 906
called by breaklet, 905
called by tracelet, 904
calls pname, 906
calls strpos, 906
defun, 906
subCopy
called by domArg2, 422
called by hasAtt, 424
called by hasCate, 433
called by hasCaty, 420
called by hasSigAnd, 425
called by hasSig, 423
called by replaceSharps, 958
called by unifyStructVar, 429

sublisFormal
   called by dbShowConsDoc1, 1393
called by kArgPage, 1346
called by kePage, 1352

sublisIs
   called by dbAddDocTable, 1383
called by dbConstructorDoc, ln, 1380
called by dbGetDocTable, ln, 1384
called by dbSearchOrder, 1356
called by dbShowConsDoc1, 1393
called by kcnPage, 1369
called by kcpPage, 1361
called by libConstructorSig, 1407
called by localurlib, 1015
called by mkDomTypeForm, 1348
called by reduceAlistForDomain, 1362

subMatch, 119
   called by scanPunct, 118
calls substringMatch, 119
defun, 119

subseq
   called by getFirstWord, 469
called by kdPageInfo, 1345

substFromAlist, 1280
defun, 1280

substituteSegmentedMsg
   called by getMsgInfoFromKey, 366
called by msgText, 60
called by sayKeyedMsgLocal, 330

substring
   called by ExecuteInterpSystemCommand, 31
called by alqlGetKindString, 1089
called by alqlGetOrigin, 1088
called by alqlGetParams, 1089
called by doSystemCommand, 446
called by getAliasIfTracedMapParameter, 889
called by getSystemCommandLine, 833
called by incDrop, 100
called by lineoftoks, 111
called by mkprompt, 40
called by printLabelledList, 474
called by processSynonyms, 31
called by removeUndoLines, 933
called by scanComment, 116
called by scanNegComment, 117
called by scanS, 131
called by scanW, 128
called by spleIf, 122
called by writeInputLines, 587

substringMatch, 119
called by subMatch, 119
calls qenum, 119
calls size, 119
defun, 119

subTypes, 866
   called by spadTrace, 877
called by subTypes, 866
calls exit, 866
calls lassoc, 866
calls seq, 866
calls subTypes, 866
defun, 866

suchthat, 406

syntax, 406

summary, 830
calls concat, 830
calls getenviron, 830
calls obey, 830
defun, 830

summary help page, 829

manpage, 829

superMatch?
   called by dbShowCons, 1388
called by koaPageFilterByName, 1380

syGeneralErrorHere, 192
called by npListAndRecover, 189
called by npTrapForm, 212
calls sySpecificErrorHere, 192
defun, 192

syIgnoredFromTo, 191
called by npRecoverTrap, 190
calls FromTo, 191
calls From, 191
calls To, 191
calls ncSoftError, 191
calls pfGlobalLinePosn, 191
defun, 191
synonym, 832
calls synonymSpad2Cmd, 832
defun, 832
synonym help page, 831
manpage, 831
synonymsForUserLevel, 833
called by printSynonyms, 474
calls commandsForUserLevel, 833
calls selectOptionLC, 833
calls string2id-n, 833
uses $UserLevel, 833
uses $systemCommands, 833
defun, 833
synonymSpad2Cmd, 832
called by synonym, 832
calls getSystemCommandLine, 832
calls printSynonyms, 832
calls processSynonymLine, 832
calls putalist, 832
calls terminateSystemCommand, 832
uses $CommandSynonymAlist, 832
defun, 832
syntax
assignment, 383
blocks, 386
clef, 388
collection, 389
for, 391
if, 395
iterate, 397
leave, 398
parallel, 399
repeat, 402
suchthat, 406
while, 407
sySpecificErrorAtToken, 192
called by sySpecificErrorHere, 192
calls ncssoftError, 192
calls tokPosn, 192
defun, 192
defun, 192
system help page, 837
manpage, 837
systemCommand, 448
called by handleParsedSystemCommands, 468
called by handleTokensizeSystemCommands, 447
calls commandsForUserLevel, 448
calls helpSpad2Cmd, 448
calls selectOptionLC, 448
calls selectOption, 448
uses $CategoryFrame, 448
uses $e, 448
uses $options, 448
uses $syscommands, 448
uses $systemCommands, 448
defun, 448
systemError
called by bcInputEquationsEnd, 1235
called by bcMatrixGen, 1188
called by dbShowConsDoc, 1392
called by domainDescendantsOf, 1349
called by explainLinear, 1241
called by kpPageAux, 1379
called by mkConform, 1374
called by pfDefinition2Sex, 315
called by pfLambdaTran, 316
systemErrorHere
called by interpret2, 53
called by reportOpsFromUnitDirectly, 822
tab
called by showInput, 599
tabbing, 362
called by getStFromMsg, 354
called by getMsgPrefix?, 362
uses $preLength, 362
defun, 362
take
called by ?t, 903
called by libConstructorSig, 1407
tangle help page, 839
manpage, 839
templateParts, 1284
defun, 1284
terminateSystemCommand, 452
called by commandAmbiguityError, 452
called by commandErrorMessage, 450
called by displayProperties, 462
called by npProcessSynonym, 473
called by setFortPers, 756
called by setStreamsCalculate, 802
called by synonymSpad2Cmd, 832
called by userLevelErrorMessage, 451
calls tersyscommand, 452
defun, 452
tersyscommand, 452
called by library, 1013
called by quitSpad2Cmd, 638
called by terminateSpad2Cmd, 452
calls spadThrow, 452
defun, 452
testBitVector
called by kTestPred, 1386
The restart function, 14
thefname, 89
called by inclmsgCannotRead, 90
called by inclmsgNoSuchFile, 89
calls pfname, 89
defun, 89
theid, 88
defun, 88
theorigin, 86
defun, 86
thisPosIsEqual, 374
calls poGlobalLinePosn, 374
calls poNopos?, 374
defun, 374
thisPosIsLess, 374
calls poGlobalLinePosn, 374
calls poNopos?, 374
defun, 374
throwEvalTypeMsg, 919
called by evaluateType1, 917
called by evaluateType, 916
calls spadThrow, 919
calls throwKeyedMsg, 919
local ref $noEvalTypeMsg, 919
defun, 919
throwKeyedMsg
called by addNewInterpreterFrame, 561
called by closeInterpreterFrame, 562
called by close, 510
called by fetchOutput, 600
called by frameSpad2Cmd, 566
called by getTraceOptions, 852
called by getTraceOption, 854
called by importFromFrame, 563
called by readSpad2Cmd, 642
called by restoreHistory, 597
called by saveHistory, 595
called by spadrwrite, 605
called by throwEvalTypeMsg, 919
called by trace1, 848
called by transTraceItem, 863
called by workfilesSpad2Cmd, 849
called by writeInputLines, 587
throwKeyedMsgCannotCoerceWithValue
called by evaluateType1, 918
called by interpret2, 53
throwListOfKeyedMsgs
called by getTraceOptions, 852
throws
cacheKeyedMsg, 329
fin, 548
intLoopReadConsole, 28
monitor-file, 1169
monitor-readinterp, 1176
monitor-spadfile, 1178
cmpError, 67
npMissing, 151
npTrap, 212
npTrapForm, 212
ScanOrPairVec, ScanOrInner, 615
spad-syntax-error, 909
writify, writifyInner, 607
timedEVALFUN
called by getAndEvalConstructorArgument, 958
tmp1
called by diffAlist, 925
To, 380
called by syIgnoredFromTo, 191
defun, 380
toFile?, 363
called by msgOutputter, 353
calls getMsgToWhere, 363
INDEX

uses $fn, 363
defun, 363
tokConstruct, 411
called by enPile, 340
called by npDDInfKey, 208
called by npDollar, 183
called by npFirstTok, 143
called by npId, 204
called by npInfixOperator, 160
called by npPrefixColon, 161
called by npPushId, 209
called by npSymbolVariable, 205
called by pfLeaf, 247
called by separatePiles, 341
calls ifcar, 411
calls ncPutQ, 411
calls pfNoPosition?, 411
defun, 411
token-stack-show, 971
called by iostat, 970
calls token-type, 971
uses current-token, 971
uses next-token, 971
uses prior-token, 971
uses valid-tokens, 971
defun, 971
token-type
called by token-stack-show, 971
tokPart, 413
called by intloopProcess, 62
called by npDDInfKey, 208
called by npFirstTok, 143
called by npInfixOperator, 160
called by npSymbolVariable, 205
called by pf2Sex1, 302
called by pfCopyWithPos, 236
called by pfIdSymbol, 247
called by pfLeafToken, 248
called by pfLiteralString, 249
called by pfSexr, 250
called by pfSymbolSymbol, 252
called by pfSymb, 251
called by pileCforest, 340
defun, 413
tokPosn, 413
called by firstTokPosn, 341
called by lastTokPosn, 341
called by ncloopDQlines, 70
called by npConstTok, 184
called by npDDInfKey, 208
called by npDollar, 183
called by npFirstTok, 143
called by npId, 205
called by npInfixOperator, 160
called by npMissingMate, 215
called by npMissing, 151
called by npParse, 141
called by npPrefixColon, 161
called by npPushId, 209
called by npRecoverTrap, 190
called by npSymbolVariable, 205
called by npTrapForm, 212
called by npTrap, 212
called by pfBraceBar, 257
called by pfBrace, 257
called by pfBracketBar, 258
called by pfBracket, 257
called by pfLeafPosition, 248
called by pileColumn, 337
called by sySpecificErrorAtToken, 192
calls ncAlist, 413
calls pfNoPosition, 413
calls qassq, 413
defun, 413
tokTran, 467
called by handleParsedSystemCommands, 468
called by handleTokensizeSystemCommands, 447
called by parseSystemCmd, 469
calls isIntegerString, 467
defun, 467
tokType, 413
called by npConstTok, 184
called by pilePlusComment, 336
calls ncTag, 413
defun, 413
Top, 75
used by incStream, 72
used by incString, 37
defvar, 75
Top?, 77
called by incLude1, 79
called by xIfSyntax, 94
calls quotient, 77
defvar, 77
toplevel
called by loadLibNoUpdate, 1037
topLevelInterpEval, 1372
calls processInteractive, 1372
uses $ProcessInteractiveValue, 1372
uses $noEvalTypeMsg, 1372
defun, 1372
toScreen?, 351
called by msgOutputter, 353
calls getMsgToWhere, 351
defun, 351

TPDHERE
Beware that this function occurs with lowercase also, 165
Beware that this function occurs with uppercase also, 164
Make this more international, not EBCDIC, 980
Note that there is also an npADD function, 159
Note that there is also an npAdd function, 158
Remove all boot references from top level, 472
The pform function has a leading percent sign. Fix this, 223, 225, 229
The pform function has a leading percent sign, 64, 221
The variable al is undeclared, 365
This could probably be replaced by the default assoc using eql, 1050
This function should be replaced by fillerspaces, 371
This should probably be a macro or eliminated, 471
Well this makes no sense., 197
getMsgFTTag is nonsense, 378
note that the file interp.exposed no longer exists., 1176
rewrite this using (dolist (item seqList)...), 310
rewrite using dsetq, 311

trace, 847
called by ltrace, 632
calls traceSpad2Cmd, 847
defun, 847
trace help page, 841
manpage, 841
trace1, 848
called by trace1, 848
called by traceSpad2Cmd, 847
calls /trace,0, 848
calls ?t, 848
calls addassoc, 848
calls centerAndHighlight, 848
calls delete, 848
calls devaluate, 848
calls exit, 848
calls getTraceOptions, 848
calls getTraceOption, 848
calls hasOption, 848
calls isFunctor, 848
calls lassoc, 848
calls pcounters, 848
calls poundsign, 848
calls ptimers, 848
calls qcar, 848
calls qcdr, 848
calls qlessp, 848
calls resetCounters, 848
calls resetSpacers, 848
calls resetTimers, 848
calls saveMapSig, 848
calls sayKeyedMsg, 848
calls say, 848
calls selectOptionLC, 848
calls seq, 848
calls throwKeyedMsg, 848
calls trace1, 848
calls transTraceItem, 848
calls unabbrev, 848
calls untraceDomainLocalOps, 848
calls untrace, 848
calls vcep, 848
uses $lastUntraced, 848
uses $optionAlist, 849
uses $options, 848
uses $traceNoisely, 848
INDEX

defun, 848
traceDomainConstructor, 880
calls concat, 881
calls embed, 881
calls exit, 881
calls getOption, 880
calls loadFunctor, 881
calls mkq, 881
calls seq, 880
calls spadTrace, 881
calls traceDomainLocalOps, 881
uses $ConstructorCache, 881
defun, 880
traceDomainLocalOps, 880
calledby traceDomainConstructor, 881
calls sayMSG, 880
defun, 880
tracelet, 904
calls bpiname, 904
calls compileBoot, 904
calls delete, 904
calls gensymq, 904
calls isgenvar, 904
calls lassoc, 904
calls lsersetprintflag, 904
calls stupiditySpadFunction, 904
calls union, 904
uses $QuickLet, 904
uses $letAssoc, 904
uses $traceletFunctions, 904
uses $traceletflag, 904
defun, 904
traceOptionError, 857
calls commandAmbiguityError, 857
calls stackTraceOptionError, 857
defun, 857
traceReply, 899
calledby traceSpad2Cmd, 847
calls abbreviate, 899
calls addTraceItem, 899
calls concat, 899
calls exit, 899
calls flowSegmentedMsg, 899
calls isDomainOrPackage, 899
calls isFunctor, 899
calls isSubForRedundantMapName, 899
calls isgenvar, 899
calls poundsign, 899
calls prefix2String, 899
calls qcar, 899
calls rassocSub, 899
calls sayBrightlyLength, 899
calls sayBrightly, 899
calls sayMSG, 899
calls sayMessage, 899
calls seq, 899
calls userError, 899
uses /tracenames, 899
uses $constructors, 899
uses $domains, 899
uses $lineLength, 899
uses $packages, 899
defun, 899
traceSpad2Cmd, 847
calledby trace, 847
calls augmentTraceNames, 847
calls getMapSubNames, 847
calls qcar, 847
calls qcdr, 847
calls trace1, 847
calls traceReply, 847
uses $mapSubNameAlist, 847
defun, 847
trademark, 523
defun, 523
translateTrueFalse2YesNo, 659
calledby displaySetOptionInformation, 655
calledby displaySetVariableSettings, 657
defun, 659
translateYesNoToTrueFalse, 658
calledby initializeSetVariables, 653
calledby set1, 809
calls member, 658
defun, 658
translateYesNoToTrueFalse, 1313
defun, 1313
transOnlyOption, 860
calledby getTraceOption, 854
calledby transOnlyOption, 860
calls qcar, 860
calls qcdr, 860
calls stackTraceOptionError, 860
calls transOnlyOption, 860
calls upcase, 860
defun, 860
transTraceItem, 863
calledby trace1, 848
calledby transTraceItem, 863
calledby untrace, 862
calls constructor?, 863
calls devaluate, 863
calls domainToGenvar, 863
calls get, 863
calls member, 863
calls objMode, 863
calls objVal, 863
calls throwKeyedMsg, 863
calls transTraceItem, 863
calls vecp, 863
uses $doNotAddEmptyModeIfTrue, 863
defun, 863
trapNumericErrors, 1080
defmacro, 1080
tree
calledby displaySetVariableSettings, 657
calledby initializeSetVariables, 653
calledby set1, 809
trimString
calledby removeUndoLines, 933
defmacro, 1071
typeCheckInputAreas, 1290
defun, 1290
types

calledby defaultTargetFE, 437
calledby undoChanges, 593

defun, 593

undo

calledby undoChanges, 593
calledby undoInCore, 592
calls boot-equal, 593
calls putHist, 593
calls undoChanges, 593
uses $HistList, 593
uses $InteractiveFrame, 594
defun, 593

undoCount, 929

calledby removeUndoLines, 934
calledby undo, 922
calls concat, 929
calls spaddifference, 929
calls userError, 929
uses $IOindex, 929
<table>
<thead>
<tr>
<th>Function</th>
<th>Calls</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>undoFromFile</td>
<td>assq, disableHist, exit, putHist, readHiFi, recordNewValue, recordOldValue, seq, updateHist</td>
<td>$HiFiAccess, $InteractiveFrame</td>
</tr>
<tr>
<td>undoInCore</td>
<td>assq, disableHist, putHist, readHiFi, sayKeyedMsg, spaddifference, undoChanges, updateHist</td>
<td>$HiFiAccess, $HistListLen, $HistList, $IOindex, $InteractiveFrame</td>
</tr>
<tr>
<td>undoLocalModemapHack</td>
<td>undoSingleStep, exit, seq, defun</td>
<td>$funnyBacks, $funnyQuote</td>
</tr>
<tr>
<td>undoSingleStep</td>
<td>undoSteps, exit, seq, defun</td>
<td>$funnyBacks, $funnyQuote</td>
</tr>
<tr>
<td>undoSteps</td>
<td>undo, copy, qcar</td>
<td>$funnyBacks, $funnyQuote</td>
</tr>
</tbody>
</table>
called by breaklet, 905
called by dbShowCons1, 1389
called by getMapSubNames, 869
called by tracelet, 904
unionq
called by getMapSubNames, 869
unparseInputForm, 1054
  local def $InteractiveMode, 1054
  local def $formatSigAsTex, 1054
defun, 1054
untrace, 862
called by trace1, 848
calls /untrace, 0, 862
calls copy, 862
calls lassocSub, 862
calls removeTracedMapSigs, 862
uses /tracenames, 862
uses $lastUntraced, 862
uses $mapSubNameAlist, 862
defun, 862
untraceDomainConstructor, 883
calls concat, 883
calls delete, 883
calls exit, 883
calls seq, 883
calls unembed, 883
  calls untraceDomainConstructor,keepTraced?, 883
uses /tracenames, 883
defun, 883
untraceDomainConstructor,keepTraced?, 882
  called by untraceDomainConstructor, 883
calls /untrace, 0, 882
calls boot-equal, 882
calls devaluate, 882
calls exit, 882
calls isDomainOrPackage, 882
calls kar, 882
calls qcar, 882
calls seq, 882
defun, 882
untraceDomainLocalOps, 880
called by trace1, 848
calls sayMSG, 880
defun, 880
untraceMapSubNames, 873
  called by clearCmdAll, 503
called by clearCmdParts, 505
calls /untrace, 2, 873
calls assocright, 873
calls getPreviousMapSubNames, 873
calls setdifference, 873
uses $lastUntraced, 873
uses $mapSubNameAlist, 873
defun, 873
unreadable?, 606
calls placep, 606
calls vecp, 606
defun, 606
upcase
called by close, 510
called by historySpad2Cmd, 582
called by importFromFrame, 563
called by listConstructorAbbreviations, 484
called by pathnameTypePld, 1041
called by quitSpad2Cmd, 638
called by readSpad2Cmd, 642
called by setOutputAlgebra, 763
called by setOutputFormula, 790
called by setOutputFortran, 770
called by setOutputHtml, 781
called by setOutputMathml, 776
called by setOutputOpenMath, 786
called by setOutputTex, 797
called by transOnlyOption, 860
called by yesanswer, 537
updateCategoryTable
called by loadLib, 1036
called by localnrlib, 1015
updateCurrentInterpreterFrame, 559
called by addNewInterpreterFrame, 561
called by changeToNamedInterpreterFrame, 560
called by clearCmdAll, 503
called by clearSpad2Cmd, 500
called by previousInterpreterFrame, 561
called by updateHist, 589
calls createCurrentInterpreterFrame, 559
calls updateFromCurrentInterpreterFrame, 559
uses $interpreterFrameRing, 559
defun, 559
updateDatabase, 1017
calledby loadLib, 1035
calledby localnrlib, 1015
calls clearAllSlams, 1017
calls clearClams, 1017
calls constructor?, 1017
local ref $forceDatabaseUpdate, 1017
defun, 1017
updateFromCurrentInterpreterFrame, 558
calledby addNewInterpreterFrame, 561
calledby changeToNamedInterpreterFrame, 560
calledby closeInterpreterFrame, 562
calledby initializeInterpreterFrameRing, 555
calledby nextInterpreterFrame, 560
calledby previousInterpreterFrame, 561
calledby updateCurrentInterpreterFrame, 559
calls sayMessage, 558
uses $HiFiAccess, 558
uses $HistListAct, 558
uses $HistListLen, 558
uses $HistList, 558
uses $HistRecord, 558
uses $IOindex, 558
uses $currentLine, 590
uses $mkTestInputStack, 590
defun, 589
updateInCoreHist, 590
calledby restoreHistory, 597
calledby updateHist, 589
uses $HistListAct, 590
uses $HistListLen, 590
uses $HistList, 590
defun, 590
updateInCoreHist, 590
calledby processInteractive, 48
calledby undoFromFile, 594
calledby undoInCore, 593
calls disableHist, 589
calls startTimingProcess, 589
calls stopTimingProcess, 589
calls updateCurrentInterpreterFrame, 589
calls updateInCoreHist, 589
calls writeHiFi, 589
uses $HiFiAccess, 590
uses $HistRecord, 590
defun, 589
updateHist, 589
calledby editSpad2Cmd, 544
calledby workfilesSpad2Cmd, 949
calls insert, 546
calls makeInputFilename, 546
calls member, 546
calls pathnameName, 546
calls pathnameTypeId, 546
calls pathnameType, 546
calls pathname, 546
uses $sourceFiles, 546
defun, 546
use-fast-links
calledby set1, 809
userError
calledby spadTrace, 876
calledby spadUntrace, 895
calledby traceReply, 899
calledby undoCount, 929
calledby undo, 922
userLevelErrorMessage, 450
calledby commandUserLevelError, 450
calledby optionUserLevelError, 450
calls commandAmbiguityError, 450
calls sayKeyedMsg, 451
calls terminateSystemCommand, 451
uses $UserLevel, 451
defun, 450
valid-tokens
calledby token-stack-show, 971
validateOutputDirectory, 724
calledby setFortDir, 726
calledby setFortTmpDir, 724
defun, 724
values
called by clearcmdparts, 505
called by vec2list, 1064
defun, 1064
called by vecp
vecp
called by scanorpairovec, scanorinner, 615
called by basculkupcheckdefaults, 1078
called by basiclookup, 1076
called by dewritify, dewritifyinner, 612
called by spadclosure?, 611
called by trace1, 848
called by transtraceitem, 863
called by unwritable?, 606
called by writify, writifyinner, 607
version
called by zsystemdevelopment1, 954
vmread
called by dewritify, dewritifyinner, 612
voidvalue, 1054
defun, 1054
warn
called by getdatabase, 1010
called by what, 939
called by whatspad2cmd, 940
defun, 939
called by what, 939
called by whatspad2cmd, 940
what help page, 937
manpage, 937
whatcommands, 941
called by whatspad2cmd, 940
called by blanklist, 942
called by centerandhighlight, 941
called by commandsforuserlevel, 942
called by filterlistofstrings, 942
called by说什么perlineaspossible, 942
called by saykeyedmsg, 942
called by saymessage, 942
called by say, 942
called by specialchar, 942
called by strconc, 942
uses $userlevel, 942
uses $linelength, 942
uses $systemcommands, 942
defun, 941
whatconstructors, 945
called by filterandformatconstructors, 944
calls workfilesSpad2Cmd, 949
defun, 949
workfiles help page, 949
manpage, 949
workfilesSpad2Cmd, 949
calledby workfiles, 949
calls centerAndHighlight, 949
calls delete, 949
calls makeInputFilename, 949
calls namestring, 949
calls pathname, 949
calls say, 949
calls sayKeyedMsg, 949
calls sayBrightly, 950
calls sayKeyedMsg, 949
calls say, 949
calls selectOptionLC, 949
calls sortby, 949
calls specialChar, 949
calls throwKeyedMsg, 949
calls updateSourceFiles, 949
uses $linelength, 950
uses $options, 950
uses $sourceFiles, 950
defun, 949
wrap, 1043
calledby wrap, 1043
calls lotsof, 1043
calls wrap, 1043
defun, 1043
write-browsedb, 1029
calledby make-databases, 1019
calls allConstructors, 1029
uses *print-pretty*, 1029
uses *sourcefiles*, 1029
uses $spadroot, 1029
defun, 1029
write-categorydb, 1030
calledby make-databases, 1019
calls genCategoryTable, 1030
uses *hasCategory-hash*, 1031
uses *print-pretty*, 1030
defun, 1030
write-interpdb, 1027
calledby make-databases, 1019
uses *ancestors-hash*, 1028
uses *print-pretty*, 1028
uses $spadroot, 1027
defun, 1027
write-operationdb, 1031
calledby make-databases, 1019
uses *operation-hash*, 1031
defun, 1031
write-warmdata, 1032
calledby make-databases, 1019
uses $topicHash, 1032
defun, 1032
writeablep
calledby myWritable?, 1093
writeHiFi, 602
calledby updateHist, 589
calls histFileName, 602
calls object2Identifier, 602
calls rdefostream, 602
calls rshut, 602
calls spadrwrite, 602
uses $HistRecord, 602
uses $IOindex, 602
uses $currentLine, 602
uses $internalHistoryTable, 602
uses $useInternalHistoryTable, 602
defun, 602
writeHistModesAndValues, 603
calledby processInteractive, 48
calls get, 603
calls putHist, 603
uses $InteractiveFrame, 603
defun, 603
writeInputLines, 587
calledby historySpad2Cmd, 582
calledby saveHistory, 595
calledby undoSteps, 930
calls concat, 587
calls defostream, 587
calls histFileErase, 587
calls histInputFileName, 587
calls namestring, 587
calls readHiFi, 587
calls sayKeyedMsg, 587
calls shut, 587
calls size, 587
calls spadDifference, 587
calls substring, 587
calls throwKeyedMsg, 587
uses $HiFiAccess, 587
uses $IOindex, 587
uses underbar, 587
defun, 587
writeLib1
calledby saveDependentsHashTable, 1022
calledby saveUsersHashTable, 1022
writify, 610
calledby safeWritify, 607
calls ScanOrPairVec, 610
calls function, 610
calls writify,writifyInner, 610
uses $seen, 610
uses $writifyComplained, 610
defun, 610
writify,writifyInner, 607
calledby writify,writifyInner, 607
calledby writify, 610
calls boot-equal, 607
calls constructor?, 607
calls devaluate, 607
calls exit, 607
calls hashtable-class, 607
calls hget, 607
calls hkeys, 607
calls hput, 607
calls qcar, 607
calls qcdr, 607
calls qrplaca, 607
calls qrplacd, 607
calls qsetvelt, 607
calls qvelt, 607
calls qvmaxindex, 607
calls seq, 607
calls spadClosure?, 607
calls vcep, 607
calls writify,writifyInner, 607
uses $NonNullStream, 607
uses $NullStream, 607
uses $seen, 607
defun, 607
throws, 607
writifyComplain, 606
calls sayKeyedMsg, 606
uses $writifyComplained, 606
defun, 606
xlCannotRead, 90
calledby include1, 80
calls inclmsgCannotRead, 90
calls xMsg, 90
defun, 90
xlCmdBug, 96
calledby include1, 80
calls inclmsgCmdBug, 96
calls xMsg, 96
defun, 96
xlConActive, 91
calledby include1, 80
calls inclmsgConActive, 91
calls xMsg, 91
defun, 91
xlConsole, 92
calledby include1, 80
calls inclmsgConsole, 92
calls xMsg, 92
defun, 92
xlConStill, 92
calledby include1, 80
calls inclmsgConStill, 92
calls xMsg, 92
defun, 92
xlFileCycle, 90
calledby include1, 80
calls inclmsgFileCycle, 90
calls xMsg, 90
defun, 90
xlIfBug, 95
calledby include1, 80
calls inclmsgIfBug, 95
calls xMsg, 95
defun, 95
xlIfSyntax, 94
calledby include1, 80
calls Else?, 94
calls Top?, 94
calls inclmsgIfSyntax, 95
calls xMsg, 94
defun, 94
INDEX

xlMsg, 84
called by xlCannotRead, 90
called by xlCmdBug, 96
called by xlConActive, 91
called by xlConStill, 92
called by xlConsole, 92
called by xlFileCycle, 90
called by xlHBug, 95
called by xII Syntax, 94
called by xIINoSuchFile, 89
called by xIPrematureEOF, 84
called by xIPrematureFin, 93
called by xISay, 88
called by xISkippingFin, 93
calls incLine, 84
defun, 84

xINoSuchFile, 89
called by incLude1, 80
calls inclmsgNoSuchFile, 89
calls xI Msg, 89
defun, 89

xlOK, 84
called by incLude1, 80
calls lxOK1, 84
defun, 84

xlOK1, 85
called by incLude1, 80
calls incLine1, 85
defun, 85

xIPrematureEOF, 84
called by incLude1, 79
calls inclmsgPrematureEOF, 84
calls xIMsg, 84
defun, 84

xIPrematureFin, 93
called by incLude1, 80
calls inclmsgPrematureFin, 93
calls xIMsg, 93
defun, 93

xISay, 88
called by incLude1, 80
calls inclmsgSay, 88
calls xIMsg, 88
defun, 88

xISkip, 87
called by incLude1, 80
calls CONCAT, 87
calls incLine, 87
defun, 87

xISkippingFin, 93
called by incLude1, 80
calls inclmsgFinSkipped, 93
calls xIMsg, 93
defun, 93

xtokenreader, 968
used by init-boot/spad-reader, 968
defvar, 968

yesanswer, 537
called by displayOperations, 537
calls queryUserKeyedMsg, 537
calls string2id-n, 537
calls upcase, 537
defun, 537

ySearch
called by conPage, 1343

zeroOneConvert
called by kePageOpAlist, 1353

zeroOneTran, 66
called by intInterpretPform, 66
calls nsubst, 66
defun, 66

zsystemdevelopment, 953
calls zsystemDevelopmentSpad2Cmd, 953
defun, 953

zsystemdevelopment help page, 953
manpage, 953

zsystemdevelopment1, 954
called by zsystemDevelopmentSpad2Cmd,
954
calls /D, 1,
954
calls /comp, 954
calls bright, 954
calls defiostream, 954
calls kaddr, 954
calls kadr, 954
calls kar, 954
calls next, 954
calls sayBrightly, 954
calls sayMessage, 954
calls selectOptionLC, 954
calls shut, 954
calls version, 954
uses /version, 954
uses /wsname, 954
uses $InteractiveMode, 954
uses $options, 954
catches, 954
defun, 954

zsystemDevelopmentSpad2Cmd, 953
calledby zsystemdevelopment, 953
calls zsystemdevelopment1, 953
uses $InteractiveMode, 953
defun, 953