axiom™

The 30 Year Horizon

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

Volume 5: Axiom Interpreter

July 2, 2018

3656d946018ca3771de000d37881df68a0d3ef29

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
Christian Aistleitner
Jerry Archibald
Henry Baker
Yurij Baransky
Gerald Baumgartner
Nelson H. F. Beebe
Fred Blair
Raoul Bourquin
Wolfgang Brehm
Manuel Bronstein
Florian Bundschuh
Ralph Byers
Robert Cavines
Tzu-Yi Chen
David V. Chudnovsky
Roland Coeurjoly
Josh Cohen
George Corliss
Meino Cramer
David Cyganski
Timothy Daly Jr.
James Demmel
Inderjit Dhillon
Gabriel Dos Reis
Nicolas James Doye
Iain Duff
Brian Dupee
Hans-Dieter Ehrich
Mark Fahey
Stuart Feldman
Albrecht Fortenbacher
Timothy Freeman
Rudiger Gebauer
Patricia Gianni
Holger Gollan
Stephen Gortler
Klaus Ebbe Grue
Oswald Gschnitzer
Gaetan Hache
Sven Hammarling
Richard Harke
Martin Hassner
Waldek Hebisch
Henderson
Hoon Hong
Pietro Iglio
Bo Kagstrom
Kai Kaminski
Tony Kennedy

Cyril Alberga
Richard Anderson
S.J. Atkins
Martin Baker
David R. Barton
Gilbert Baumslag
Jay Belanger
Vladimir Bondarenko
Alexandre Bouyer
Peter A. Broadbery
Christopher Brown
Luanne Burns
Quentin Carpent
Bruce Char
Bobby Cheng
Gregory V. Chudnovsky
James Cloos
Christophe Conil
Robert Corless
Karl Crary
Nathaniel Daly
James H. Davenport
Didier Deshommes
Jack Dongarra
Claire DiCrescendo
Zlatko Drmac
Lee Duhem
Dominique Duval
Heow Eide-Goodman
Richard Fateman
John Fletcher
George Frances
Korrinn Fu
Van de Geijn
Gustavo Goertkin
Teresa Gomez-Diaz
Johannes Grabmeier
James Griesmer
Ming Gu
Steve Hague
Mike Hansen
Bill Hart
Arthur S. Hathaway
Karl Hegbloom
Antoine Hersen
Roger House
Alejandro Jakubi
William Kahan
Grant Keady
David Kincaid

Roy Adler
George Andrews
Jeremy Avigad
Stephen Balzac
Thomas Baruchel
Michael Becker
David Bindel
Mark Botch
Karen Braman
Martin Brock
Stephen Buchwald
William Burge
Pierre Casteran
Ondrej Certik
Cheekai Chin
Mark Clements
Jia Zhao Cong
Don Coppersmith
Gary Cornell
Jeremy Du Croz
Timothy Daly Sr.
David Day
Michael Dewar
Jean Della Dora
Sam Dooley
Lionel Ducos
Martin Dunstan
Robert Edwards
Lars Erickson
Bertfried Fauser
Brian Ford
Constantine Frangos
Marc Gaetano
Kathy Gerber
Samantha Goldrich
Laureano Gonzalez-Vega
Matt Grayson
Vladimir Grinberg
Jocelyn Guidry
Satoshi Hamaguchi
Richard Hanson
Vilya Harvey
Dan Hatton
Ralf Hemmecke
Nicholas J. Higham
Gernot Hueber
Richard Jenks
Kyriakos Kalorkoti
Wilfrid Kendall
Keshav Kini

Ted Kosan
Paul Kosinski
Igor Kozachenko
Fred Krogh
Klaus Kusche
Bernhard Kutzler
Tim Lahey
Larry Lambe
Kaj Laurson
Charles Lawson
George L. Legendre
Franz Lehner
Frederic Lehobey
Michel Levaud
Howard Levy
J. Lewis
Ren-Cang Li
Rudiger Loos
Craig Lucas
Michael Lucks
Richard Luczak
Camm Maguire
Francois Maltey
William Martin
Osni Marques
Alasdair McAndrew
Bob McElrath
Michael McGettrick
Edi Meier
Ian Meikle
David Mentre
Jonathan Millen
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
Frank Pfenning
Jose Alfredo Portes
E. Quintana-Orti
Gregorio Quintana-Orti
Beresford Parlett
A. Petitet
Andre Platzer
Peter Poromaas
Claude Quitte
Arthur C. Ralfs
Norman Ramsey
Anatoly Raportirenko
Guilherme Reis
Huan Ren
Albert D. Rich
Michael Richardson
Jason Riedy
Renaud Rioboo
Jean Rivlin
Nicolas Robidoux
Simon Robinson
Raymond Rogers
Michael Rothstein
Martin Rubey
Jeff Rutter
Philip Santas
David Saunders
Alfred Scheerhorn
William Schelter
Gerhard Schneider
Martin Schoenert
Marshall Schor
Frithjof Schulze
Fritz Schwarz
Steven Segletes
V. Sima
Nick Simicich
William Sit
Elena Smirnova
Jacob Nyffeler Smith
Matthieu Sozeau
Ken Stanley
Jonathan Steinbach
Fabio Stumbo
Christine Sundaresan
Klaus Sutner
Robert Sutor
Moss E. Sweedler
Eugene Surowitz
Yong Kiam Tan
Max Tegmark
T. Doug Telford
James Thatcher
Laurent Thery
Balbir Thomas
Mike Thomas
Dylan Thurston
Francoise Tisseur
Steve Toleque
Raymond Toy
Barry Trager
Themos T. Tsikas
Gregory Vanuxem
Kresimir Veselic
Christof Voemel
Bernhard Wall
Stephen Watt
Andreas Weber
Jaap Weel
Juergen Weiss
M. Weller
Mark Wegman
James Wen
Thorsten Werther
Michael Wester
R. Clint Whaley
James T. Wheeler
John M. Wiley
Berhard Will
Clifton J. Williamson
Stephen Wilson
Shmuel Winograd
Robert Wisbauer
Sandra Wityak
Waldemar Wiwianka
Knut Wolf
Yanyang Xiao
Liu Xiaojun
Clifford Yapp
David Yun
Qian Yun
Vadim Zhytnikov
Richard Zippel
Evelyn Zoernack
Bruno Zuercher
Dan Zwillinger

Contents

New Foreword

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

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

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

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

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

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

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

Chapter 1

Type Inference and Coercion

1.1 Introduction

The Axiom system is centered on a strongly typed abstract datatype programming language with multiple inheritance. The compiler for this language produces a library of packages of polynomorpic functions and parameterized abstract datatypes. The system interpreter provides an interface to the compiler and library and supports a subset of the Axiom language. The design of the interpreter includes extensive datatype inferences and coercion facilities to ease the burden of working with a strongly typed language and a library containing over 1100 Categories, Domains, and Packages.

The Axiom interpreter is unusual because

- Axiom has an infinite number of datatypes and packages
- each generic operation generally has an infinite number of interpretations
- datatypes and packages are instantiated dynamically in response to user input, and
- operation interpretation is done by pattern matching on modemaps.

The design philosophy is that the interpreter language equals the compiler language minus the constraints.

This means that declarations are largely optional and that automatic datatype conversions may take place when they are deemed appropriate.

In Axiom we use the term *conversion* to mean the transformation an object of one datatype into an object of another datatype. A special kind of conversion is *coercion*, where we further require that transformation is reasonable (usually in an algebraic or non-information-losing sense). Examples of coercions involve changing integers into rational numbers, rational numbers into (constant) polynomials with rational number coefficients, lists of lists into two dimensional arrays, etc.

Conversion is done through two operations, **coerce** and **convert** which are otherwise ordinary operations exported by the abstract datatypes of the system. The interpreter is allowed to perform conversions named **coerce** whenever it determines they are necessary. Most coercions are reversible and their application does not lose information. An example of a conversion that is not a coercion is the transformation of a rational number into a floating point number. Having an automatic conversion from an exact to an inexact object is not

acceptable.

All atomic expressions (as produced by the parser) have default datatypes. These are **Boolean**, **Float**, **Integer**, **String**, **Symbol**, and **List(None)** (for an empty list). If these are assigned in declared variables or used as arguments to functions, coercions are often performed to create objects of other datatypes.

Coercion is combined with the *type resolve facility* to determine the result datatype of an expression presented to the interpreter. Datatypes **T1** and **T2** *resolve* to a datatype **T3** if all objects belonging to **T1** and **T2** can be coerced to **T3**. In Axiom, the resolve facility is driven by a rule system, by information about coercions, and by the structure of the datatypes involved.

1.2 Overview of the Abstract Datatype System

It is not within the scope of this chapter to fully discuss the Axiom abstract datatype system. This section is an overview of those terms and concepts needed to describe the implemention of the interpreter. Other sources are available that contain more detailed discussions. [Jenk81][Lisk79][Fort85][Swee86] [Jenk86][Watt87]

The Axiom library consists of compiled code that creates "categories", "domains", and "packages". A *domain* is a datatype, a *package* is a collection of functions, and a *category* is a collection of operation signatures and attributes, as shown below. These are created by special functions called *constructors*, which are generally parameterized.

Domains and packages implement the functions in categories. Categories serve to separate the contract from the implementation. However, default implementations can be given in categories by assuming the existence of other implemented operations. For example, a default implementation of binary "-" can be given by using unary "-" and binary "+"".

1.2.1 Categories

A domain is an instance of an abstract datatype specified by one or more categories. Categories serve to group together domains with common properties (e.g. operations). A basic category in Axiom is **Set** which has the definition:

```
Set(): Category == with
  "=" : ($,$) -> Boolean
  coerce : $  -> Expression
```

The syntax means the following. **Set** is a category constructor having no parameters. Lines 2 and 3 of the definition present *signatures* for the two operations that are exported by **Set**: an operation "=" that takes 2 elements of the set and returns a **Boolean**, and an operation **coerce** that takes an element of the set and returns a printable representation of it. The symbol "$" refers to the set itself. All domains which belong to **Set** contain "=" and **coerce** among their operations (though it is possible that they are not implemented).

Categories may be extended. A semigroup is a set that has an associative multiplication operation. Given such a mulitplication, it is easy to define an exponentiation by positive integers ($x**1 = x$, $x**2 = x*x$, $x**3 = x*x**2$,...). The definition of the **SemiGroup** constructor in Axiom is

```
SemiGroup(): Category == Set with
```

```
"*" : ($,$) -> $
"**" : ($, PositiveInteger) => $
associate("*")
```

The last line of this definition presents an *attribute* declaring that multiplication is associative. Attributes declare properties of the operations or the datatype.

As an example, **Integer** belongs to *SemiGroup* which means that it should have everything from **Set** plus the listed mutiplication and exponentiation operators. Multiplication is asserted to be associative and so this may be assumed for **Integer**. If a category **C2** extends a category **C1**, any domain that belongs to **C2** also belongs to **C1**.

Categories may be parameterized.

```
ListCategory(S: Set): Category == Set with
  first : $ -> $
  rest :  $ -> $
  null :  $ -> Boolean
  . . .
```

presumably describes the operations and attributes that any abstract datatype that purports to act like a linked list should have. The parameter **S** can be any domain that belongs to **Set**. If we defined a special version of this

```
SemiGroupListCategory(S: SemiGroup): Category == Set
  with
    first : $ -> $
    rest :  $ -> $
    null :  $ -> Boolean
    . . .
```

then **S** must have been explicitly declared to belong to **SemiGroup**. Category membership is by name: a domain having the operations and attributes of **SemiGroup** does not belong to this category unless it has been explicitly declared to belong to **SemiGroup** or one of its extensions.

Categories allow multiple inheritance.

```
FiniteList(S: Set): Category ==
  Join(Finite, ListCategory(S))
```

defines a category constructor that contains all operation signatures and attributes from both the categories **Finite** and **ListCategory(S)**.

1.2.2 Domains

Domains are created by domain constructors and are objects. They provide a representation for an instance of an abstract datatype, implement the operations of the categories to which they belong, and presumably satisfy the attributes of the categories. The representation of a domain does not depend at all on the categories to which the domain belongs.

In the implementation of **Integer**, the following category definition is given and specifies the behavior of the domain:

```
Integer(): Join(UniqueFactorizationDomain,
  EuclideanDomain, OrderedRing, DifferentialRing)
    with
      oddp : $ -> Boolean
```

```
random : () -> $
numberOfDigits : ($,$) -> $
abs : $ -> $
canonicalUnitNormal
```

The category of **Integer** is not one named category but, instead, is composed of several pieces. The full category is that obtained by joining 4 named categories, 4 additional operations, and 1 additional attribute. Note, though, that **Integer** belongs to each of the 4 named categories, *plus* their ancestors. In particular, **Integer** is a **SemiGroup** because it is a **DifferentialRing**, which is a **Ring**, which is a **SemiGroup**. In fact, the **Join** creates several paths back to the category **SemiGroup**. **Integer** may be used anywhere a **SemiGroup** is required.

Domains may provide functions for objects other than those which they create. For example, **RationalNumber** implements a "/" for two **Integer** arguments.

Two domain constructors that are particularly useful in representing other domains are **Record** and **Union**. A record is similar to structures of the same name in many other languages: it has one or more fields accessed via selector names. The objects in the fields can be of any datatype. The construtor **Union** creates *discriminated* unions: an object of **Union(Integer,String)** belongs either to the **Integer** branch or the **String** branch of the union. The **case** function is used to test for membership in branches.

The constructor **Mapping** creates the datatypes of functions. Declaring **oddp : Mapping(Boolean,Integer)** states that **oddp** is a function taking one integer argument and returning a boolean. A convenient alternative notation for this is **oddp: Integer** $\rightarrow$ **Boolean**. Functions are first-class objects and can be passed to other functions.

The Axiom language design completely separates the concrete implementation inheritance from the abstract specification inheritance. Domains can inherit implementations from their representations, but the external view of them is only determined by the categories to which they belong. The idea of *subtype* in languages such Trellis/Owl[1] [Scha86] is replaced in Axiom by the two notions of category and domain.

In addition to providing an organizational hierarchy, categories allow Axiom to have extra information about domains. Thus, for example, the compiler knows where operations will be located in domains and can compile efficient function calls. This information is part of a structure associated with each operation called a *modemap*.

1.2.3 Packages

A package is a special kind of domain in that it provides no new objects to the system (other than the package itself). It consists of a category listing operations and attributes and implementations of the operations. Because they may be parameterized, packages can be used to implement algorithms at their most natural level of abstraction. For example, a repeating-squaring algorithm may be contained in a package that is parameterized by a domain that is a **SemiGroup**.

Packages provide an excellent vehicle to implement both domain-specific and category-general algorithms. They can be used to implement algorithms that are not included in other domains for reasons of convenience or necessity. For example, the repeated squaring algorithm and the integration algorithms are implemented in packages. They also require

[1] Trellis is a trademark of Digital Equipment Corporation

extra work by the interpreter because argument datatypes give no simple clue to the possible location of a package function. For example, in the expression $2 + 9$ the operation "+" is found in the domain **Integer**, which is the common datatype of the two arguments. However, the function **solve** in `solve(x**3-1,x)` is not present in either the domain **Polynomial(Integer)** of the first argument or the domain **Symbol** of the second. This, along with the last example of the previous section, demonstrates the need for a systematic search mechanism of the Axiom library.

1.2.4 Modemaps

A *modemap* is a syntactic specification of an operation. It gives the name of the operation, source and target datatypes for its parameters, and the name of the datatype or package exporting the operation. For example, the modemap

```
oddp : Integer -> Boolean from Integer
```

specifies an operation from type *Integer* which takes an integer argument and returns a boolean value. The "from"-clause indicates the *domain of implementation* of the operation (in this case, **Integer**).

When the datatype or package is parameterized, the conditions on their parameters appear in an "if"-clause for the modemap. For example, every domain created by domain constructor **Matrix** exports a trace operation:

```
trace: Matrix(R) -> $ if R has Ring from Matrix(R)
```

The domain constructor **Matrix** takes one parameter **R** which is required to belong to the category **Ring**.

Operations may also be conditionally exported by a domain constructor. For example, domains created by **Matrix** export a **determinant** operation only if the homogeneous multiplication "*" in the underlying domain is commutative, that is, **R** has **commutative("*")**. The modemap for **determinant** is

```
determinant: Matrix(R) -> $ if R has Ring
  and R has commutative("*") from Matrix(R)
```

Similarly, datatypes creaated by **Matrix** export an **inverse** operation only if their argument domain belongs to **Field**. Since the requirement "**R** has **Field**" implies "**R** has **Ring**" the modemap for **inverse** is

```
inverse: Matrix(R) -> Union(R,"failed")
   if R has Field from Matrix(R)
```

The collection of modemaps from all abstract datatypes and packages constitutes the *global modemap database* for Axiom. To standardize the presentation of modemaps, arguments and other parameters are prefaced by "pattern variables", here called ***1**, ***2**, etc. The resulting form of the modemap for **Matrix inverse** in the modemap database is, for example,

```
inverse: *1 -> Union(*1,"failed")
   if *1 is Matrix(*2) and *2 has Field from *1
```

In addition to this general modemap database, each domain or package has a *local modemap database* for all its exported operations. Note that the pattern variables and "if"-clauses of the above global modemaps depend on the parameters to a domain or package constructor. For a domain or package itself, all parameters to the constructor are known. Thus local modemaps have no patterns. Predicates reduce to *true* or *false*. All modemaps in a local

modemap database for a domain or package **D** therefore have the general form

```
f: S -> T if true from D
```

Those having a predicate of `false` simply do not exist for a given domain or package. For example, **Matrix(RationalNumber)** will have an **inverse** operation whereas **Matrix(Integer)** will not, since **RationalNumber** is a field but **Integer** is not.

1.2.5 Interpretation

The role of the interpreter is to evaluate input expressions entered by the user. In addition to computations, these expressions may be declarations, function definitions, or system commands. A value in Axiom is described by a pair $< a$,A> where a is an object and A is its type.

Evaluation of an operator-operand expression $f(x, y, \ldots)$, with n $(n > 1)$ arguments $x, y, \ldots$, is done in a bottom-up manner. The interpreter will evaluate the arguments to produce a corresponding n-tuple of argument values $< a$,A>, $< b$,B>, ... At this point an attempt will be made to select an applicable modemap for f. If this modemap has a return type of **T**, f is applied to the (possibly coerced) arguments to yield a result t and subsequent value pair $< t$,T>.

1.2.6 Modemap Selection

Given a function call $f(x, y, \ldots)$ and evaluated arguments $< a$,A>, $< b$,B>, ... as in the previous section, the interpreter tries to find an appropriate f to apply. The first attempt involves constructing a list of the domains A, B, ... of the evaluated arguments, plus the target type of the function call, if it exists,[2] plus those types contained in any of the argument types if they are constructed from **Record**, **Union**, or **Mapping**. Duplicates are removed from the list and domains and packages in the list are searched for an applicable f. If a modemap is found of the form

```
f: (A,B,...) -> T from C
```

with arity n, for some **T** and **C**, the function is gotten from **C** and applied to $a, b, \ldots$ to yield a result t and subsequent value pair $< t$,T>.

This kind of search for an applicable f generalizes the idea of a *controlling object* in a function call [Lisk79][Scha86]. In Axiom, though, the arguments in a function call need not give any hint to the actual location of the function to be applied. If there is no applicable modemap to be found among the argument domains of a function, a two-stage search for a suitable modemap is made in the global modemap database. A set M of candidate modemaps is constructed, each of the form:

```
f:(D,E,...) -> T if p from C
```

for some result type **T** and domain of implementation **C**, each of arity n and each generally containing pattern variables and predicates p. The set M is partitioned into two subsets: those modemaps coming from domains or packages whose names contain those of any of the arguments of the function, and those that do not. The modemaps in the first subset are examined for applicability and, if one is found, it is returned. Otherwise, those in the second subset are checked for applicability.

[2] As an example of a target type, consider the expression `m:Matrix(Integer):=f(x,y,z)`. The target type of the function call is `Matrix(Integer)`.

At first this partitioning may seem odd, but it reflects a naming convention for domains and packages in the Axiom library. As an example, the constructor **ListPackage1** takes one **Set** argument and implements several functions that could be but are not now implemented in the **List** domain. The constructor **ListPackage2** takes two domain arguments, each belonging to the category **Set**. The functions in this package implement operations that requires lists with two different element types. For example, the function

```
map: (S -> T, List(S)) -> List(T)
```

which maps a function from **S** to **T** across the elements of **List(S)**, producing an object of **List(T)**, is implemented in **ListPackage2(S,T)**. Looking at the modemaps in the first partition of M reflects an extension of the search method that looks for applicable functions in the domains of the function arguments. In the future we anticipate using attributes in such associated packages to determine the first subset of the general candidate modemaps. That is, **ListPackage2** will contain the attribute **associated("List")** in its category.

For our purposes here, it suffices to regard the predicate of a modemap as a conjunction of simple predicates of two kinds:

```
X is Y
X has Y
```

Predicates of the first kinds state that pattern X matches only the explicit domain `Y`. If a modemap only has predicates of the first kind, patterns are replaced by the domain or package names to produce a new modemap free of patterns.

When the modemap has predicates of the second kind, substitutions are sought for the pattern variables such that the predicate is satisfied. Here `Y` is a category or an attribute. Pattern variables are given initial substitutions based on the value pairs of the arguments. Any pattern variable X for which there is a predicate "X is `Y`" has `Y` assigned as a *permanent* substitution. Other substitutions are labeled *tentative*. To satisfy the predicate, any or all the operations of *coercing*, *resolving*, and *forcing* (see below) may be necessary. As above, if pattern variable X has permanent substitution `Y`, any argument value of f of type **Z** must be coerced to type `Y`.

The *resolve* of two types **T1** and **T2** will always succeed. It produces a third type **T3** to which all objects of type **T1** and **T2** can be coerced. The type **Any** is used for **T3** if a less general type cannot be found. Resolving is necessary to use a modemap with homogeneous arguments. For example, the modemap

```
"+":(*1,*1) -> *1 if *1 has AbelianMonoid from *1
```

is used for addition in most algebraic domains in Axiom. Given an expression $1 + (2/3)$, a bottom-up analysis will first produce <1,Integer> and <2/3,RationalNumber> as the values of the two operands of "+". The resolve of **Integer** and **RationalNumber** is defined to be **RationalNumber**. This causes the integer object 1 to be coerced to the rational number object 1/1. **RationalNumber** now matches *1 in the above modemap, the predicate evaluates to true, and the corresponding function is gotten from **RationalNumber** and applied to produce the result 5/3 and pair <5/3,RationalNumber>.

A *force* is an operation performed on one or more types to satisfy a predicate. For example, the predicate "*1 has **Field**" where *1 has a tentative substitution **Integer** results in the forcing of **Integer** to **RationalNumber** by the application of **QuotientField**, i.e. **QuotientField(Integer)** is equivalent to **RationalNumber**. As another example, x has default type **Symbol**. When appearing in a sum (e.g. $x + 1$), the modemap of x requires its argument to be from a domain which belongs to the category **AbelianMonoid**. As a

result, **Symbol** is forced to **Polynomial(Integer)**.

A list of all applicable modemaps is produced together with a list of required coercions on the source parameters. Duplicates and modemaps subsumed by others are discarded. Each modemap is assigned a cost based on the required coercions and the target type of the operation. By definition of the cost function, any modemap which directly matches the argument types (e.g. **A=D**, B=E, etc.) will have the cheapest cost.

1.2.7 Ambiguity

Two remaining modemaps are said to be *ambiguous* unless there are coercions to and from the respective substitution datatypes for the pattern variables *1,*2,... . If there are no ambiguities, the cheapest modemap is selected and applied.

As a practical matter, users can always use a "package call" to avoid ambiguities, e.g. `x *$D y` and `foo$D(x,y)` direct the interpreter to apply the functions "*" and *foo* from the domain or package **D**. Package calling is the only way to identify uniquely functions of no arguments if they are ambiguous.

1.2.8 Modes

If the explicit conversion `p::P` or `p::P(?)` is given, `p` is converted to a polynomial. The datatype of coefficients may be any domain which satisfies the categorical requirements of the argument to the domain constructor `Polynomial`. The form `P(?)` is a *mode* specification rather than a *type* specification: the interpreter is free to choose what replaces the `?`.

A mode is a partial type specification in that zero or more arguments in a domain constructor call are replaced by `?`. The process of *merging* takes a type **T1** and a mode **M** and determines type substitutions for the `?`'s in **M** to create a new type **T2** to which **T1** is coercable. For example, the merger of **RationalNumber** and **Polynomial(?)** is **PolynomialRationalNumber**. If the mode **M** has no `?`'s, it is a type and the merger will only succeed if **T1** is coercable to **M**. Thus merging a type and a mode may fail, unlike resolving two types. If **M** is simply `?`, then the result of the merger is **T1**.

Axiom now only supports modes with at most one `?`, and that must be in the innermost constructor call position (e.g. **List(Polynomial(?))**. In our experience, this restriction has not seemed burdensome, though a future area of research might be the extension of the merging process to more general modes.

1.3 The Coerce Facility

Our goal in the design of the coerce facility is to have as much as possible controlled by modemap selection of compiled Axiom functions. This allows the domain and package writer maximum control over the behaviour of the interpreter and removes the requirement of having a system developer tune the interpreter for dealing with new datatypes. The coerce facility has several components.

1.3.1 Coerce by Function

The interpreter does modemap selection for an operation named **coerce** that has the appropriate argument and target types. This is the easist way for a programmer to control the coerce facility. For example, the category **Ring** provides a modemap

```
coerce: Integer -> R from R
```

where **R** is the ring being defined. Thus any ring has a coercion from **Integer** and, in fact, this operation has a default categorical definition.

Coercions can sometimes be defined in domains but are often defined in packages. A coercion of the form

```
coerce: Polynomial(QuotientField(R)) ->
  QuotientField(Polynomial(R))
    if R has IntegralDomain
```

would typically be defined in a package parameterized by the domain **R**. On the other hand, it is not feasible to have so many explicit coercions being written and, in fact, there are general methods that will perform these kinds of coercions.

It is not now possible to define a coercion of the form

```
coerce: List(S) -> List(T)
```

in the domain constructor **List**. This is because **List** is parameterized by only one set (**S** or **T**) and the modemap cannot, therefore, be part of the category of **List**. Such an explicit coercion can be provided in a package, but is, in fact, handled by the general mechanism described in the next section.

1.3.2 Coerce by Mapping

When the interpreter encounters a coercion of the form

```
D(T1) -> D(T2)
```

where **D** is a domain constructor with a parameter (for reasons of exposition, we here omit the case of **D** having multiple parameters), it looks for a function

```
map: (T1 -> T2, D(T1)) -> D(T2)
```

that takes a function from **T1** to **T2** and an object of **D(T1)** and produces an object of **D(T2)**. It then creates a function stub that coerces objects of **T1** to those of **T2** and passes it to **map**, along with the original argument.

Since the function **map** is part of the library of Axiom compiled code, it allows the package writer to automatically provide an interpreter mechanism of "lifting" coercions from **T1** to **T1** to **D(T1)** and **D(T2)**.

1.3.3 Coerce by Internal System Code

Some special cases of coercions are handled by internal system code rather than compiled Axiom code. These typically involve polynomials where the variable ordering is changed or distributed across a tower of parameterized domains. It will eventually be moved into Axiom code as the pattern matching facilities are improved.

Another case that is now handled internally involves rearrangement of a tower of paramter-

ized domains. The domain constructor **Gaussian** creates domains with objects similar to the complex numbers in that they contain real and imaginary parts. The coercion

```
Gaussian(Polynomial(RationalNumber)) ->
   QuotientField(Polynomial(Gaussian(Integer)))
```

is performed in the following steps. The type **RationalNumber** is changed into the equivalent type **QuotientField(Integer)** and the **QuotientField** is bubbled to the top of the original type to get **QuotientField(Gaussian(Polynomial(Integer)))**. This new type and the old target now have the same top level constructor (**QuotientField**) and the coerce facility is called recursively on the underlying domains **Gaussian(Polynomial(Integer))** and **Polynomial(Gaussian(Integer))**.

1.3.4 Coercion of Algebraic Constants

Several categories specify the existence of constants. For example, **AbelianMonoid** specifies the operation "+":$ → $" and the constant 0 : $. If **T1** and **T2** each belong to a common category with specified constants and the object of **T1** to be coerced is one of the constants, the corresponding constant in **T2** may be extacted and returned.

1.3.5 Retraction

A retraction is a coercion of an object of a domain to a more specific (degenerate) domain. Unlike other forms of coercion where the target type is known, retraction involves examining an operand pair < *t*,T> to see if there exists a degenerate form of **T** to which *t* can be coerced. For example, <7/1,RationalNumber> retracts to <7,Integer> and <1,Polynomial(Integer)> retracts to <1,Integer>. Retraction can occur multiple times, so arbitrarily long sructures collapse to their simplest forms, e.g. <1,Polynomial(RationalNumber)> to <1,Integer>.

If no applicable modemap is found in modemap selection, retraction is done on the arguments and then selection is attempted again. Retraction is also attempted when a coercion is requested from a domain to its underlying domain. For example, if `p` is an element of **Polynomial(Integer)**, the statement `p::I` will cause the interpreter to try to retract `p` to an element of **Integer**.

For the most part, retraction can be accomplished by functions in the Axiom library. The category **RetractWithUnderDomain(R)** specifies two operations

```
retractable? : $ -> Boolean
retract :      $ -> R
```

In our example of polynomials above, the function **retractable?** would be called to see if the object was a constant polynomial. If the result was *true*, **retract** can be called to extract the constant.

1.3.6 Coercion Query

There are situations where one wishes to know ahead of time whether it is possible to coerce an object of type **T1** to one of type **T2**. The Axiom interpreter provides this information, for example, to the type resolve and modemap selection facilities. The facility is used when one needs to know absolutely when a coercion will be successful. An answer of "no", however, does not guarantee that a coercion could not be performed for specific data. For example

the system will respond "no" when asked whether an object of **Polynomial(Integer)** can be coerced to an object of **Integer**. As we saw above, though, retractions can be performed for constant polynomials.

1.4 The Resolve Facility

The resolve facility is used to determine a type **T3** to which two types **T1** and **T2** can be coerced. It is used when an operation has homogeneous arguments (such as "+") or when a statement has several exit points and they must all return the same type (`then` and `else` clauses of an `if` statement, or multiple `return` statements in a function).

The resolve facility is symmetric: `resolve(T1,T2)` = `resolve(T2,T1)`. The resolve facility is always successful because type **Any** is returned if a less general type cannot be found. **Any** is represented by a record with two components, the first being the original type of the object and the second being the object itself. Thus anything can be coerced to an object of type **Any** and `resolve(Any,T)` = **Any** for all **T**.

Two other types have special resolve rules. Type **Void** has but one object and is the type returned by such operations as variable declaration, function definition, `if` statements without `else` clauses and `repeat` loops. Several functions also return the object of type **Void**, including those that display things in two dimensional algebaic, TeX[3] and FORTRAN forms. Like **Any**, `resolve(Void,T)` = **Void** for all **T**. Type **Exit** is used for `return` statements and error statements. Its rule is `resolve(Exit,T)` = **T** for all **T**.

After some checks for special cases, the resolve facility has three components.

1.4.1 Resolve by Coercion Query

If **T1** = **T2**, then the resolve of the pair is just **T1**. If **T1** can be coerced to **T2** and **T2** cannot be coerced to **T1**, then `resolve(T1,T2)` = **T2**. If **T1** and **T2** are coercable to one another, an arbitrary but canonical choice is made and returned.

1.4.2 Resolve by Rules

The interpreter has an internal database of rules which it tries to use to resolve two types. The rules are not complete, as they only attempt to take care of cases that cannot be dealt with in a more general way. Almost all of the rules deal with polynomials. For example, one rule is

```
resolve(Polynomial(T1),UnivariatePoly(x,T2))=
   resolve(Polynomial(T1),T2)
```

The variable of **UnivariatePoly** can be absorbed into the general constructor **Polynomial** and then the resolve facility is called again with different arguments. The second call may or may not use the rule system.

[3] TeX is a trademark of the American Mathematical Society

1.4.3 Resolve by Type Destructuring

This type of resolution is similar to the process involved in the coercion described in the last paragraph of "Coerce by Internal System Code". Given two towers of parameterized types, the interpreter tries to rearrange the towers and create a new type to which both of the original types are coerceable. This is a recursive process and involves using the coercion query facility to determine what tower rearrangements are possible.

1.5 An Example

As an example of coerce and resolve, we describe the inference involved in determining that the expression $x+1/2$ evaluates to an object of the datatype **Polynomial(RationalNumber)**. We assume x has not previously been given a value.

- Choose a default datatype of **Symbol** for x
- Choose the datatype of **Integer** for 1 and 2
- Look for an operation "/" in **Integer** that has two arguments, each an integer. It is not found.
- Start a general search in the library for an operation "/" with two **Integer** arguments. One is found in **RationalNumber** and applied
- Look for a "+" that takes a **Symbol** and a **RationalNumber**. None is found.
- Force x to an object of type **Polynomial(Integer)**
- Look for a "+" that takes a **Polynomial(Integer)** and a **RationalNumber**. None is found.
- Start a general search in the library for "+" operations. The only ones found take two arguments, each of the same datatype. Resolve **Polynomial(Integer)** and **RationalNumber** to get the datatype **Polynomial(RationalNumber)**
- Do the coercions

```
Polynomial(Integer) -> Polynomial(RationalNumber)
RationalNumber -> Polynomial(RationalNumber)
```

- Apply the "+" in the datatype **Polynomial(RationalNumber)** and return the result

1.6 Acknowledgement

In addition to the authors, Richard Jenks and Robert Sutor, three people have contributed significantly to the development of the Axiom interpreter. Scott C. Morrison (University of California, Berkeley) is responsible for the overall structure of the interpreter as it is today, having largely rewritten this part of the system in 1984. Albrecht Fortenbacher (University of Karlsruhe) rewrote and greatly extended the resolve and coerce facilities in 1985. Michael Lucks (Southern Methodist University) contributed to the coerce and modemap selection facilities in 1986.

Chapter 2

The Interpreter

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

Chapter 3

The Fundamental Data Structures

3.0.1 defvar $PatternVariableList

These are temporary variable names that will be replaced by FormalMapVariableList.

— **initvars** —

```
(defvar |$PatternVariableList|
  '(*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *20
    *21 *22 *23 *24 *25 *26 *27 *28 *29 *30 *31 *32 *33 *34 *35 *36 *37 *38
    *39 *40 *41 *42 *43 *44 *45 *46 *47 *48 *49 *50))
```

———-

3.0.2 defvar $FormalMapVariableList

— **initvars** —

```
(defvar |$FormalMapVariableList|
  '(|#1| |#2| |#3| |#4| |#5| |#6| |#7| |#8| |#9| |#10|
    |#11| |#12| |#13| |#14| |#15| |#16| |#17| |#18| |#19| |#20|
    |#21| |#22| |#23| |#24| |#25| |#26| |#27| |#28| |#29| |#30|
    |#31| |#32| |#33| |#34| |#35| |#36| |#37| |#38| |#39| |#40|
    |#41| |#42| |#43| |#44| |#45| |#46| |#47| |#48| |#49| |#50|))
```

———-

3.1 Frames and the Interpreter Frame Ring

Axiom has the notion of "frames". A frame is a data structure which holds all the vital data from an Axiom session.

The list of frames is structured as a ring. New frames can be added which will hold computations of independent information. The interpreter **)frame** command allows operations on frames. From the command line the user can create, modify, change, and delete frames.

3.2)frame Command

3.2.1 frame man page

— frame.help —

```
====================================================================
A.11.  )frame
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )frame  new  frameName
  - )frame  drop  [frameName]
  - )frame  next
  - )frame  last
  - )frame  names
  - )frame  import frameName [objectName1 [objectName2 ...] ]
  - )set message frame on | off
  - )set message prompt frame

Command Description:

A frame can be thought of as a logical session within the physical session
that you get when you start the system. You can have as many frames as you
want, within the limits of your computer's storage, paging space, and so on.
Each frame has its own step number, environment and history. You can have a
variable named a in one frame and it will have nothing to do with anything
that might be called a in any other frame.

Some frames are created by the HyperDoc program and these can have pretty
strange names, since they are generated automatically. To find out the names
of all frames, issue

)frame names

It will indicate the name of the current frame.

You create a new frame ``quark'' by issuing

)frame new quark

The history facility can be turned on by issuing either )set history on or
)history )on. If the history facility is on and you are saving history
information in a file rather than in the AXIOM environment then a history
```

```
file with filename quark.axh will be created as you enter commands. If you
wish to go back to what you were doing in the ‘‘initial’’ frame, use

)frame next

or

)frame last

to cycle through the ring of available frames to get back to ‘‘initial’’.

If you want to throw away a frame (say ‘‘quark’’), issue

)frame drop quark

If you omit the name, the current frame is dropped.

If you do use frames with the history facility on and writing to a file, you
may want to delete some of the older history files. These are directories, so
you may want to issue a command like rm -r quark.axh to the operating system.

You can bring things from another frame by using )frame import. For example,
to bring the f and g from the frame ‘‘quark’’ to the current frame, issue

)frame import quark f g

If you want everything from the frame ‘‘quark’’, issue

)frame import quark

You will be asked to verify that you really want everything.

There are two )set flags to make it easier to tell where you are.

)set message frame on | off

will print more messages about frames when it is set on. By default, it is
off.

)set message prompt frame

will give a prompt that looks like

initial (1) ->

when you start up. In this case, the frame name and step make up the prompt.

Also See:
o )history
o )set
```

———-

[1]

The frame mechanism uses several dollar 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 Axiom 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
```

Setting frame messages on will output a line detailing the current frame after every output is complete.

See the **set message frame**(p901) setmessageframe section for more details.

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.

3.3 Data Structures

The interpreter information is kept in a frame which is a 10 part data structure of the form (see 3.5.4). The parts of a frame and their initial, default values are:

1. **$interpreterFrameName**(p34) interpreterFrameName, a string, is the name of the current frame.
2. **$InteractiveFrame**(p34) InteractiveFrame which defaults to ((nil))
3. **$IOindex**(p34) IOindex also known as the step number, which defaults to 1
4. **$HiFiAccess**(p895) HiFiAccess
5. **$HistList**(p41) HistList
6. **$HistListLen**(p41) HistListLen
7. **$HistListAct**(p42) HistListAct
8. **$HistRecord**(p42) HistRecord
9. **$internalHistoryTable**(p42) internalHistoryTable which defaults to nil

[1] "history" (26.23.11 p 791) "set" (26.51.1 p 962)

10. **localExposureDataDefault**(p148) is a copy of the current local exposure data

There are a set of functions to manipulate frames. The internal set of frame functions are

- **initializeInterpreterFrameRing**(p23) creates the original frame ring, inserts an initial frame, and updates all the global variables from the initial frame.
- **emptyInterpreterFrame**(p24) creates a new, empty frame.
- **createCurrentInterpreterFrame**(p25) collects the environment into a frame.
- **updateFromCurrentInterpreterFrame**(p26) sets all of the global variables from the current frame.
- **frameEnvironment**(p27) returns the frameInteractive component of a named frame or a new, empty environment.
- **findFrameInRing**(p28) given the name, find the named frame
- **updateCurrentInterpreterFrame**(p27) 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.
- **nextInterpreterFrame**(p28) 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).
- **previousInterpreterFrame**(p29) moves to the previous frame in the ring.
- **changeToNamedInterpreterFrame**(p28) change to the named frame.
- **addNewInterpreterFrame**(p29) update the current frame, initialize the history, make a new empty frame, and initialize all of the global variables from the empty frame.
- **closeInterpreterFrame**(p33) when there is more than one frame, delete the current frame and initialize all the global variables from the next frame in the ring.
- **displayFrameNames**(p25) print all the frame names and indicate which one is the current frame.
- **importFromFrame**(p30) imports items from a different frame into the current frame

3.4 Frame Access Macros

First Frame Component – frameName

3.4.1 defmacro frameName

type : **FrameName** → **Symbol**
frameName : **Frame** → **FrameName**
— **defmacro frameName 0** —

```
(defmacro frameName (frame)
 `(first ,frame))
```

———-

Second Frame Component – frameInteractive

3.4.2 defmacro frameInteractive

frameInteractive : **Frame** → **Interactive**
— **defmacro frameInteractive 0** —

```
(defmacro frameInteractive (frame)
 `(second ,frame))
```

———-

Third Frame Component – frameIOIndex

3.4.3 defmacro frameIOIndex

frameIOIndex : **Frame** → **IOIndex**
— **defmacro frameIOIndex 0** —

```
(defmacro frameIOIndex (frame)
 `(third ,frame))
```

———-

Fourth Frame Component – frameHiFiAccess

3.4.4 defmacro frameHiFiAccess

frameHiFiAcecss : **Frame** → **HiFiAccess**
— **defmacro frameHiFiAccess 0** —

```
(defmacro frameHiFiAccess (frame)
 `(fourth ,frame))
```

———-

Fifth Frame Component – frameHistList

3.4.5 defmacro frameHistList

frameHistList : **Frame** → **HistList**
— **defmacro frameHistList 0** —

```
(defmacro frameHistList (frame)
 `(fifth ,frame))
```

———-

Sixth Frame Component – frameHistListLen

3.4.6 defmacro frameHistListLen

type : HistListLen → NonNegativeInteger
frameHistListLen : Frame → HistListLen
— defmacro frameHistListLen 0 —

```
(defmacro frameHistListLen (frame)
 `(sixth ,frame))
```

———-

Seventh Frame Component – frameHistListAct

3.4.7 defmacro frameHistListAct

frameHistListAct : Frame → HistListAct
— defmacro frameHistListAct 0 —

```
(defmacro frameHistListAct (frame)
 `(seventh ,frame))
```

———-

Eighth Frame Component – frameHistRecord

3.4.8 defmacro frameHistRecord

frameHistRecord : Frame → HistRecord
— defmacro frameHistRecord 0 —

```
(defmacro frameHistRecord (frame)
 `(eighth ,frame))
```

———-

Ninth Frame Component – frameHistoryTable

3.4.9 defmacro frameHistoryTable

frameHistoryTable : Frame → HistoryTable
— defmacro frameHistoryTable 0 —

```
(defmacro frameHistoryTable (frame)
 `(ninth ,frame))
```

———-

Tenth Frame Component – frameExposureData

3.4.10 defmacro frameExposureData

frameExposureData : **Frame** → **ExposureData**
enum : **FrameArgs** → **(nil,drop,import,last,names,new,next)**
frameSpad2Cmd : **FrameArgs** → **nil**
— **defmacro frameExposureData 0** —

```
(defmacro frameExposureData (frame)
 `(tenth ,frame))
```

———-

3.5 Functions to manipulate frames

3.5.1 The top level frame command

[frameSpad2Cmd p22]

Frame : **nil** → **nil**
— **defun frame** —

```
(defun |frame| (l)
 "The top level frame command"
 (|frameSpad2Cmd| l))
```

———-

3.5.2 The top level frame command handler

[throwKeyedMsg p**??**]
[helpSpad2Cmd p782]
[selectOptionLC p728]
[qcdr p**??**]
[qcar p**??**]
[object2Identifier p**??**]
[frameSpad2Cmd drop (vol9)]
[closeInterpreterFrame p33]
[import p**??**]
[importFromFrame p30]
[last p**??**]

[previousInterpreterFrame p29]
[names p**??**]
[displayFrameNames p25]
[new p**??**]
[addNewInterpreterFrame p29]
[next p298]
[nextInterpreterFrame p28]
[$options p63]

— **defun frameSpad2Cmd** —

```
(defun |frameSpad2Cmd| (args)
 "The top level frame command handler"
 (let (frameArgs arg a)
  (declare (special |$options|))
  (setq frameArgs '(|drop| |import| |last| |names| |new| |next|))
  (cond
   (|$options|
    (|throwKeyedMsg| "The %1 system command takes arguments but no 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| "%1 is not a valid frame name."
             (cons args nil))
          (|closeInterpreterFrame| args)))
      (|import| (|importFromFrame| args))
      (|last|   (|previousInterpreterFrame|))
      (|names|  (|displayFrameNames|))
      (|new|
        (if (and args (consp args))
          (|throwKeyedMsg| "%1 is not a valid frame name."
             (cons args nil))
          (|addNewInterpreterFrame| args)))
      (|next| (|nextInterpreterFrame|))
      (t nil))))))
```

———-

3.5.3 Initializing the Interpreter Frame Ring

There can be multiple frames and these live in a top-level variable called **$interpreterFrameRing**. This variable holds a circular list of frames.

This function creates an empty, initial frame named "initial" and creates a list of frames containing an empty frame. This list is the interpreter frame ring and is not actually circular but is managed as a circular list.

As a final step we update the world from this frame. This has the side-effect of resetting all the important global variables to their initial values.

[emptyInterpreterFrame p24]
[updateFromCurrentInterpreterFrame p26]
[$interpreterFrameName p34]
[$interpreterFrameRing p34]

emptyInterpreterFrame : **Symbol** → **Frame**
— **defun initializeInterpreterFrameRing** —

```
(defun |initializeInterpreterFrameRing| ()
 "Initializing the Interpreter Frame Ring"
 (declare (special |$interpreterFrameName| |$interpreterFrameRing|))
 (setq |$interpreterFrameName| '|initial|)
 (setq |$interpreterFrameRing|
   (list (|emptyInterpreterFrame| |$interpreterFrameName|)))
 (|updateFromCurrentInterpreterFrame|)
 nil)
```

———-

3.5.4 Create a new, empty Interpreter Frame

[$HiFiAccess p895]
[$HistList p41]
[$HistListLen p41]
[$HistListAct p42]
[$HistRecord p42]
[$localExposureDataDefault p148]

emptyInterpreterFrame : **Symbol** → **Frame**
— **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
```

———-

3.5.5 Create 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 p34]

frameNames : nil → List Symbol
— defun frameNames 0 —

```
(defun |frameNames| ()
 "Creating a List of all of the Frame Names"
 (declare (special |$interpreterFrameRing|))
 (mapcar #'(lambda (f) (frameName f)) |$interpreterFrameRing|))
```

———-

3.5.6 Display the frame name list message

[bright p??]
[framename p??]
[$interpreterFrameRing p34]

displayFrameNames : nil → nil
— defun displayFrameNames 0 —

```
(defun |displayFrameNames| ()
 "Display the Frame Names"
 (declare (special |$interpreterFrameRing|))
 (format t "   The names of the existing frames are:~%")
 (format t "~{            ~a ~%~^~}" (|frameNames|))
 (format t "      The current frame is the first one listed.~%"))
```

———-

3.5.7 Collect the global variables 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.

[$interpreterFrameName p34]
[$InteractiveFrame p34]
[$IOindex p34]
[$HiFiAccess p895]
[$HistList p41]
[$HistListLen p41]
[$HistListAct p42]
[$HistRecord p42]
[$internalHistoryTable p42]

[$localExposureData p147]

createCurrentInterpreterFrame : nil → Frame
— defun createCurrentInterpreterFrame 0 —

```
(defun |createCurrentInterpreterFrame| ()
 "Collecting up the Environment into a Frame"
 (declare (special |$interpreterFrameName| |$InteractiveFrame| |$IOindex|
   |$HiFiAccess| |$HistList| |$HistListLen| |$HistListAct| |$HistRecord|
   |$internalHistoryTable| |$localExposureData|))
 (list
   |$interpreterFrameName|
   |$InteractiveFrame|
   |$IOindex|
   |$HiFiAccess|
   |$HistList|
   |$HistListLen|
   |$HistListAct|
   |$HistRecord|
   |$internalHistoryTable|
   |$localExposureData|))
```

———-

3.5.8 Update global variables from the Current Frame

The frames are kept on a circular list. The first element on that list is known as "the current frame". This will initialize all of the interesting interpreter data structures from that frame.
[sayMessage p**??**]
[$interpreterFrameRing p34]
[$interpreterFrameName p34]
[$InteractiveFrame p34]
[$IOindex p34]
[$HiFiAccess p895]
[$HistList p41]
[$HistListLen p41]
[$HistListAct p42]
[$HistRecord p42]
[$internalHistoryTable p42]
[$localExposureData p147]
[$frameMessages p901]

updateFromCurrentInterpreterFrame : nil → nil
— defun updateFromCurrentInterpreterFrame —

```
(defun |updateFromCurrentInterpreterFrame| ()
 "Update from the Current Frame"
 (let (tmp1)
  (declare (special |$interpreterFrameRing| |$interpreterFrameName|
    |$InteractiveFrame| |$IOindex| |$HiFiAccess| |$HistList| |$HistListLen|
    |$HistListAct| |$HistRecord| |$internalHistoryTable| |$localExposureData|
    |$frameMessages|))
```

```
(setq tmp1 (first |$interpreterFrameRing|))
(setq |$interpreterFrameName| (frameName tmp1))
(setq |$InteractiveFrame|     (frameInteractive tmp1))
(setq |$IOindex|              (frameIOIndex tmp1))
(setq |$HiFiAccess|           (frameHiFiAccess tmp1))
(setq |$HistList|             (frameHistList tmp1))
(setq |$HistListLen|          (frameHistListLen tmp1))
(setq |$HistListAct|          (frameHistListAct tmp1))
(setq |$HistRecord|           (frameHistRecord tmp1))
(setq |$internalHistoryTable| (frameHistoryTable tmp1))
(setq |$localExposureData|    (frameExposureData tmp1))
(when |$frameMessages|
 (format t "   Current interpreter frame is called ~a"
   |$interpreterFrameName|))))
```

———-

3.5.9 Replace the current frame and update from the globals

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 p25]
[updateFromCurrentInterpreterFrame p26]
[$interpreterFrameRing p34]

updateCurrentInterpreterFrame : nil → nil
— defun updateCurrentInterpreterFrame —

```
(defun |updateCurrentInterpreterFrame| ()
  "Update the Current Interpreter Frame"
  (declare (special |$interpreterFrameRing|))
  (rplaca |$interpreterFrameRing| (|createCurrentInterpreterFrame|))
  (|updateFromCurrentInterpreterFrame|))
```

———-

3.5.10 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 p20]

frameEnvironment : FrameName → nil
— defun frameEnvironment —

```
(defun |frameEnvironment| (fname)
 "Get Named Frame Environment (aka Interactive)"
 (let ((frame (|findFrameInRing| fname)))
  (if frame
```

```
  (frameInteractive frame)
  (list (list nil)))))
```

———-

3.5.11 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 p19]
[$interpreterFrameRing p34]

findFrameInRing : **FrameName** → **Union(Frame,nil)**
— **defun findFrameInRing 0** —

```
(defun |findFrameInRing| (name)
 "Find a Frame in the Frame Ring by Name"
  (declare (special |$interpreterFrameRing|))
  (dolist (frame |$interpreterFrameRing|)
    (when (eq (frameName frame) name) (return frame))))
```

———-

3.5.12 Change to the Named Interpreter Frame

[updateCurrentInterpreterFrame p27]
[findFrameInRing p28]
[updateFromCurrentInterpreterFrame p26]
[$interpreterFrameRing p34]

changeToNamedInterpreterFrame : **FrameName** → **nil**
— **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 (delete |$interpreterFrameRing| frame :test #'equal)))
     (|updateFromCurrentInterpreterFrame|))))
```

———-

3.5.13 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 p26]
[$interpreterFrameRing p34]

nextInterpreterFrame : nil → nil
— **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|)))
```

———-

3.5.14 Move to the previous Interpreter Frame in Ring

[updateCurrentInterpreterFrame p27]
[updateFromCurrentInterpreterFrame p26]
[$interpreterFrameRing p34]

previousInterpreterFrame : nil → nil
— **defun previousInterpreterFrame** —

```
(defun |previousInterpreterFrame| ()
 "Move to the previous Interpreter Frame in Ring"
 (let (tmp1 l b)
 (declare (special |$interpreterFrameRing|))
  (|updateCurrentInterpreterFrame|)
  (when  (cdr |$interpreterFrameRing|)
    (setq tmp1 (reverse |$interpreterFrameRing|))
    (setq l (car tmp1))
    (setq b (nreverse (cdr tmp1)))
    (setq |$interpreterFrameRing| (nconc (cons l nil) b))
    (|updateFromCurrentInterpreterFrame|))))
```

———-

3.5.15 Add a New Interpreter Frame

[boot-equal p**??**]
[framename p**??**]
[throwKeyedMsg p**??**]
[updateCurrentInterpreterFrame p27]
[initHistList p790]
[emptyInterpreterFrame p24]
[updateFromCurrentInterpreterFrame p26]

[$erase p??]
[histFileName p789]
[$interpreterFrameRing p34]

— **defun addNewInterpreterFrame** —

```
(defun |addNewInterpreterFrame| (name)
 "Add a New Interpreter Frame"
 (declare (special |$interpreterFrameRing|))
 (if (null name)
   (|throwKeyedMsg| "You must provide a name for the new frame." nil)
   (progn
     (|updateCurrentInterpreterFrame|)
     (dolist (f |$interpreterFrameRing|)
       (when (eq name (frameName f)) ; existing frame with same name
         (|throwKeyedMsg|
          (format nil
            " You cannot use the name %1 for a new frame because an existing ~
              frame already has that name.")
          (list name))))
     (|initHistList|)
     (setq |$interpreterFrameRing|
       (cons (|emptyInterpreterFrame| name) |$interpreterFrameRing|))
     (|updateFromCurrentInterpreterFrame|)
     ($erase (|histFileName|)))))
```

———-

3.5.16 Import items from another frame

[member p1108]
[frameNames p25]
[throwKeyedMsg p??]
[boot-equal p??]
[framename p??]
[frameEnvironment p27]
[upcase p1140]
[queryUserKeyedMsg p??]
[string2id-n p??]
[importFromFrame p30]
[sayKeyedMsg p39]
[clearCmdParts p747]
[seq p??]
[exit p??]
[putHist p800]
[get p??]
[getalist p??]
[$interpreterFrameRing p34]

— **defun importFromFrame** —

```
(defun |importFromFrame| (args)
```

```
"Import items from another frame"
(prog (temp1 fname fenv x v props vars plist prop val m)
(declare (special |$interpreterFrameRing|))
 (when (and args (atom args)) (setq args (cons args nil)))
 (if (null args)
  (|throwKeyedMsg|
   (format nil ")frame import must be followed by the frame name. The names~
    of objects in that frame can then optionally follow the frame name.~
    For example,~
    %ceon )frame import calculus %ceoff ~
    imports all objects in the calculus frame, and ~
    %ceon )frame import calculus epsilon delta %ceoff ~
    imports the objects named epsilon and delta from the ~
    frame calculus. ~
    Please note that if the current frame contained any information ~
    about objects with these names, then that information would be ~
    cleared before the import took place.")
   nil)
  (progn
    (setq temp1 args)
    (setq fname (car temp1))
    (setq args (cdr temp1))
    (cond
     ((null (|member| fname (|frameNames|)))
       (|throwKeyedMsg|
        (format nil " You cannot import anything from the frame %1 because ~
                      that is not the name of an existing frame.")
        (cons fname nil)))
     ((boot-equal fname (frameName (car |$interpreterFrameRing|)))
       (|throwKeyedMsg|
        "You cannot import from the current frame (nor is there a need!)."
        nil))
     (t
       (setq fenv (|frameEnvironment| fname))
       (cond
        ((null args)
          (setq x
            (upcase (|queryUserKeyedMsg|
             (format nil "Do you really want to import everything from the ~
                          frame %1? If so, please enter y or yes :")
             (cons fname nil))))
          (cond
           ((member (string2id-n x 1) '(y yes))
             (setq vars nil)
             (do ((tmp0 (caar fenv) (cdr tmp0)) (tmp1 nil))
                 ((or (atom tmp0)
                      (progn (setq tmp1 (car tmp0)) nil)
                      (progn
                       (progn
                        (setq v (car tmp1))
                        (setq props (cdr tmp1))
                        tmp1)
                       nil))
                  nil)
```

```
      (cond
       ((eq v '|--macros|)
         (do ((tmp2 props (cdr tmp2))
              (tmp3 nil))
             ((or (atom tmp2)
                  (progn (setq tmp3 (car tmp2)) nil)
                  (progn
                   (progn (setq m (car tmp3)) tmp3)
                   nil))
              nil)
           (setq vars (cons m vars))))
       (t (setq vars (cons v vars)))))
     (|importFromFrame| (cons fname vars)))
   (t
     (|sayKeyedMsg| "AXIOM will not import everything from frame %1."
      (cons fname nil)))))
(t
 (do ((tmp4 args (cdr tmp4)) (v nil))
     ((or (atom tmp4) (progn (setq v (car tmp4)) nil)) nil)
  (seq
   (exit
    (progn
     (setq plist (getalist (caar fenv) v))
     (cond
      (plist
       (|clearCmdParts| (cons '|propert| (cons v nil)))
       (do ((tmp5 plist (cdr tmp5)) (tmp6 nil))
           ((or (atom tmp5)
                (progn (setq tmp6 (car tmp5)) nil)
                (progn
                 (progn
                  (setq prop (car tmp6))
                  (setq val (cdr tmp6))
                  tmp6)
                 nil))
            nil)
         (seq
          (exit (|putHist| v prop val |$InteractiveFrame|)))))
      ((setq m (|get| '|--macros--| v fenv))
        (|putHist| '|--macros--| v m |$InteractiveFrame|))
      (t
        (|sayKeyedMsg|
         (format nil "AXIOM cannot import %1 from frame %2 because ~
                      it cannot be found.")
          (cons v (cons fname nil)))))))))
 (|sayKeyedMsg|
   (format nil "Import from frame %1 is complete. Please issue ~
                )display all if you wish to see the contents of ~
                the current frame.")
   (cons fname nil))))))))))
```

———-

3.5.17 Close an Interpreter Frame

[framename p??]
[throwKeyedMsg p??]
[$erase p??]
[makeHistFileName p789]
[updateFromCurrentInterpreterFrame p26]
[$interpreterFrameRing p34]
[$interpreterFrameName p34]

closeInterpreterFrame : **FrameName** → **nil**
— **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|
        (format nil "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 %1.")
          (cons |$interpreterFrameName| nil))
       (|throwKeyedMsg|
        (format nil "The current frame is the only active one. Issue ~
                     )clear all to clear its contents.")
        nil))
     (progn
       (if (null name)
        (setq |$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|
             "There is no frame called %1. Your command cannot be processed."
            (cons name nil))
           (progn
             ($erase (|makeHistFileName| name))
             (setq |$interpreterFrameRing| (nreverse ifr))))))
       (|updateFromCurrentInterpreterFrame|)))))
```

————-

3.6 Global variables associated with the frame

3.6.1 defvar $interpreterFrameRing

All existing frames are kept in a ring held in this variable.

— **initvars** —

```
(defvar |$interpreterFrameRing| nil "The ring of all frames")
```

———-

3.6.2 defvar $interpreterFrameName

The `$interpreterFrameName` variable, set in initializeInterpreterFrameRing to the constant initial to indicate that this is the initial (default) frame.

Frames are structures that capture all of the variables defined in a session. There can be multiple frames and the user can freely switch between them. Frames are kept in a ring data structure so you can move around the ring.

— **initvars** —

```
(defvar |$interpreterFrameName| '|initial|)
```

———-

3.6.3 defvar $InteractiveFrame

`$InteractiveFrame` 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.

This variable is set in the **restart** function as the value returned by **makeInitialModemapFrame**—.

— **initvars** —

```
(defvar |$InteractiveFrame| nil)
```

———-

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.

3.6.4 defvar $IOindex

— **initvars** —

```
(defvar $IOindex 1 "The current Axiom prompt number")
```

3.7 Interpreter Functions using Frames

The **??** function

The **u**(p**??**)ndoSteps function, part of the undo mechanism can reset the `$InteractiveFrame`.

Chapter 4

The Message Mechanism

Throughout the interpreter there are messages printed using a symbol for a database lookup. This was done to enable translation of these messages languages other than English.

Axiom messages are read from a flat file database and returned as one long string. They are preceded in the database by a key and this is how they are referenced from code. For example, one key is S2IL0001 which means:

```
S2          Scratchpad II designation
I           from the interpreter
L           originally from LISPLIB BOOT
0001        a sequence number
```

Each message may contain formatting codes and and parameter codes. The formatting codes are:

```
%ceoff      turn off centering
%ceon       turn on centering
%d          turn off bright printing
%f          user defined printing
%i          start indentation of 3 more spaces
%l          start a new line
%m          math-print an expression
%rjoff      turn off right justification (actually ragged left)
%rjon       turn on right justification (actually ragged left)
%s          pretty-print as an S-expression
%u          unindent 3 spaces
%x#         insert # spaces
```

The parameter codes look like %1, %2b, %3p, %4m, %5bp, %6s where the digit is the parameter number and the letters following indicate additional formatting. You can indicate as many additional formatting qualifiers as you like, to the degree they make sense.

- The "p" code means to call prefix2String on the parameter, a standard way of printing abbreviated types.
- The "P" operator maps prefix2String over its arguments.
- The "o" operation formats the argument as an operation name.
- The "b" means to print that parameter in a bold (bright) font.

- The “c” means to center that parameter on a new line.
- The “r” means to right justify (ragged left) the argument.
- The “f” means that the parameter is a list [fn, :args] and that “fn” is to be called on “args” to get the text.

4.0.1 defvar $msgAlist

— **initvars** —

```
(defvar |$msgAlist| nil)
```

———-

4.0.2 defvar $testingErrorPrefix

— **initvars** —

```
(defvar |$testingErrorPrefix| "Daly Bug")
```

———-

4.0.3 defvar $msgdbPrims

— **initvars** —

```
(defvar |$msgdbPrims|
 '(|%b| |%d| |%l| |%i| |%u| %U |%n| |%x| |%ce| |%rj| "%U" "%b" "%d"
   "%l" "%i" "%u" "%U" "%n" "%x" "%ce" "%rj"))
```

———-

4.0.4 defvar $msgdbPunct

— **initvars** —

```
(defvar |$msgdbPunct|
  '(|.| |,| ! |:| |;| ? ] |)| "." "," "!" ":" ";" "?" "]" ")"))
```

———-

4.0.5 defvar $msgdbNoBlanksBeforeGroup

— **initvars** —

```
(defvar |$msgdbNoBlanksBeforeGroup|
 `(" " | | "%" % ,@|$msgdbPrims| ,@|$msgdbPunct|))
```

———-

4.0.6 defvar $msgdbNoBlanksAfterGroup

— **initvars** —

```
(defvar |$msgdbNoBlanksAfterGroup|
 `(" " | | "%" % ,@|$msgdbPrims| [ |(| "[" "("))
```

———-

4.0.7 defun Say a message using a keyed lookup

[sayKeyedMsgLocal p39]

— **defun sayKeyedMsg** —

```
(defun |sayKeyedMsg| (key args)
  (|sayKeyedMsgLocal| key args))
```

———-

4.0.8 defun Handle msg formatting and print to file

[segmentKeyedMsg p40]
[substituteSegmentedMsg p**??**]
[flowSegmentedMsg p**??**]
[sayMSG2File p40]
[sayMSG p40]
[$printMsgsToFile p901]
[$linelength p936]
[$margin p935]
[$displayMsgNumber p906]

— **defun sayKeyedMsgLocal** —

```
(defun |sayKeyedMsgLocal| (key args)
 (let (msg msgp)
 (declare (special |$printMsgsToFile| $linelength $margin |$displayMsgNumber|))
  (setq msg (|segmentKeyedMsg| key))
  (setq msg (|substituteSegmentedMsg| msg args))
  (when |$displayMsgNumber| (setq msg `(,key |:| . ,msg)))
  (setq msgp (|flowSegmentedMsg| msg $linelength $margin))
  (when |$printMsgsToFile| (|sayMSG2File| msgp))
  (|sayMSG| msgp)))
```

———-

4.0.9 defun Break a message into words

[string2Words p??]

— **defun segmentKeyedMsg** —

```
(defun |segmentKeyedMsg| (msg) (|string2Words| msg))
```

———-

4.0.10 defun Write a msg into spadmsg.listing file

[makePathname p1104]
[defiostream p1046]
[sayBrightly1 p1110]
[shut p1046]

— **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)))
```

———-

4.0.11 defun sayMSG

[saybrightly1 p??]
[$algebraOutputStream p920]

— **defun sayMSG** —

```
(defun |sayMSG| (x)
 (declare (special |$algebraOutputStream|))
 (when x (sayBrightly1 x |$algebraOutputStream|)))
```

———-

Chapter 5

The History Mechanism

5.0.12 defvar $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. This is remembered in the current frame.

— **initvars** —

```
(defvar |$HiFiAccess| nil "Is the history function on?")
```

————-

5.0.13 defvar $HistList

Thie `$HistList` variable is set by initHistList to an initial value of NIL elements. The last element of the list is smashed to point to the first element to make the list circular. This is a circular list of length `$HistListLen`. This is remembered in the current frame.

— **initvars** —

```
(defvar |$HistList| nil "A circular list of history elements")
```

————-

5.0.14 defvar $HistListLen

The `$HistListLen` variable is set by initHistList to 20. This is the length of a circular list maintained in the variable `$HistList`. This is remembered in the current frame.

— **initvars** —

```
(defvar |$HistListLen| 0 "The length of the circular history list")
```

————-

5.0.15 defvar $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. This is remembered in the current frame.

— **initvars** —

```
(defvar |$HistLIstAct| 0 "The number of un-doable steps")
```

————-

5.0.16 defvar $internalHistoryTable

The `$internalHistoryTable` variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism. This is remembered in the current frame.

— **initvars** —

```
(defvar |$internalHistoryTable| nil)
```

————-

5.0.17 defvar $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.

— **initvars** —

```
(defvar |$HistRecord| nil)
```

————-

5.0.18 defvar $historyFileType

The `$historyFileType` is set at load time by a call to initvars to a value of "axh". It appears that this is intended to be used as a filetype extension. It is part of the history mechanism. It is used in makeHistFileName as part of the history file name.

— initvars —

```
(defvar |$historyFileType| nil)
```

———-

Chapter 6

The undo mechanism

6.1 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.

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`.

6.2 Initial Undo Variables

6.2.1 defvar $frameRecord

— **initvars** —

```
(defvar |$frameRecord| nil "a list of value changes")
```

———-

6.2.2 defvar $previousBindings

— **initvars** —

```
(defvar |$previousBindings| nil "a copy of Interactive Frame info for undo")
```

————-

6.2.3 defvar $reportundo

— **initvars** —

```
(defvar |$reportundo| nil "t means we report the steps undo takes")
```

————-

6.3 The undo functions

6.3.1 defun undo

[stringPrefix? p1254]
[pname p1106]
[read p838]
[userError p??]
[qcdr p??]
[qcar p??]
[identp p1107]
[undoSteps p47]
[undoCount p54]
[$options p63]
[$InteractiveFrame p34]

— **defun undo** —

```
(defun |undo| (l)
 (let (tmp1 key s undoWhen n)
 (declare (special |$options| |$InteractiveFrame|))
  (setq undoWhen '|after|)
  (when
    (and (consp |$options|)
         (eq (qcdr |$options|) nil)
         (progn
          (setq tmp1 (qcar |$options|))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq key (qcar tmp1)) t)))
     (cond
      ((|stringPrefix?| (setq s (pname key)) "redo")
        (setq |$options| nil)
        (|read| '(|redo.input|)))
      ((null (|stringPrefix?| s "before"))
        (|userError| "only option to undo is \")redo\""))
```

```
      (t
        (setq undoWhen '|before|))))))
  (if (null l)
    (setq n (- 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))
```

———-

6.3.2 defun undoSteps

```
-- undoes m previous commands; if )before option, then undo one extra at end
--Example: if $IOindex now is 6 and m = 2 then general layout of $frameRecord,
--  after the call to recordFrame below will be:
--  (<change for systemcommands>
--  (<change for #5> <change for system commands>
--  (<change for #4> <change for system commands>
--  (<change for #3> <change for system commands>
--   <change for #2> <change for system commands>
--   <change for #1> <change for system commands>) where system
--  command entries are optional and identified by (systemCommand . change).
--  For a ")undo 3 )after", m = 2 and undoStep swill restore the environment
--  up to, but not including <change for #3>.
--  An "undo 3 )before" will additionally restore <change for #3>.
--  Thus, the later requires one extra undo at the end.
```

[writeInputLines p797]
[recordFrame p1001]
[copy p??]
[undoSingleStep p48]
[qcdr p??]
[qcar p??]
[$IOindex p34]
[$InteractiveFrame p34]
[$frameRecord p45]

— **defun undoSteps** —

```
(defun |undoSteps| (m beforeOrAfter)
 (let (tmp1 tmp2 systemDelta lastTailSeen env)
 (declare (special |$IOindex| |$InteractiveFrame| |$frameRecord|))
  (|writeInputLines| '|redo| (- |$IOindex| m))
  (recordFrame 'normal)
  (setq env (copy (caar |$InteractiveFrame|)))
  (do ((i 0 (1+ i)) (framelist |$frameRecord| (cdr framelist)))
      ((or (> i m) (atom framelist)) nil)
    (setq env (|undoSingleStep| (CAR framelist) env))
    (if (and (consp framelist)
```

```
            (progn
             (setq tmp1 (qcdr framelist))
             (and (consp tmp1)
                  (progn
                   (setq tmp2 (qcar tmp1))
                   (and (consp tmp2)
                        (eq (qcar tmp2) '|systemCommand|)
                        (progn
                         (setq systemDelta (qcdr tmp2))
                         t))))))
      (progn
        (setq framelist (cdr framelist))
        (setq env (|undoSingleStep| systemDelta env)))
      (setq lastTailSeen framelist)))
  (cond
   ((eq beforeOrAfter '|before|)
     (setq env (|undoSingleStep| (car (cdr lastTailSeen)) env))))
  (setq |$frameRecord| (cdr |$frameRecord|))
  (setq |$InteractiveFrame| (list (list env)))))
```

———-

6.3.3 defun undoSingleStep

```
undoSingleStep(changes,env) ==
--Each change is a name-proplist pair. For each change:
--  (1) if there exists a proplist in env, then for each prop-value change:
--      (a) if the prop exists in env, RPLAC in the change value
--      (b) otherwise, CONS it onto the front of prop-values for that name
--  (2) add change to the front of env
--  pp '"----Undoing 1 step--------"
--  pp changes
```

[assq p1110]
[seq p??]
[exit p??]
[lassoc p??]
[undoLocalModemapHack p49]

— **defun undoSingleStep** —

```
(defun |undoSingleStep| (changes env)
 (prog (name changeList pairlist proplist prop value node)
  (return
   (seq
    (progn
     (do ((tmp0 changes (cdr tmp0)) (|change| nil))
         ((or (atom tmp0)
              (progn (setq |change| (car tmp0)) nil)
              (progn
               (progn
                (setq name (car |change|))
                (setq changeList (cdr |change|))
```

```
              |change|)
             nil))
           nil)
      (seq
       (exit
        (progn
         (when (lassoc '|localModemap| changeList)
           (setq changeList (|undoLocalModemapHack| changeList)))
         (cond
          ((setq pairlist (assq name env))
            (cond
             ((setq proplist (cdr pairlist))
               (do ((tmp1 changeList (cdr tmp1)) (pair nil))
                   ((or (atom tmp1)
                        (progn (setq pair (car tmp1)) nil)
                        (progn
                         (progn
                          (setq prop (car pair))
                          (setq value (cdr pair))
                          pair)
                          nil))
                        nil)
                (seq
                 (exit
                  (cond
                   ((setq node (assq prop proplist))
                     (rplacd node value))
                   (t
                     (rplacd proplist
                       (cons (car proplist) (cdr proplist)))
                     (rplaca proplist pair)))))))
              (t (rplacd pairlist changeList))))
           (t
             (setq env (cons |change| env))))))))
     env)))))
```

———-

6.3.4 defun undoLocalModemapHack

[seq p**??**]
[exit p**??**]

— **defun undoLocalModemapHack** —

```
(defun |undoLocalModemapHack| (changeList)
 (prog (name value)
  (return
   (seq
    (prog (tmp0)
     (setq tmp0 nil)
     (return
      (do ((tmp1 changeList (cdr tmp1)) (pair nil))
```

```
          ((or (atom tmp1)
               (progn (setq pair (car tmp1)) nil)
               (progn
                (progn
                 (setq name (car pair))
                 (setq value (cdr pair))
                 pair)
                nil))
              (nreverse0 tmp0))
       (seq
        (exit
         (cond
          ((cond
             ((eq name '|localModemap|) (cons name nil))
             (t pair))
           (setq tmp0
            (cons
             (cond
              ((eq name '|localModemap|) (cons name nil))
              (t pair)) tmp0))))))))))))
```

———-

6.3.5 Remove undo lines from history write

Removing undo lines from `)hist )write linelist` [stringPrefix? p1254]
[seq p**??**]
[exit p**??**]
[trimString p**??**]
[substring p293]
[charPosition p**??**]
[maxindex p**??**]
[undoCount p54]
[concat p1107]
[$currentLine p**??**]
[$IOindex p34]

— **defun removeUndoLines** —

```
(defun |removeUndoLines| (u)
 "Remove undo lines from history write"
 (prog (xtra savedIOindex s s1 m s2 x code c n acc)
 (declare (special |$currentLine| |$IOindex|))
  (return
   (seq
    (progn
     (setq xtra
      (cond
       ((stringp |$currentLine|) (cons |$currentLine| nil))
       (t (reverse |$currentLine|))))
     (setq xtra
      (prog (tmp0)
```

```
  (setq tmp0 nil)
  (return
   (do ((tmp1 xtra (cdr tmp1)) (x nil))
       ((or (atom tmp1)
            (progn (setq x (car tmp1)) nil))
          (nreverse0 tmp0))
    (seq
     (exit
      (cond
       ((null (|stringPrefix?| ")history" x))
         (setq tmp0 (cons x tmp0)))))))))))
(setq u (append u xtra))
(cond
 ((null
   (prog (tmp2)
    (setq tmp2 nil)
    (return
     (do ((tmp3 nil tmp2) (tmp4 u (cdr tmp4)) (x nil))
         ((or tmp3 (atom tmp4) (progn (setq x (car tmp4)) nil)) tmp2)
      (seq
       (exit
        (setq tmp2
          (or tmp2 (|stringPrefix?| ")undo" x)))))))))) u)
 (t
   (setq savedIOindex |$IOindex|)
   (setq |$IOindex| 1)
   (do ((y u (cdr y)))
       ((atom y) nil)
    (seq
     (exit
      (cond
       ((eql (elt (setq x (car y)) 0) #\) )
         (cond
          ((|stringPrefix?| ")undo"
                            (setq s (|trimString| x)))
           (setq s1 (|trimString| (substring s 5 nil)))
           (cond
             ((not (string= s1 ")redo"))
               (setq m (|charPosition| #\) s1 0))
               (setq code
                (cond
                  ((> (maxindex s1) m) (elt s1 (1+ m)))
                  (t #\a)))
               (setq s2 (|trimString| (substring s1 0 m)))))
           (setq n
            (cond
             ((string= s1 ")redo")
                0)
             ((not (string= s2 ""))
               (|undoCount| (parse-integer s2)))
             (t -1)))
           (rplaca y
             (concat ">" code (princ-to-string n))))
          (t nil)))
```

```
        (t (setq |$IOindex| (1+ |$IOindex|)))))))
    (setq acc nil)
    (do ((y (nreverse u) (cdr y)))
        ((atom y) nil)
     (seq
      (exit
       (cond
        ((eql (elt (setq x (car y)) 0) #\>)
          (setq code (elt x 1))
          (setq n (parse-integer (substring x 2 nil)))
          (setq y (cdr y))
          (do ()
              ((null y) nil)
           (seq
            (exit
             (progn
              (setq c (car y))
              (cond
               ((or (eql (elt c 0) #\))
                    (eql (elt c 0) #\>))
                 (setq y (cdr y)))
               ((eql n 0)
                 (return nil))
               (t
                 (setq n (- n 1))
                 (setq y (cdr y))))))))
          (cond
           ((and y (not (eql code #\b)))
             (setq acc (cons c acc)))))
        (t (setq acc (cons x acc)))))))
    (setq |$IOindex| savedIOindex)
    acc)))))))
```

6.3.6 defun reportUndo

This function is enabled by setting `$reportundo` to a non-nil value. An example of the output generated is:

```
r := binary(22/7)


          ---
   (1)  11.001
                                                    Type: BinaryExpansion
Properties of % ::
  value was: NIL
  value is:  ((|BinaryExpansion|) WRAPPED . #(1 (1 1) NIL (0 0 1)))
Properties of r ::
  value was: NIL
  value is:  ((|BinaryExpansion|) WRAPPED . #(1 (1 1) NIL (0 0 1)))
```

[seq p??]
[exit p??]
[sayBrightly p??]
[concat p1107]
[pname p1106]
[lassoc p??]
[sayBrightlyNT p??]
[pp p??]
[$InteractiveFrame p34]

— **defun reportUndo** —

```
(defun |reportUndo| (acc)
 (prog (name proplist curproplist prop value)
 (declare (special |$InteractiveFrame|))
  (return
   (seq
    (do ((tmp0 acc (cdr tmp0)) (tmp1 nil))
        ((or (atom tmp0)
             (progn (setq tmp1 (car tmp0)) nil)
             (progn
              (progn
               (setq name (car tmp1))
               (setq proplist (cdr tmp1))
               tmp1)
              nil))
            nil)
     (seq
      (exit
       (progn
        (|sayBrightly|
          (concat '|Properties of | (pname name) " ::"))
        (setq curproplist (lassoc name (caar |$InteractiveFrame|)))
        (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
            ((or (atom tmp2)
                 (progn (setq tmp3 (car tmp2)) nil)
                 (progn
                  (progn
                   (setq prop (car tmp3))
                   (setq value (cdr tmp3))
                   tmp3)
                  nil))
                nil)
         (seq
          (exit
           (progn
            (|sayBrightlyNT|
             (cons "  " (cons prop (cons " was: " nil))))
            (|pp| value)
            (|sayBrightlyNT|
             (cons "  " (cons prop (cons " is:  " nil))))
            (|pp| (lassoc prop curproplist))))))))))))))
```

———-

6.3.7 Undo previous n commands

[userError p??]
[concat p1107]
[$IOindex p34]

— **defun undoCount** —

```
(defun |undoCount| (n)
 "Undo previous n commands"
 (prog (m)
 (declare (special |$IOindex|))
  (return
   (progn
    (setq m
     (cond
       ((>= n 0) (- (- |$IOindex| n) 1))
       (t (- n))))
    (cond
     ((>= m |$IOindex|)
       (|userError|
        (concat "Magnitude of undo argument must be less than step number ("
           (princ-to-string |$IOindex|) ").")))
     (t m))))))
```

———-

Chapter 7

Tracing

7.1 The help text

— trace.help —

```
=======================================================================
A.26.  )trace
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )trace
  - )trace )off

  - )trace function [options]
  - )trace constructor [options]
  - )trace domainOrPackage [options]

where options can be one or more of

  - )after S-expression
  - )before S-expression
  - )break after
  - )break before
  - )cond S-expression
  - )count
  - )count n
  - )depth n
  - )local op1 [... opN]
  - )nonquietly
  - )nt
  - )off
  - )only listOfDataToDisplay
  - )ops
```

```
  - )ops op1 [... opN ]
  - )restore
  - )stats
  - )stats reset
  - )timer
  - )varbreak
  - )varbreak var1 [... varN ]
  - )vars
  - )vars var1 [... varN ]
  - )within executingFunction

Command Description:

This command is used to trace the execution of functions that make up the
AXIOM system, functions defined by users, and functions from the system
library. Almost all options are available for each type of function but
exceptions will be noted below.

To list all functions, constructors, domains and packages that are traced,
simply issue

)trace

To untrace everything that is traced, issue

)trace )off

When a function is traced, the default system action is to display the
arguments to the function and the return value when the function is exited.
Note that if a function is left via an action such as a THROW, no return
value will be displayed. Also, optimization of tail recursion may decrease
the number of times a function is actually invoked and so may cause less
trace information to be displayed. Other information can be displayed or
collected when a function is traced and this is controlled by the various
options. Most options will be of interest only to AXIOM system developers. If
a domain or package is traced, the default action is to trace all functions
exported.

Individual interpreter, lisp or boot functions can be traced by listing their
names after )trace. Any options that are present must follow the functions to
be traced.

)trace f

traces the function f. To untrace f, issue

)trace f )off

Note that if a function name contains a special character, it will be
necessary to escape the character with an underscore

)trace _/D_,1

To trace all domains or packages that are or will be created from a
```

```
particular constructor, give the constructor name or abbreviation after
)trace.

)trace MATRIX
)trace List Integer

The first command traces all domains currently instantiated with Matrix. If
additional domains are instantiated with this constructor (for example, if
you have used Matrix(Integer) and Matrix(Float)), they will be automatically
traced. The second command traces List(Integer). It is possible to trace
individual functions in a domain or package. See the )ops option below.

The following are the general options for the )trace command.

  )break after
    causes a Lisp break loop to be entered after exiting the traced function.

  )break before
    causes a Lisp break loop to be entered before entering the traced
    function.

  )break
    is the same as )break before.

  )count
    causes the system to keep a count of the number of times the traced
    function is entered. The total can be displayed with )trace )stats and
    cleared with )trace )stats reset.

  )count n
    causes information about the traced function to be displayed for the
    first n executions. After the nth execution, the function is untraced.

  )depth n
    causes trace information to be shown for only n levels of recursion of
    the traced function. The command

    )trace fib )depth 10

    will cause the display of only 10 levels of trace information for the
    recursive execution of a user function fib.

  )math
    causes the function arguments and return value to be displayed in the
    AXIOM monospace two-dimensional math format.

  )nonquietly
    causes the display of additional messages when a function is traced.

  )nt
    This suppresses all normal trace information. This option is useful if
    the )count or )timer options are used and you are interested in the
    statistics but not the function calling information.
```

```
  )off
    causes untracing of all or specific functions. Without an argument, all
    functions, constructors, domains and packages are untraced. Otherwise,
    the given functions and other objects are untraced. To immediately
    retrace the untraced functions, issue )trace )restore.

  )only listOfDataToDisplay
    causes only specific trace information to be shown. The items are listed
    by using the following abbreviations:

    a        display all arguments
    v        display return value
    1        display first argument
    2        display second argument
    15       display the 15th argument, and so on

  )restore
    causes the last untraced functions to be retraced. If additional options
    are present, they are added to those previously in effect.

  )stats
    causes the display of statistics collected by the use of the )count and
    )timer options.

  )stats reset
    resets to 0 the statistics collected by the use of the )count and )timer
    options.

  )timer
    causes the system to keep a count of execution times for the traced
    function. The total can be displayed with )trace )stats and cleared with
    )trace )stats reset.

  )varbreak var1 [... varN]
    causes a Lisp break loop to be entered after the assignment to any of the
    listed variables in the traced function.

  )vars
    causes the display of the value of any variable after it is assigned in
    the traced function. Note that library code must have been compiled (see
    description of command )compile ) using the )vartrace option in order to
    support this option.

  )vars var1 [... varN]
    causes the display of the value of any of the specified variables after
    they are assigned in the traced function. Note that library code must
    have been compiled (see description of command )compile ) using the
    )vartrace option in order to support this option.

  )within executingFunction
    causes the display of trace information only if the traced function is
    called when the given executingFunction is running.

The following are the options for tracing constructors, domains and packages.
```

```
  )local [op1 [... opN] ]
    causes local functions of the constructor to be traced. Note that to
    untrace an individual local function, you must use the fully qualified
    internal name, using the escape character _ before the semicolon.

    )trace FRAC )local
    )trace FRAC_;cancelGcd )off

  )ops op1 [... opN]
    By default, all operations from a domain or package are traced when the
    domain or package is traced. This option allows you to specify that only
    particular operations should be traced. The command

    )trace Integer )ops min max _+ _-

    traces four operations from the domain Integer. Since + and - are special
    characters, it is necessary to escape them with an underscore.

Also See:
o )lisp
o )ltrace
```

———-

[1]

7.2 The trace global variables

7.2.1 defvar $breakCondition

— initvars —

```
(defvar |$breakCondition| nil)
```

———-

7.2.2 defvar $constructors

— initvars —

```
(defvar |$constructors| nil)
```

———-

[1] "lisp" (26.27 p 831) "ltrace" (26.28.2 p 832)

7.2.3 defvar $constructors

— **initvars** —

```
(defvar |$constructors| nil)
```

———-

7.2.4 defvar $countList

— **initvars** —

```
(defvar |$countList| nil "A list of traced functions to count")
```

———-

7.2.5 defvar $depthAlist

— **initvars** —

```
(defvar |$depthAlist| nil)
```

———-

7.2.6 defvar $domains

— **initvars** —

```
(defvar |$domains| nil)
```

———-

7.2.7 defvar $domainTraceNameAssoc

This is an alist whose car is the domain and whose cdr is a gensym.

— **initvars** —

```
(defvar |$domainTraceNameAssoc| nil)
```

———-

7.2.8 defvar $doNotAddEmptyModeIfTrue

— **initvars** —

```
(defvar |$doNotAddEmptyModeIfTrue| nil)
```

———-

7.2.9 defvar $embeddedFunctions

— **initvars** —

```
(defvar |$embeddedFunctions| nil)
```

———-

7.2.10 defvar $fromSpadTrace

— **initvars** —

```
(defvar |$fromSpadTrace| nil)
```

———-

7.2.11 defvar $lastUntraced

— **initvars** —

```
(defvar |$lastUntraced| nil)
```

———-

7.2.12 defvar $letAssoc

— **initvars** —

```
(defvar |$letAssoc| nil)
```

———-

7.2.13 defvar $mapSubNameAlist

— **initvars** —

```
(defvar |$mapSubNameAlist| nil)
```

———-

7.2.14 defvar $mathTrace

— **initvars** —

```
(defvar |$mathTrace| nil)
```

———-

7.2.15 defvar $mathTraceList

— **initvars** —

```
(defvar |$mathTraceList| nil "A list of functions with math trace output")
```

———-

7.2.16 defvar $monitorArgs

— **initvars** —

```
(defvar |$monitorArgs| nil)
```

———-

7.2.17 defvar $monitorCaller

— **initvars** —

```
(defvar |$monitorCaller| nil)
```

———-

7.2.18 defvar $monitorDepth

— **initvars** —

```
(defvar |$monitorDepth| 0)
```

———-

7.2.19 defvar $monitorFunDepth

— **initvars** —

```
(defvar |$monitorFunDepth| nil)
```

———-

7.2.20 defvar $monitorName

— **initvars** —

```
(defvar |$monitorName| nil)
```

———-

7.2.21 defvar $monitorPretty

— **initvars** —

```
(defvar |$monitorPretty| nil)
```

———-

7.2.22 defvar $monitorValue

— **initvars** —

```
(defvar |$monitorValue| nil)
```

———-

7.2.23 defvar $optionAlist

— **initvars** —

```
(defvar |$optionAlist| nil)
```

———-

7.2.24 defvar $options

— **initvars** —

```
(defvar |$options| nil)
```

———-

7.2.25 defvar $OutputForm

— **initvars** —

```
(defvar |$OutputForm| nil)
```

———-

7.2.26 defvar $packages

— **initvars** —

```
(defvar |$packages| nil)
```

———-

7.2.27 defvar $QuickLet

— **initvars** —

```
(defvar |$QuickLet| nil)
```

———-

7.2.28 defvar $reportSpadtrace

This reports the traced functions

— **initvars** —

```
(defvar |$reportSpadtrace| nil)
```

———-

7.2.29 defvar $spaceList

— **initvars** —

```
(defvar |$spaceList| nil "A list of traced functions to calculate space used")
```

———-

7.2.30 defvar $streamCount

— **initvars** —

```
(defvar |$streamCount| t)
```

———-

7.2.31 defvar $timerList

— **initvars** —

```
(defvar |$timerList| nil "A list of traced functions to time")
```

———-

7.2.32 defvar $tracedMapSignatures

— **initvars** —

```
(defvar |$tracedMapSignatures| nil)
```

———-

7.2.33 defvar $traceDomains

— **initvars** —

```
(defvar |$traceDomains| t)
```

———-

7.2.34 defvar $traceErrorStack

— **initvars** —

```
(defvar |$traceErrorStack| t)
```

———-

7.2.35 defvar $TraceFlag

— **initvars** —

```
(defvar |$TraceFlag| t)
```

———-

7.2.36 defvar $traceletflag

— **initvars** —

```
(defvar $traceletflag nil)
```

———-

7.2.37 defvar $traceletFunctions

— **initvars** —

```
(defvar |$traceletFunctions| nil)
```

———-

7.2.38 defvar $traceNames

— **initvars** —

```
(defvar |$traceNames| nil "The names of traced domains and packages")
```

———-

7.2.39 defvar $traceNoisely

This decides when to give trace and untrace messages.

— **initvars** —

```
(defvar |$traceNoisely| nil)
```

———-

7.2.40 defvar $traceOptionList

— **initvars** —

```
(defvar |$traceOptionList|
 '(|after| |before| |break| |cond| |count| |depth| |local| |mathprint|
   |nonquietly| |nt| |of| |only| |ops| |restore| |timer| |varbreak|
   |vars| |within|))
```

———-

7.2.41 defvar $tracedSpadModemap

— **initvars** —

```
(defvar |$tracedSpadModemap| nil)
```

————-

7.2.42 defvar $traceSize

This is the size limit of output during tracing. See 7.4.110

— **initvars** —

```
(defvar |$traceSize| nil "size limit of output during tracing")
```

————-

7.2.43 defvar $traceStream

— **initvars** —

```
(defvar |$traceStream| *standard-output*)
```

————-

7.3 The trace initialization

— **postvars** —

```
(eval-when (eval load)
 (put '|coerce| '/TRANSFORM '(& & *))
 (put '|comp| '/TRANSFORM '(& * * &))
 (put '|compIf| '/TRANSFORM '(& * * &))
 ; by having no transform for the 3rd argument, it is simply not printed
 (put '|compFormWithModemap| '/TRANSFORM '(& * * & &)))
```

————-

7.4 The trace functions

7.4.1 defun The Top Level)trace Command Handler

[traceSpad2Cmd p68]

— **defun trace** —

```
(defun |trace| (l)
 (traceSpad2Cmd l))
```

———-

7.4.2 defun traceSpad2Cmd

[getMapSubNames p115]
[trace1 p68]
[augmentTraceNames p68]
[traceReply p82]
[$mapSubNameAlist p61]
[$options p63]

— **defun traceSpad2Cmd** —

```
(defun traceSpad2Cmd (l)
 (declare (special |$mapSubNameAlist| |$options|))
  (when (eq (first l) '|Tuple|) (setq l (second l)))
  (setq |$mapSubNameAlist| (getMapSubNames l))
  (trace1 (augmentTraceNames l) |$options|)
  (traceReply))
```

———-

7.4.3 defun augmentTraceNames

If we have values on the **localModemap** property of `$InteractiveFrame` we collect the names, otherwise we simply add the arguments to the result. [get p**??**]
[$InteractiveFrame p34]

— **defun augmentTraceNames** —

```
(defun augmentTraceNames (arg)
 (let (mml res)
 (declare (special |$InteractiveFrame|))
 (loop for traceName in arg do
  (if (setq mml (|get| tracename '|localModemap| |$InteractiveFrame|))
   (setq res (append (loop for mm in mml collect (cadr mm))) res)
   (setq res (cons tracename res)))
 res))
```

———-

7.4.4 defun trace1

The trace1 function handles the options **off**, **stats**, **restore**, and the **help** options. [hasOption p704]
[throwKeyedMsg p**??**]
[unabbrev p**??**]

[isFunctor p??]
[getTraceOption p104]
[untraceDomainLocalOps p120]
[qslessp p1174]
[poundsign p??]
[untrace p108]
[ptimers p86]
[say p??]
[pcounters p87]
[selectOptionLC p728]
[resetSpacers p85]
[resetTimers p85]
[resetCounters p85]
[qcar p??]
[qcdr p??]
[vecp p??]
[sayKeyedMsg p39]
[devaluate p??]
[lassoc p??]
[trace1 p68]
[delete p??]
[?t p129]
[seq p??]
[exit p??]
[transTraceItem p111]
[addassoc p??]
[getTraceOptions p102]
[/trace,0 p??]
[saveMapSig p102]
[$traceNoisely p66]
[$options p63]
[$lastUntraced p61]
[$optionAlist p63]

— defun trace1 —

```
(defun trace1 (l options)
 (prog (|$traceNoisely| varList y domainList optionList traceList
        opList newOptions oldL a opt constructor lops ops)
  (declare (special |$traceNoisely| |$options|))
  (setq |$traceNoisely| nil)
  (when (|hasOption| options '|nonquietly|) (setq |$traceNoisely| t))
  (cond
    ((|hasOption| options '|off|)
     (cond
      ((or (setq ops (|hasOption| options '|ops|))
           (setq lops (|hasOption| options '|local|)))
        (cond
         ((null l)
            (|throwKeyedMsg|
             (format nil
```

```
            "If you use the )off option for )trace and you also use the ~
             )local or )ops option, you must specify the name of a ~
             constructor.  You have not done so.")
            nil))
       (t
         (setq constructor
           (|unabbrev|
             (cond
              ((atom l) l)
              ((null (cdr l))
                (cond
                 ((atom (car l)) (car l))
                 (t (car (car l)))))
              (t nil))))
         (cond
          ((null (|isFunctor| constructor))
              (|throwKeyedMsg|
               (format nil
                "If you use the )off option for )trace and you also use ~
                 the )local or )ops option, you must specify the name ~
                 of a constructor. What you gave after )trace is not a ~
                 valid constructor name.")
               nil))
          (t
            (when ops (setq ops (|getTraceOption| ops)) nil)
            (when lops
             (setq lops (cdr (|getTraceOption| lops)))
             (|untraceDomainLocalOps| constructor lops)))))))
  ((and (< 1 (length options))
        (null (|hasOption| options '|nonquietly|)))
      (|throwKeyedMsg|
       (format nil
        "If you use the )off option for )trace then the only other ~
         options you can use are )nonquietly, )ops and )local.")
       nil))
  (t
    (|untrace| l)
    (|clearConstructorCaches|))))
((|hasOption| options '|stats|)
  (cond
    ((< 1 (length options))
      (|throwKeyedMsg| "%1 can have no other options."
        (cons ")trace ... )stats" nil)))
    (t
       (setq opt (cdar options))
       (cond
        ((null opt)
          (format t "~v,,,'-:@<~a~>~%" (- $linelength 2)
            " Traced function execution times ")
          (|ptimers|)
          (say " ")
          (format t "~v,,,'-:@<~a~>~%" (- $linelength 2)
            " Traced function execution counts ")
          (|pcounters|))
```

```
          (t
            (|selectOptionLC| (car opt) '(|reset|) '|optionError|)
            (|resetSpacers|)
            (|resetTimers|)
            (|resetCounters|)
            (|throwKeyedMsg|
             (format nil
              "Trace facility timers, space counts and execution counts ~
               have been reset.")
              nil))))))
  ((setq a (|hasOption| options '|restore|))
    (cond
     ((null (setq oldL |$lastUntraced|)) nil)
     (t
       (setq newOptions (|delete| a options))
       (cond
         ((null l) (trace1 oldL options))
         (t
          (loop for x in l do
           (cond
            ((simple-vector-p (car x))
              (|sayKeyedMsg| "Please retrace the domain %1."
                (list (|devaluate| (car x)))))
            (t
              (setq options
                (append newOptions (lassoc x |$optionAlist|)))
              (trace1 (list x) options)))))))))
  ((null l) nil)
  ((and (consp l) (eq (cdr l) nil) (eq (car l) '?)) (|?t|))
  (t
    (setq traceList
     (or
      (loop for x in l collect (|transTraceItem| x))
      (return nil)))
    (loop for x in traceList do
      (setq |$optionAlist| (addassoc x options |$optionAlist|)))
    (setq optionList (getTraceOptions options))
    (cond
     ((setq domainList (lassoc '|of| optionList))
       (cond
        ((lassoc '|ops| optionList)
            (|throwKeyedMsg|
             ")ops and )of cannot both be options to )trace"
             nil))
        (t
         (setq opList (when traceList (list (cons '|ops| traceList))))
         (setq varList
           (when
             (setq y (lassoc '|vars| optionList)) (list (cons '|vars| y))))
         (setq optionList (append opList varList))
         (setq traceList domainList)))))
    (loop for funName in traceList do
      (trace2 funName nil optionList))
     (|saveMapSig| traceList))))
```

———-

7.4.5 defun trace2

[trace3 p72]

— **defun trace2** —

```
(defun trace2 (fn modemap options)
  (trace3 fn modemap options nil))
```

———-

7.4.6 defun trace3

The **trace3** examines the options to the top level trace command and builds the lambda expression in the variable **newdef**

```
(lambda (&rest g6) (monitorx g6 fn ll))
```

The **fn** variable is an uninterned gensym name that is unique. The **ll** variable is a list of

- tracename – The spad internal name for a function from a domain which can be found in the compiler output files. e.g. Polynomial.univariate.21
- (when g4 'macro) – most likely nil
- tracecode – a string controlling what tracing will occur where tracecode means

```
; 0:        Caller (0,1)           print caller if 1
; 1:        Value (0,1)            print value if 1
; 2...:     Arguments (0,...,9)    stop if 0; print ith if i; all if 9
```

- countnam –
- timernam –
- before – An expression to execute before executing the function. It defaults to

```
(eq nil nil)
```

- after –
- condition –
- break –
- modemap – The modemap for the function. For Polynomial.variables.15 it shows up as

```
((List (Symbol)) (Polynomial (Integer)))
```

- (list 'quote t) –

[untrace2 p??]
[options2uc p??]
[isFunctor p??]
[traceDomainConstructor p120]
[rassoc p??]

[isSubForRedundantMapName p92]
[rassocSub p91]
[spadThrowBrightly p??]
[getTraceOption p104]
[tracelet p136]
[breaklet p136]
[isUncompiledMap p91]
[sayBrightly p??]
[isInterpOnlyMap p92]
[isDomainOrPackage p94]
[spadTrace p116]
[strconc p??]
[pname p1106]
[trace2 p72]
[adjoinEqual p??]
[embededFunction p77]
[embed2 p77]
[$timerList p65]
[$countList p60]
[$fromSpadTrace p61]
[$traceNoisely p66]
[$mathTraceList p62]
[$mapSubNameAlist p61]
[$traceNames p66]
[$traceDomains p65]

— **defun trace3** —

```
(defun trace3 (fn modemap options binDef)
 (labels (
  (getTraceOption (traceopts opt)
   (loop for x in traceopts do (when (eq (car x) opt) (return x)))))
 (let (mathTrace vars break varbreak fnval u tracename letfuncode before
       after caller fromCondition condition withinCondition g countnam
       countCondition timernam depthCondition onlys f a v c nl buf
       tracecode g4 ll newDef oldDef)
    (declare (special |$traceNames| |$traceDomains| |$mapSubNameAlist|
              |$mathTraceList| |$traceNoisely| |$fromSpadTrace|
              |$countList| |$timerList|))
    (when (member fn |$traceNames| :test #'eq) (|untrace2| fn nil))
    (setq options (options2uc options))
    (cond
     ((and |$traceDomains| (|isFunctor| fn) (atom fn))
      (|traceDomainConstructor| fn options))
     (t
       (setq mathTrace (getTraceOption options 'mathprint))
       (when
        (and mathTrace
             (null (eql (elt (symbol-name fn) 0) #\$))
             (null (gensymp fn)))
         (if (rassoc fn |$mapSubNameAlist|)
           (setq |$mathTraceList| (cons fn |$mathTraceList|)))
```

```
       (|spadThrowBrightly|
         (format nil "mathprint not available for ~A" fn)))
(setq vars (getTraceOption options 'vars))
(when vars
  (setq vars (unless (cdr vars) 'all (cdr vars)))
  (|tracelet| fn binDef vars))
(setq break (getTraceOption options 'break))
(setq varbreak (getTraceOption options 'varbreak))
(when varbreak
  (setq vars (if (null (cdr varbreak)) 'all (cdr varbreak)))
  (|breaklet| fn binDef vars))
(cond
  ((and (null bindef) (symbolp fn) (null (boundp fn)) (null (fboundp fn)))
    (cond
      ((|isUncompiledMap| fn)
       (|sayBrightly|
        (format nil
    "~A must be compiled before it may be traced -- invoke ~A to compile"
         fn fn)))
      ((|isInterpOnlyMap| fn)
       (|sayBrightly|
        (format nil
           "~A cannot be traced because it is an interpret-only function"
           fn)))
      (t
        (|sayBrightly| (format nil "~A is not a function" fn)))))
  ((and (null bindef) (symbolp fn) (boundp fn)
        (|isDomainOrPackage| (setq fnval (eval fn))))
    (|spadTrace| fnval options))
  (t
    (when (setq u (getTraceOption options 'mask=))
      (makeprop fn '/transform (elt u 1)))
    (setq |$traceNames|
      (if (and options (getTraceOption options 'alias))
        |$traceNames|
        (cons fn |$traceNames|)))
    (setq tracename
      (cond
        ((setq u (getTraceOption options 'alias))
          (princ-to-string (elt u 1)))
        (t
          (when (and |$traceNoisely| (null vars)
                     (null (|isSubForRedundantMapName| fn)))
             (|sayBrightly|
              (list (|rassocSub| fn |$mapSubNameAlist|) "traced")))
          (princ-to-string fn))))
    (cond
      (|$fromSpadTrace|
       (when mathTrace (push (intern tracename) |$mathTraceList|))
       (setq letfuncode (list 'eq nil nil))
       (setq before
        (if (setq u (getTraceOption options 'before))
          (list 'progn (elt u 1) letfuncode)
          letfuncode)))
```

```
  (t
   (setq before
    (when (setq u (getTraceOption options 'before)) (elt u 1)))))
(setq after
  (when (setq u (getTraceOption options 'after)) (elt u 1)))
(setq caller (getTraceOption options 'caller))
(setq fromCondition
  (if (setq u (getTraceOption options 'from))
    (list 'eq '|#9| (list 'quote (elt u 1)))
    t))
(setq condition
  (if (setq u (getTraceOption options 'when))
    (elt u 1)
    t))
(setq withinCondition t)
(when (setq u (getTraceOption options 'within))
  (setq g
   (intern (strconc (symbol-name fn) "/" (symbol-name (elt u 1)))))
  (set g 0)
  (trace2 (elt u 1) nil
    (list (list 'when nil)
          (list 'before (list 'setq g (list '1+ g)))
          (list 'after (list 'setq g (list '1- g))))
    (setq withinCondition (list '> g 0)))
(when (getTraceOption options 'count)
  (setq countnam (intern (strconc tracename '|,COUNT|))))
(setq countCondition
  (cond
    ((setq u (getTraceOption options 'count))
      (setq |$countList| (|adjoinEqual| tracename |$countList|))
      (cond
        ((and (cdr u) (integerp (elt u 1)))
          (list 'cond (list (list '<= countnam (elt u 1)) t)
                (list t (list '|untrace2| (mkq fn) nil) nil)))
        (t t)))
    (t t)))
(when (getTraceOption options 'timer)
  (setq timernam (intern (strconc tracename ",TIMER")))
  (setq |$timerList| (|adjoinEqual| tracename |$timerList|)))
(setq depthCondition
 (if (setq u (getTraceOption options 'depth))
  (if (and (cdr u) (integerp (elt u 1)))
    (list '<= '|$monitorFunDepth| (elt u 1))
    (traceOptionError 'depth))
  t))
(setq condition
 (mkpf
   (list condition withinCondition fromCondition
         countCondition depthCondition)
   'and))
(setq onlys (getTraceOption options 'only))
(setq tracecode
 (cond
  ((getTraceOption options 'nt) "000")
```

```
  (t
    (setq onlys
      (mapcar #'(lambda (x) (if (integerp x) x (upcase x))) onlys))
    (setq f (or   (member 'f onlys) (member 'full onlys)))
    (setq a (or f (member 'a onlys) (member 'args onlys)))
    (setq v (or f (member 'v onlys) (member 'value onlys)))
    (setq c (or f (member 'c onlys) (member 'caller onlys)))
    (setq nl
      (if a
        (list #\9)
        (loop for x in onlys
              when (and (integerp x) (< 0 x) (< x 9))
              collect (char (princ-to-string x) 0))))
    (cond
     ((null (or a v c nl))
       (if caller "119" "019"))
     (t
      (setq nl (append nl (list #\0)))
      (setq buf (|make_spaces| (cond (a 3) (t (+ 2 (length nl))))))
      (setf (elt buf 0) (if (or c caller) #\1 #\0))
      (setf (elt buf 1) (if v #\1 #\0))
      (cond
        (a
         (setf (elt buf 2) #\9)
         buf)
        (t
         (loop for x in nl for i from 2 do (setf (elt buf i) x))
         buf)))))))
(setq g4 (macro-function fn))
(when countnam (set countnam 0))
(when timernam (set timernam 0))
(setq ll
 (list 'quote
  (list (intern tracename) (when g4 'macro)
        tracecode countnam timernam before after
        condition break modemap (list 'quote t))))
(setq newDef
 (list (if g4 'mlambda 'lambda) (list '&rest 'g6)
       (list 'monitorX 'g6 fn ll)))
(cond
 (binDef (embededFunction fn newDef binDef))
 (t
  (setq oldDef (symbol-function fn))
  (setq newDef (embededFunction fn new_def old_def))
  (embed2 fn new_def old_def)
  fn)))))))))
```

———-

7.4.7 defun embededFunction

[flatBvList p77]

— **defun embededFunction** —

```
(defun embededFunction (name newDef oldDef)
 (let (body op bv)
  (setq newDef
   (cond
    ((null (consp newDef)) newDef)
    (t
      (setq body (cddr newDef))
      (setq op (car newDef))
      (if (and body (setq bv (cadr newDef))
                    (or (eq op 'lambda) (eq op 'mlambda)))
         (if (null (member name (flatBvList bv)))
          (list op bv
            (list
              (cons 'lambda (cons (list name) body))
              (list 'quote oldDef)))
          newDef)
        (break)))))
  (coerce newDef 'function)))
```

———-

7.4.8 defun embed2

embed2 : Symbol,Function,Function → Symbol

— **defun embed2 0** —

```
(defun embed2 (name newDef oldDef)
 (declare (special |$embeddedFunctions|))
  (setf (symbol-function name) newDef)
  (push (list name newDef oldDef) |$embeddedFunctions|)
  name)
```

———-

7.4.9 defun flatBvList

[varp p78]
[refvecp p??]
[flatBvList p77]
[nconc p??]

— **defun flatBvList** —

```
(defun flatBvList (bvlist)
 (let (tmp1)
```

```
  (cond
   ((varp bvlist) (list bvlist))
   ((refvecp bvlist) (break))
   ((null (consp bvlist)) nil)
   ((eq '= (setq tmp1 (car bvlist)))
     (flatBvList (cdr bvlist)))
   ((varp tmp1) (cons tmp1 (flatBvList (cdr bvlist))))
   ((and (null (consp tmp1)) (null (refvecp tmp1)))
     (flatBvList (cdr bvlist)))
   (t
    (nconc (flatBvList tmp1) (flatBvList (cdr bvlist)))))))
```

———-

7.4.10 defun varp

— **defun var** —

```
(defun varp (testItem)
 (cond
  ((identp testItem) testItem)
  ((and (consp testItem)
        (or (eq (car testItem) 'fluid) (eq (car testItem) 'lex))
        (consp (cdr testItem)) (identp (cadr testItem)))
    testItem)
  (t nil)))
```

———-

7.4.11 defun monitorX

[monitorXX p78]
[$monitorDepth p62]
[$depthAlist p60]

— **defun monitorX** —

```
(defun monitorX (args funct opts)
 (declare (special |$monitorDepth| |$depthAlist|))
  (monitorXX args funct opts |$monitorDepth| |$depthAlist|)))
```

———-

7.4.12 defun monitorXX

[stopTimer p??]
[rassocSub p91]
[whocalled p??]
[moan p??]

[monitorEvalTran p81]
[monitorEnter p130]
[monitorEvalBefore p80]
[break p137]
[startTimer p??]
[timerValue p??]
[monitorEvalAfter p81]
[monitorExit p133]
[$monitorArgs p62]
[$monitorValue p63]
[$monitorFunDepth p62]
[$depthAlist p60]
[$monitorCaller p62]
[$breakCondition p59]
[$monitorName p63]
[$monitorDepth p62]
[$tracedSpadModemap p67]
[$mathTrace p62]
[$mapSubNameAlist p61]

— defun monitorXX —

```
(defun monitorXX (|$monitorArgs| funct opts oldDepth oldDepthAlist)
 (declare (special |$monitorArgs|))
 (let (|$monitorValue| |$monitorFunDepth| |$depthAlist| |$monitorCaller|
       |$breakCondition| |$monitorName| |$monitorDepth|
       |$tracedSpadModemap| |$mathTrace| evalTime initTime yes
       notTopLevel a v c name1 breakcondition tracedmodemap break
       condition after before timernam countnam tracecode type name)
    (declare
     (special |$monitorValue| |$monitorFunDepth| |$depthAlist|
      |$monitorCaller| |$breakCondition| |$monitorName| |$monitorDepth|
      |$tracedSpadModemap| |$mathTrace| |$mapSubNameAlist|))
  (|stopTimer|)
  (setq name (elt opts 0))
  (setq type (elt opts 1))
  (setq tracecode (elt opts 2))
  (setq countnam (elt opts 3))
  (setq timernam (elt opts 4))
  (setq before (elt opts 5))
  (setq after (elt opts 6))
  (setq condition (elt opts 7))
  (setq break (elt opts 8))
  (setq tracedmodemap (elt opts 9))
  (setq breakcondition (elt opts 10))
  (setq |$mathTrace| nil)
  (setq |$tracedSpadModemap| tracedmodemap)
  (setq |$monitorDepth| (+ oldDepth 1))
  (setq |$monitorName| (symbol-name name))
  (setq name1 (|rassocSub| name |$mapSubNameAlist|))
  (setq |$breakCondition| breakcondition)
  (setq |$monitorCaller| (|rassocSub| (whocalled 6) |$mapSubNameAlist|))
  ;TRACECODE meaning:
```

```
  ; 0:        Caller (0,1)           print caller if 1
  ; 1:        Value (0,1)            print value if 1
  ; 2...:     Arguments (0,...,9)    stop if 0; print ith if i; all if 9
  (cond
   ((null (stringp tracecode))
     (moan "set tracecode to \\'1911\\' and restart"))
   (t
     (setq c (digit-char-p (elt tracecode 0)))
     (setq v (digit-char-p (elt tracecode 1)))
     (setq a (digit-char-p (elt tracecode 2)))
     (when countnam (set countnam (+ (eval countnam) 1)))
     (setq |$depthAlist| (copy-tree oldDepthAlist))
     (setq notTopLevel (assoc name |$depthAlist|))
     (if (null notTopLevel)
        (setq |$depthAlist| (cons (cons name 1) |$depthAlist|))
        (rplacd notTopLevel (+ (cdr notTopLevel) 1)))
     (setq |$monitorFunDepth| (cdr (assoc name |$depthAlist|)))
     (setq condition (monitorEvalTran condition nil))
     (setq yes (eval condition))
     (when (member name |$mathTraceList|) (setq |$mathTrace| t))
     (when (and yes |$TraceFlag|) (monitorEnter tracecode c type name name1))
     (when before (monitorEvalBefore before))
     (when (member '|before| break)
       (|break| (list "Break on entering" (symbol-name name1) ":")))
     (when timernam (setq initTime (|startTimer|)))
     (setq |$monitorValue|
       (if (eq type 'macro)
        (macroexpand funct |$monitorArgs|)
        (apply funct |$monitorArgs|)))
     (|stopTimer|)
     (setq evalTime nil)
     (when timernam (setq evalTime (- (|timerValue|) initTime)))
     (when (and timernam (null notTopLevel))
       (set timernam (+ (eval timernam) evalTime)))
     (when after (monitorEvalAfter after))
     (when (and yes |$TraceFlag|)
       (monitorExit tracecode name name1 V timernam evalTime))
     (when (member '|after| break)
       (|break| (list "Break on exiting" (symbol-name name1) ":")))
     (|startTimer|)
     |$monitorValue|))))
```

———-

7.4.13 defun monitorEvalBefore

[monitorEvalTran p81]

— **defun monitorEvalBefore** —

```
(defun monitorEvalBefore (x)
  (eval (monitorEvalTran x nil)))
```

———-

7.4.14 defun monitorEvalAfter

[monitorEvalTran p81]

— **defun monitorEvalAfter** —

```
(defun monitorEvalAfter (x)
  (eval (monitorEvalTran x nil)))
```

———-

7.4.15 defun monitorEvalTran

[hasSharpVar p81]
[monitorEvalTran1 p81]

— **defun monitorEvalTran** —

```
(defun monitorEvalTran (x fg)
 (if (|hasSharpVar| x) (monitorEvalTran1 x fg) x))
```

———-

7.4.16 defun monitorEvalTran1

[monitorEvalTran1 p81]
[isSharpVarWithNum p96]
[monitorGetValue p82]

— **defun monitorEvalTran1** —

```
(defun monitorEvalTran1 (x fg)
 (let (n)
  (cond
   ((setq n (|isSharpVarWithNum| x)) (monitorGetValue n fg))
   ((atom x) x)
   (t
    (cons (monitorEvalTran1 (car x) fg)
          (monitorEvalTran1 (cdr x) fg))))))
```

———-

7.4.17 defun hasSharpVar

[hasSharpVar p81]
[isSharpVar p96]

— **defun hasSharpVar** —

```
(defun |hasSharpVar| (x)
 (cond
  ((and (atom x) (|isSharpVar| x)) t)
  ((atom x) nil)
  (t (or (|hasSharpVar| (car x)) (|hasSharpVar| (cdr x))))))
```

————-

7.4.18 defun monitorGetValue

[mkq p??]
[spadThrowBrightly p??]
[$monitorValue p63]
[$monitorCaller p62]
[$monitorName p63]
[$monitorArgs p62]

— **defun monitorGetValue** —

```
(defun monitorGetValue (n fg)
  (cond
   ((eql n 0)
     (if fg
      (mkq |$monitorValue|)
      (|spadThrowBrightly| "cannot ask for value before execution")))
   ((eql n 9) (mkq |$monitorCaller|))
   ((not (< (size |$monitorArgs|) n)) (mkq (elt |$monitorArgs| (- n 1))))
   (t
    (|spadThrowBrightly|
     (list 'function |$monitorName| "does not have" n "arguments")))))
```

————-

7.4.19 defun traceReply

[sayMessage p??]
[sayBrightly p??]
[qcar p??]
[isDomainOrPackage p94]
[addTraceItem p129]
[isFunctor p??]
[isgenvar p??]
[userError p??]
[seq p??]
[exit p??]
[isSubForRedundantMapName p92]
[rassocSub p91]

[poundsign p??]
[sayMSG p40]
[sayBrightlyLength p??]
[flowSegmentedMsg p??]
[concat p1107]
[prefix2String p??]
[abbreviate p??]
[$domains p60]
[$packages p64]
[$constructors p60]
[$linelength p936]
[$traceNames p66]

— **defun traceReply** —

```
(defun traceReply ()
 (let (|$domains| |$packages| |$constructors| d functionList displayList)
  (declare (special |$domains| |$packages| |$constructors| |$traceNames|
     $linelength))
     (setq |$domains| nil)
     (setq |$packages| nil)
     (setq |$constructors| nil)
     (cond
      ((null |$traceNames|) (|sayMessage| "   Nothing is traced now."))
      (t
        (|sayBrightly| " ")
        (loop for x in |$traceNames| do
         (cond
          ((and (consp x) (|isDomainOrPackage| (car x)))
               (|addTraceItem| (car x)))
          ((atom x)
            (cond
              ((|isFunctor| x) (|addTraceItem| x))
              ((isgenvar x) (|addTraceItem| (eval x)))
              (t (setq functionList (cons x functionList)))))
          (t (|userError| "bad argument to trace"))))))
     (setq functionList
          (prog (t1)
           (setq t1 nil)
           (return
            (do ((t2 functionList (cdr t2)) (x nil))
                ((or (atom t2) (progn (setq x (car t2)) nil)) t1)
              (seq
               (exit
                (cond
                 ((null (|isSubForRedundantMapName| x))
                   (setq t1
                    (append t1
                     (cons (|rassocSub| x |$mapSubNameAlist|)
                           (cons " " nil))))))))))))
     (cond
        (functionList
         (cond
```

```
        ((eql 2 (|#| functionList))
           (|sayMSG| (cons '|   Function traced: | functionList)))
        ((<= (+ 22 (|sayBrightlyLength| functionList)) $linelength)
           (|sayMSG| (cons '|   Functions traced: | functionList)))
        (t
           (|sayBrightly| "   Functions traced:")
           (|sayBrightly|
            (|flowSegmentedMsg| functionList $linelength 6))))))
    (when |$domains|
     (setq displayList
      (|concat| (|prefix2String| (car |$domains|))
       (loop for x in (cdr |$domains|) append
        (|concat| ", " (|prefix2String| x)))))
     (when (atom displayList) (setq displayList (list displayList)))
     (|sayBrightly| "   Domains traced: ")
     (|sayBrightly| (|flowSegmentedMsg| displayList $linelength 6)))
    (when |$packages|
     (setq displayList
      (|concat| (|prefix2String| (car |$packages|))
       (loop for x in (cdr |$packages|) append
        (|concat| ", " (|prefix2String| x)))))
     (when (atom displayList) (setq displayList (list displayList)))
     (|sayBrightly| "   Packages traced: ")
     (|sayBrightly| (|flowSegmentedMsg| displayList $linelength 6)))
    (when |$constructors|
     (setq displayList
      (|concat| (|abbreviate| (car |$constructors|))
       (loop for x in (cdr |$constructors|) append
        (|concat| ", " (|abbreviate| x)))))
     (when (atom displayList) (setq displayList (list displayList)))
     (|sayBrightly| "   Parameterized constructors traced:")
     (|sayBrightly| (|flowSegmentedMsg| displayList $linelength 6)))))
```

———-

7.4.20 defun /options

[/options p84]
[isFunctor p??]

— **defun /options** —

```
(defun /options (x)
 (cond
  ((atom x) nil)
  ((or (atom (car x)) (|isFunctor| (caar x))) (/options (cdr x)))
  (x)))
```

———-

7.4.21 defun Truncate list L at the point marked by TL.

Truncate list L at the point marked by TL.

— **defun trunclist** —

```
(defun trunclist (l tl)
 (labels (
  (trunclist-1 (l tl)
   (cond
    ((atom l) l)
    ((eql (cdr l) tl) (rplacd l nil))
    ((trunclist-1 (cdr l) tl)))))
 (let ((u l))
  (trunclist-1 l tl)
  u)))
```

————-

7.4.22 defun resetTimers

[concat p1107]
[$timerList p65]

— **defun resetTimers** —

```
(defun |resetTimers| ()
 (declare (special |$timerList|))
 (dolist (timer |$timerList|)
  (set (intern (concat timer ",TIMER")) 0)))
```

————-

7.4.23 defun resetSpacers

[concat p1107]
[$spaceList p64]

— **defun resetSpacers** —

```
(defun |resetSpacers| ()
 (declare (special |$spaceList|))
 (dolist (spacer |$spaceList|)
   (set (intern (concat spacer ",SPACE")) 0)))
```

————-

7.4.24 defun resetCounters

[concat p1107]
[$countList p60]

— **defun resetCounters** —

```
(defun |resetCounters| ()
 (declare (special |$countList|))
 (dolist (k |$countList|)
  (set (intern (concat k ",COUNT")) 0)))
```

————-

7.4.25 defun ptimers

[sayBrightly p**??**]
[bright p**??**]
[quotient p**??**]
[concat p1107]
[float p**??**]
[$timerList p65]
[$timerTicksPerSecond p**??**]

— **defun ptimers** —

```
(defun |ptimers| ()
  (declare (special |$timerList| |$timerTicksPerSecond|))
  (if (null |$timerList|)
   (|sayBrightly| "   no functions are timed")
   (dolist (timer |$timerList|)
     (|sayBrightly|
      `("  " ,@(|bright| timer) |:| " "
         ,(quotient (eval (intern (concat timer ",TIMER")))
            (float |$timerTicksPerSecond|)) " sec.")))))
```

————-

7.4.26 defun pspacers

[sayBrightly p**??**]
[bright p**??**]
[concat p1107]
[$spaceList p64]

— **defun pspacers** —

```
(defun |pspacers| ()
 (declare (special |$spaceList|))
 (if (null |$spaceList|)
   (|sayBrightly| "   no functions have space monitored")
   (dolist (spacer |$spaceList|)
     (|sayBrightly|
      `("  " ,@(|bright| spacer) |: |
        ,(eval (intern (concat spacer ",SPACE"))) " bytes")))))
```

———-

7.4.27 defun pcounters

[sayBrightly p??]
[bright p??]
[concat p1107]
[$countList p60]

— 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")))))
```

———-

7.4.28 defun transOnlyOption

[transOnlyOption p87]
[upcase p1140]
[stackTraceOptionError p88]
[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
          "%1 The )trace option )only does not permit %2 as a legal option."
          (list (list n))))
        (|transOnlyOption| y))))))
```

———-

7.4.29 defun stackTraceOptionError

[$traceErrorStack p65]

— defun stackTraceOptionError —

```
(defun |stackTraceOptionError| (x)
 (declare (special |$traceErrorStack|))
 (push x |$traceErrorStack|)
 nil)
```

———-

7.4.30 defun removeOption

removeOption : **Option** → **List Option**

— defun removeOption 0 —

```
(defun removeOption (op options)
 (loop for optentry in options
  when (not (equal (car optentry) op))
  collect optentry))
```

———-

7.4.31 defun domainToGenvar

For a call to trace, such as

```
)trace POLY(INT)
```

This would be called with

```
(POLY INT)
```

which gets unabbreviated to the full form of

```
(|Polynomail| (|Integer|))
```

Since **Polynomial** is a domain we call **genDomainTraceName** to add it to the global alist `$domainTraceNameAssoc`, which returns a unique new genvar, say `$1`. We then set the value of `$1` to the domain (a vector), returning it as the result. [unabbrevAndLoad p**??**]
[getdatabase p1070]
[opOf p**??**]
[genDomainTraceName p107]
[evalDomain p993]
[$doNotAddEmptyModeIfTrue p60]

— 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)))
```

———-

7.4.32 defun subTypes

[lassoc p??]
[seq p??]
[exit p??]
[subTypes p89]

— **defun subTypes** —

```
(defun |subTypes| (|mm| |sublist|)
 (prog (s)
  (return
   (seq
    (cond
     ((atom |mm|)
       (cond ((setq s (lassoc |mm| |sublist|)) s) (t |mm|)))
     (t
      (prog (t0)
       (setq t0 nil)
       (return
        (do ((t1 |mm| (cdr t1)) (|m| nil))
            ((or (atom t1) (progn (setq |m| (car t1)) nil)) (nreverse0 t0))
         (seq
          (exit
           (setq t0 (cons (|subTypes| |m| |sublist|) t0)))))))))))))
```

———-

7.4.33 defun isListOfIdentifiers

[seq p??]
[exit p??]
[identp p1107]

— **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))))))))))))
```

———-

7.4.34 defun isListOfIdentifiersOrStrings

[seq p??]
[exit p??]
[identp p1107]

— **defun isListOfIdentifiersOrStrings** —

```
(defun |isListOfIdentifiersOrStrings| (arg)
 (prog ()
  (return
   (seq
    (prog (t0)
     (setq t0 t)
     (return
      (do ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
          ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
       (seq
        (exit
         (setq t0 (and t0 (or (identp x) (stringp x))))))))))))))
```

———-

7.4.35 defun getPreviousMapSubNames

[get p??]
[exit p??]
[seq p??]
[$InteractiveFrame p34]

— **defun getPreviousMapSubNames** —

```
(defun |getPreviousMapSubNames| (Names)
 (let (lmm subs)
 (declare (special |$InteractiveFrame|))
  (loop for mapName in (assocleft (caar |$InteractiveFrame|)) do
   (when (setq lmm (|get| mapname '|localModemap| |$InteractiveFrame|))
    (when (member (cadar lmm) Names)
     (loop for mm in lmm do
      (setq subs (cons (cons mapname (cadr mm)) subs))))))
  subs))
```

———-

7.4.36 defun lassocSub

[lassq p??]

— **defun lassocSub** —

```
(defun |lassocSub| (x subs)
 (let (y)
  (if (setq y (lassq x subs))
    y
    x)))
```

———-

7.4.37 defun rassocSub

[rassoc p??]

— **defun rassocSub** —

```
(defun |rassocSub| (x subs)
 (let (y)
   (if (setq y (|rassoc| x subs))
    y
    x)))
```

———-

7.4.38 defun isUncompiledMap

[get p??]
[$InteractiveFrame p34]

— **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|))))))
```

———-

7.4.39 defun isInterpOnlyMap

[get p??]
[$InteractiveFrame p34]

— defun isInterpOnlyMap —

```
(defun |isInterpOnlyMap| (map)
 (let (x)
 (declare (special |$InteractiveFrame|))
  (when (setq x (|get| map '|localModemap| |$InteractiveFrame|))
    (eq (caaar x) '|interpOnly|))))
```

———-

7.4.40 defun isSubForRedundantMapName

[rassocSub p91]
[member p1108]
[assocleft p??]
[$mapSubNameAlist p61]

— 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| :test #'equalp))
      (member mapname (cdr (assocleft tail)))))))
```

———-

7.4.41 defun untraceMapSubNames

[assocright p??]
[/untrace,2 p??]
[setdifference p??]
[getPreviousMapSubNames p90]
[$mapSubNameAlist p61]
[$lastUntraced p61]
[$traceNames p66]

— defun untraceMapSubNames —

```
(defun |untraceMapSubNames| (names)
 (let (|$mapSubNameAlist| subs)
 (declare (special |$mapSubNameAlist| |$lastUntraced|))
  (if
   (null (setq |$mapSubNameAlist| (|getPreviousMapSubNames| names)))
```

```
   nil
   (dolist (name (setq subs (assocright |$mapSubNameAlist|)))
     (when (member name |$traceNames|)
       (|/UNTRACE,2| name nil)
       (setq |$lastUntraced| (setdifference |$lastUntraced| subs)))))))
```

———-

7.4.42 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)))))))))))))
```

———-

7.4.43 defmacro funfind

— **defmacro funfind** —

```
(defmacro |funfind| (&whole t0 &rest notused)
 (declare (ignore notused))
 (let (t1 t0)
  (cons '|funfind,LAM| (wrap (cdr t1) '(quote quote)))))
```

———-

7.4.44 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)))))
```

———-

7.4.45 defun flattenOperationAlist

The operation alist for something like POLY(INT) is a list of operations. Each operation, such as '*' is a list of all possible signatures for '*'. For example, the operation alist looks like:

```
   (* (($ (|PositiveInteger|) $) NIL . (T . (ELT)))
      (($ $ $) NIL . (T . (ELT)))
      (($ $ |#1|) NIL . (T . (ELT)))
      (($ |#1| $) NIL . (T . (ELT)))
      (($ (|Fraction| (|Integer|))) NIL
       (|has| |#1| (|Algebra| (|Fraction| (|Integer|)))) . (ELT))
      ....)
```

This function rearranges the operations so that each signature is independent. Thus it becomes

```
   (* (($ (|PositiveInteger|) $) NIL . (T . (ELT))))
   (* (($ $ $) NIL . (T . (ELT))))
   (* (($ $ |#1|) NIL . (T . (ELT))))
   (* (($ |#1| $) NIL . (T . (ELT))))
   (* (($ (|Fraction| (|Integer|))) NIL
       (|has| |#1| (|Algebra| (|Fraction| (|Integer|)))) . (ELT)))
      ....
```

flattenOperationAlist : **OperationAlist** → **OperationAlist**

— **defun flattenOperationAlist 0** —

```
(defun flattenOperationAlist (opAlist)
 (let (res)
  (loop for t1 in opAlist do
   (setq res
    (append res (loop for mm in (cdr t1) collect (cons (car t1) mm)))))
  res))
```

———-

7.4.46 defun letPrint

[lassoc p??]
[isgenvar p??]
[isSharpVarWithNum p96]
[gensymp p??]
[sayBrightlyNT p??]
[bright p??]
[shortenForPrinting p99]
[hasPair p99]
[pname p1106]
[break p137]
[$letAssoc p61]

— 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|))))
```

———-

7.4.47 defun Identifier beginning with a sharpsign-number?

This tests if x is an identifier beginning with # followed by a number.

[isSharpVar p96]
[pname p1106]
[qcsize p??]
[digitp p1106]
[dig2fix p??]

— **defun isSharpVarWithNum** —

```
(defun |isSharpVarWithNum| (x)
 (let (p n d ok c)
  (cond
   ((null (|isSharpVar| x)) nil)
   ((> 2 (setq n (qcsize (setq p (pname x))))) nil)
   (t
    (setq ok t)
    (setq c 0)
    (do ((t1 (1- n)) (i 1 (1+ i)))
        ((or (> i t1) (null ok)) nil)
      (setq d (elt p i))
      (when (setq ok (digitp d))
        (setq c (+ (* 10 c) (dig2fix d)))))
    (when ok c)))))
```

———-

7.4.48 defun Identifier beginning with a sharpsign?

This tests if x is an identifier beginning with #

[identp p1107]

— **defun isSharpVar** —

```
(defun |isSharpVar| (x)
 (and (symbolp x)
      (eql (elt (symbol-name x) 0) #\#)))
;      (integerp (parse-integer (symbol-name x) :start 1))))
```

———-

7.4.49 defun letPrint2

[letPrint2 p96]
[lassoc p??]
[isgenvar p??]
[isSharpVarWithNum p96]
[gensymp p??]
[mathprint p??]

[print p**??**]
[hasPair p99]
[pname p1106]
[break p137]
[bright p**??**]
[$BreakMode p863]
[$letAssoc p61]

— defun letPrint2 —

```
(defun |letPrint2| (x |printform| |currentFunction|)
 (prog (|$BreakMode| |flag| y)
  (declare (special |$BreakMode| |$letAssoc|))
  (return
   (progn
    (setq |$BreakMode| nil)
    (cond
     ((and |$letAssoc|
           (or (setq y (lassoc |currentFunction| |$letAssoc|))
               (setq y (lassoc '|all| |$letAssoc|))))
      (cond
       ((and
          (or (eq y '|all|) (member x y))
          (null (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))
         (setq |$BreakMode| '|letPrint2|)
         (setq |flag| nil)
         (catch '|letPrint2|
          (|mathprint| (cons '= (cons x (cons |printform| nil)))) |flag|)
         (cond
          ((eq |flag| '|letPrint2|) (|print| |printform|))
          (t nil))))
      (cond
       ((and
         (setq y (|hasPair| 'break y))
         (or (eq y '|all|)
             (and
              (member x y)
              (null (member (elt (pname x) 0) '($ |#|)))
              (null (gensymp x)))))
        (|break|
         (append
          (|bright| |currentFunction|)
          (cons "breaks after"
           (append (|bright| x) (cons '|:= | (cons |printform| nil)))))))
       (t nil))))
    x))))
```

———-

7.4.50 defun letPrint3

This is the version for use when we have our hands on a function to convert the data into type "Expression" [letPrint2 p96]
[lassoc p??]
[isgenvar p??]
[isSharpVarWithNum p96]
[gensymp p??]
[mathprint p??]
[spadcall p??]
[print p??]
[hasPair p99]
[pname p1106]
[break p137]
[bright p??]
[$BreakMode p863]
[$letAssoc p61]

— **defun letPrint3** —

```
(defun |letPrint3| (x |xval| |printfn| |currentFunction|)
 (prog (|$BreakMode| |flag| y)
  (declare (special |$BreakMode| |$letAssoc|))
  (return
   (progn
    (setq |$BreakMode| nil)
    (cond
     ((and |$letAssoc|
       (or (setq y (lassoc |currentFunction| |$letAssoc|))
           (setq y (lassoc '|all| |$letAssoc|))))
      (cond
       ((and
          (or (eq y '|all|) (member x y))
          (null (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))
         (setq |$BreakMode| '|letPrint2|)
         (setq |flag| nil)
         (catch '|letPrint2|
          (|mathprint|
           (cons '= (cons x (cons (spadcall |xval| |printfn|) nil))))
          |flag|)
         (cond
          ((eq |flag| '|letPrint2|) (|print| |xval|))
          (t nil))))
      (cond
       ((and
         (setq y (|hasPair| 'break y))
         (or
          (eq y '|all|)
          (and
           (member x y)
           (null (member (elt (pname x) 0) '($ |#|)))
           (null (gensymp x)))))
         (|break|
```

```
          (append
           (|bright| |currentFunction|)
           (cons "breaks after"
            (append (|bright| x) (cons ":= " (cons |xval| nil)))))))
       (t nil))))
   x))))
```

———-

7.4.51 defun hasPair

[qcar p**??**]
[qcdr p**??**]
[hasPair p99]

— **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)))))))
```

———-

7.4.52 defun shortenForPrinting

[isDomainOrPackage p94]
[devaluate p**??**]

— **defun shortenForPrinting** —

```
(defun |shortenForPrinting| (|val|)
 (if (|isDomainOrPackage| |val|)
  (|devaluate| |val|)
  |val|))
```

———-

7.4.53 defun getOption

[assoc p??]

— defun getOption —

```
(defun getOption (opt l)
 (let (y)
  (when (setq y (|assoc| opt l))  (cdr y))))
```

———-

7.4.54 defun orderBySlotNumber

[seq p??]
[assocright p??]
[orderList p??]
[exit p??]

— defun orderBySlotNumber —

```
(defun |orderBySlotNumber| (arg)
 (prog (n)
  (return
   (seq
    (assocright
     (|orderList|
      (prog (t0)
       (setq t0 nil)
       (return
        (do ((t1 arg (cdr t1)) (x nil))
            ((or (atom t1)
                 (progn (setq x (car t1)) nil)
                 (progn (progn (setq n (caddr x)) x) nil))
                (nreverse0 t0))
         (seq
          (exit
           (setq t0 (cons (cons n x) t0)))))))))))))
```

———-

7.4.55 defun spadReply

Collect all of the print names of the `$traceNames` [isDomainOrPackage p94]
[$traceNames p66]

— defun spadReply —

```
(defun |spadReply| ()
 (declare (special |$traceNames|))
 (loop for x in |$traceNames| collect
  (if (and (consp x) (|isDomainOrPackage| (car x)))
```

```
   (|devaluate| (car x))
   x)))
```

———-

7.4.56 defun remover

[remover p101]

— **defun remover** —

```
(defun remover (lst item)
 (cond
  ((null (consp lst)) (cond ((equal lst item) nil) (t lst)))
  ((equal (car lst) item) (cdr lst))
  (t
   (rplnode lst (remover (car lst) item) (remover (cdr lst) item))
   (rplaca lst (remover (car lst) item))
   (rplacd lst (remover (cdr lst) item))
   lst)))
```

———-

7.4.57 defun stupidIsSpadFunction

[strpos p1106]
[pname p1106]

— **defun stupidIsSpadFunction** —

```
(defun |stupidIsSpadFunction| (fn)
 (strpos ";" (pname fn) 0 nil))
```

———-

7.4.58 defun compileBoot

[/D,1 p??]

— **defun compileBoot** —

```
(defun |compileBoot| (fn)
 (|/D,1| (list fn) '(/comp) nil nil))
```

———-

7.4.59 defun getTraceOptions

[throwKeyedMsg p??]
[throwListOfKeyedMsgs p??]
[poundsign p??]
[seq p??]
[exit p??]
[getTraceOption p104]
[$traceErrorStack p65]

— **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|
           "There are %1 problems with your )trace system command:"
           (cons (|#| |$traceErrorStack|) nil)
           (nreverse |$traceErrorStack|)))))
      (t optionlist)))))))
```

———-

7.4.60 defun saveMapSig

[rassoc p??]
[addassoc p??]
[getMapSig p103]
[$tracedMapSignatures p65]
[$mapSubNameAlist p61]

— **defun saveMapSig** —

```
(defun |saveMapSig| (funnames)
 (let (map)
 (declare (special |$tracedMapSignatures| |$mapSubNameAlist|))
  (dolist (name funnames)
   (when (setq map (|rassoc| name |$mapSubNameAlist|))
     (setq |$tracedMapSignatures|
       (addassoc name (|getMapSig| map name) |$tracedMapSignatures|))))))
```

———-

7.4.61 defun getMapSig

[get p??]
[boot-equal p??]
[$InteractiveFrame p34]

— **defun getMapSig** —

```
(defun |getMapSig| (mapname subname)
 (let (lmms sig)
 (declare (special |$InteractiveFrame|))
  (when (setq lmms (|get| mapname '|localModemap| |$InteractiveFrame|))
    (do ((t0 lmms (cdr t0)) (mm nil) (t1 nil sig))
        ((or (atom t0) (progn (setq mm (car t0)) nil) t1) nil)
      (when (equal (cadr mm) subname) (setq sig (cdar mm))))
    sig)))
```

———-

7.4.62 defun getTraceOption,hn

[seq p??]
[exit p??]
[isDomainOrPackage p94]
[stackTraceOptionError p88]
[domainToGenvar p88]

— **defun getTraceOption,hn** —

```
(defun |getTraceOption,hn| (x)
 (let (g)
  (cond
   ((and (atom x) (null (upper-case-p (elt (stringimage x) 0))))
     (cond
      ((|isDomainOrPackage| (eval x)) x)
      (t
       (|stackTraceOptionError|
        (cons
         (format nil
```

```
          "%1 The )trace option )of should be followed by the name of a ~
          domain and %2 is not one.")
         (list (list x)))))))
  ((setq g (|domainToGenvar| x)) g)
  (t (|stackTraceOptionError|
      (cons
       (format nil
        "%1 The )trace option )of should be followed by the name of a ~
         domain and %2 is not one.")
        (list (list x))))))))
```

———-

7.4.63 defun getTraceOption

[seq p??]
[exit p??]
[selectOptionLC p728]
[identp p1107]
[stackTraceOptionError p88]
[concat p1107]
[object2String p??]
[transOnlyOption p87]
[qcdr p??]
[qcar p??]
[getTraceOption,hn p103]
[isListOfIdentifiersOrStrings p90]
[isListOfIdentifiers p89]
[throwKeyedMsg p??]
[$traceOptionList p66]

— **defun getTraceOption** —

```
(defun |getTraceOption| (arg)
 (prog (l opts key a n)
 (declare (special |$traceOptionList|))
  (return
   (seq
    (progn
     (setq key (car arg))
     (setq l (cdr arg))
     (setq key (|selectOptionLC| key |$traceOptionList| '|traceOptionError|))
     (setq arg (cons key l))
     (cond
      ((member key '(|nonquietly| |timer| |nt|)) arg)
      ((eq key '|break|)
       (cond
        ((null l) (cons '|break| (cons '|before| nil)))
        (t
          (setq opts
           (loop for y in l collect
            (|selectOptionLC| y '(|before| |after|) nil)))
```

```
     (cond
      ((prog (t2)
        (setq t2 t)
        (return
         (do ((t3 nil (null t2)) (t4 opts (cdr t4)) (y nil))
             ((or t3 (atom t4) (progn (setq y (car t4)) nil)) t2)
            (seq
             (exit
              (setq t2 (and t2 (identp y))))))))
        (cons '|break| opts))
       (t
        (|stackTraceOptionError|
         (cons
          (format nil
           "%1 The )trace option )break can only have one or both of ~
           before and after as arguments.")
          (cons nil nil)))))))
((eq key '|restore|)
 (cond
  ((null l) arg)
  (t
   (|stackTraceOptionError|
    (cons "%1 The )trace option %2 can have no arguments."
     (cons (cons (concat ")" (|object2String| key)) nil) nil))))))
((eq key '|only|) (cons '|only| (|transOnlyOption| l)))
((eq key '|within|)
  (cond
   ((and (consp l)
         (eq (qcdr l) nil)
         (progn (setq a (qcar l)) t)
         (identp a))
     arg)
   (t
     (|stackTraceOptionError|
      (cons
       "%1 The )trace option %2 takes exactly one name as an argument."
       (cons (cons ")within" nil) nil))))))
((member key '(|cond| |before| |after|))
  (setq key
   (cond
    ((eq key '|cond|) '|when|)
    (t key)))
  (cond
   ((and (consp l)
         (eq (qcdr l) nil)
         (progn (setq a (qcar l)) t))
      (cons key l))
   (t
    (|stackTraceOptionError|
     (cons
   "%1 The )trace option %2 takes exactly one expression as an argument."
      (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
      "%1 The )trace option %2 takes exactly one integer argument."
      (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
       "%1 The )trace option %2 takes exactly one integer argument."
       (cons (cons ")count" nil) nil))))))
((eq key '|of|)
  (cons '|of| (loop for y in l collect (|getTraceOption,hn| y))))
((member key '(|local| ops vars))
  (cond
   ((or (null l)
        (and (consp l) (eq (qcdr l) nil) (eq (qcar l) '|all|)))
     (cons key '|all|))
   ((|isListOfIdentifiersOrStrings| l) arg)
   (t
     (|stackTraceOptionError|
      (cons
       "%1 The )trace option %2 should be followed by a list of names."
       (cons
        (cons (concat ")" (|object2String| key)) nil) nil))))))
((eq key '|varbreak|)
  (cond
   ((or (null l)
        (and (consp l) (eq (qcdr l) nil) (eq (qcar l) '|all|)))
     (cons '|varbreak| '|all|))
   ((|isListOfIdentifiers| l) arg)
   (t
     (|stackTraceOptionError|
      (cons
"%1 The )trace option %2 should be followed by a list of variable names."
       (cons
        (cons (concat ")" (|object2String| key)) nil) nil))))))
((eq key '|mathprint|)
  (cond
   ((null l) arg)
```

```
        (t
          (|stackTraceOptionError|
           (cons "%1 The )trace option %2 can have no arguments."
            (cons
             (cons (concat ")" (|object2String| key)) nil) nil))))))
      (key (|throwKeyedMsg| "The %1 option is not implemented yet."
            (cons key nil)))))))))
```

———-

7.4.64 defun traceOptionError

[stackTraceOptionError p88]
[commandAmbiguityError p705]

— **defun traceOptionError** —

```
(defun |traceOptionError| (opt keys)
 (if (null keys)
  (|stackTraceOptionError|
   (cons
    "%1 Axiom does not understand the )trace option %2 which you used."
    (cons (cons opt nil) nil)))
  (|commandAmbiguityError| '|trace option| opt keys)))
```

———-

7.4.65 defun genDomainTraceName

This function maintains an alist whose car is the domain to trace and the cdr is the genvar. So for POLY(INT) it is called with

```
y == (|Polynomial| (|Integer|))
```

which, if it is already traced would be in the alist as

```
(((|Polynomial| (|Integer|)) . $1))
```

If it is not already traced it would be cons with a new genvar and added to the alist.

[lassoc p**??**]
[genvar p**??**]
[$domainTraceNameAssoc p60]

— **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))))
```

———-

7.4.66 defun untrace

[copy p??]
[transTraceItem p111]
[/untrace,0 p??]
[lassocSub p91]
[removeTracedMapSigs p112]
[$lastUntraced p61]
[$mapSubNameAlist p61]
[$traceNames p66]

— **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)))
```

———-

7.4.67 defun /untrace-0

[/options p84]
[trunclist p85]
[/untrace-1 p109]

— **defun /untrace-0** —

```
(defun /untrace-0 (l)
 (let (optionl options fnl)
  (cond
   ((member '? l :test #'eq) (format t "Use )help trace~%"))
   (t
    (setq optionl (/options l))
```

```
    (setq fnl (trunclist l optionl))
    (setq options (if optionl (car optionl)))
    (/untrace-1 fnl options)))))
```

———-

7.4.68 defun /untrace-1

[/untrace-2 p109]
[/untrace-reduce p109]
[/tracereply p125]
[$traceNames p66]

— **defun /untrace-1** —

```
(defun /untrace-1 (l options)
 (declare (special |$traceNames|))
 (cond
  ((not l)
    (unless (atom |$traceNames|)
     (mapcar #'(lambda (u) (/untrace-2 (/untrace-reduce u) options))
       (append |$traceNames| nil))))
  ((mapcar #'(lambda (x) (/untrace-2 x options)) l)))
 (/tracereply))
```

———-

7.4.69 defun /untrace-reduce

(CAR X) is now a domain
/untrace-reduce : Union(Atom,List) → Atom

— **defun /untrace-reduce 0** —

```
(defun /untrace-reduce (x)
 (if (atom x) x (first x)))
```

———-

7.4.70 defun /untrace-2

[isFunctor p**??**]
[untraceDomainConstructor p122]
[isDomainOrPackage p94]
[isDomain p**??**]
[spadUntrace p126]
[eqcar p**??**]
[sayBrightly p**??**]
[unembed p**??**]
[isSubForRedundantMapName p92]

[rassocSub p91]
[isGenvar p111]
[devaluate p??]
[concat p1107]
[$traceNoisely p66]
[$mapSubNameAlist p61]
[$mathTraceList p62]
[$timerList p65]
[$traceNames p66]

— **defun /untrace-2** —

```
(defun /untrace-2 (x options)
 (declare (special |$traceNoisely| |$mapSubNameAlist| |$mathTraceList|
           |$timerList| |$traceNames|))
 (let (u y)
  (cond
   ((and (|isFunctor| X) (atom x))
     (|untraceDomainConstructor| X))
   ((or (|isDomainOrPackage| (setq u x))
        (and (symbolp X) (boundp X) (|isDomain| (setq u (eval x)))))
     (|spadUntrace| u options))
   ((eqcar options 'alias)
     (when |$traceNoisely|
       (|sayBrightly| (list (cadr options) '**untraced)))
     (setq |$timerList|
       (remove (princ-to-string (cadr options)) |$timerList| :test 'equal))
     (setq |$countList|
       (remove (princ-to-string (cadr options)) |$countList| :test 'equal))
     (setq |$mathTraceList|
       (remove (cadr options) |$mathTraceList| :test 'equal))
     (unembed x))
   ((and (not (member x |$traceNames|)) (not (|isSubForRedundantMapName| X)))
     (|sayBrightly| (list (|rassocSub| X |$mapSubNameAlist|) "not traced")))
   (t
     (setq |$traceNames| (remove x |$traceNames| :test 'equal))
     (setq |$mathTraceList|
       (remove (if (stringp x) (intern x) x) |$mathTraceList|))
     (setq |$letAssoc| (remove x |$letAssoc| :key #'car))
     (setq y (if (isGenvar x) (|devaluate| (eval x)) x))
     (setq |$timerList| (remove (princ-to-string y) |$timerList| :test 'equal))
     (set (intern (concat y ",TIMER")) 0)
     (setq |$countList| (remove (princ-to-string y) |$countList| :test 'equal))
     (set (intern (concat y ",COUNT")) 0)
     (when (and |$traceNoisely| (not (|isSubForRedundantMapName| y)))
        (|sayBrightly| (list (|rassocSub| Y |$mapSubNameAlist|) "untraced")))
     (unembed x)))))
```

————-

7.4.71 defun isGenvar

[identp p1107]
[size p1106]
[digitp p1106]

— **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))))))
```

————-

7.4.72 defun transTraceItem

For a call to trace INT this return Integer. For a call to trace `POLY(INT)` this returns a genvar (e.g. `$1`) which is set to the domain to be traced, recorded on the alist `$domainTraceNameAssoc`. In either case, a symbol is returned. [get p**??**]
[member p1108]
[objMode p462]
[objVal p462]
[domainToGenvar p88]
[unabbrev p**??**]
[constructor? p**??**]
[vecp p**??**]
[transTraceItem p111]
[devaluate p**??**]
[throwKeyedMsg p**??**]
[$doNotAddEmptyModeIfTrue p60]

transTraceItem : **traceArgument** → **symbol**

— **defun transTraceItem** —

```
(defun |transTraceItem| (x)
 (let (|$doNotAddEmptyModeIfTrue| |value| y)
  (declare (special |$doNotAddEmptyModeIfTrue|))
  (setq |$doNotAddEmptyModeIfTrue| t)
  (cond
   ((atom x)
     (cond
      ((and (setq |value| (|get| x '|value| |$InteractiveFrame|))
            (member (|objMode| |value|)
             '((|Mode|) (|Domain|) (|SubDomain| (|Domain|))) :test #'equalp))
        (setq x (|objVal| |value|))
        (cond
         ((setq y (|domainToGenvar| x)) y)
         (t x)))
      ;; case for trace INT
      ((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)))
   ((simple-vector-p (car x)) (|transTraceItem| (|devaluate| (car x))))
   ;; case for trace POLY(INT)
   ((setq y (|domainToGenvar| x)) y)
   (t (|throwKeyedMsg|
        "Axiom does not understand the use of %1 here." (list x))))))
```

———-

7.4.73 defun removeTracedMapSigs

[$tracedMapSignatures p65]

— **defun removeTracedMapSigs** —

```
(defun |removeTracedMapSigs| (untraceList)
 (declare (special |$tracedMapSignatures|))
 (dolist (name untraceList)
   (remprop name |$tracedMapSignatures|)))
```

———-

7.4.74 defun coerceTraceArgs2E

[spadsysnamep p**??**]
[pname p1106]
[coerceSpadArgs2E p113]
[objValUnwrap p462]
[coerceInteractive p658]
[mkObjWrap p461]
[$OutputForm p631]
[$mathTraceList p62]
[$tracedMapSignatures p65]

— **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 |$tracedMapSignatures|)) (cdr t3))
          (type nil)
          (t0 nil))
         ((or (atom t1)
              (progn (setq name (car t1)) nil)
              (atom t2)
              (progn (setq arg (car t2)) nil)
              (atom t3)
              (progn (setq type (car t3)) nil))
           (nreverse0 t0))
         (setq t0
          (cons
           (list '= name
            (|objValUnwrap|
             (|coerceInteractive|
              (mkObjWrap arg type) |$OutputForm|))) t0)))))
  ((spadsysnamep (pname name)) (reverse (cdr (reverse args))))
  (t args))))
```

———-

7.4.75 defun coerceSpadArgs2E

[seq p??]
[exit p??]
[objValUnwrap p462]
[coerceInteractive p658]
[mkObjWrap p461]
[$streamCount p956]
[$OutputForm p631]
[$tracedSpadModemap p67]

— **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|
             (mkObjWrap arg type)
             |$OutputForm|)) nil)))
         t0)))))))
```

———-

7.4.76 defun coerceTraceFunValue2E

[spadsysnamep p**??**]
[pname p1106]
[coerceSpadFunValue2E p115]
[lassoc p**??**]
[objValUnwrap p462]
[coerceInteractive p658]
[mkObjWrap p461]
[$tracedMapSignatures p65]
[$OutputForm p631]
[$mathTraceList p62]

— **defun coerceTraceFunValue2E** —

```
(defun |coerceTraceFunValue2E| (tracename subname |value|)
 (let (name u)
 (declare (special |$tracedMapSignatures| |$OutputForm| |$mathTraceList|))
   (if (member (setq name subname) |$mathTraceList|)
    (cond
     ((spadsysnamep (pname tracename)) (|coerceSpadFunValue2E| |value|))
     ((setq u (lassoc subname |$tracedMapSignatures|))
       (|objValUnwrap|
        (|coerceInteractive| (mkObjWrap |value| (car u)) |$OutputForm|)))
     (t |value|))
    |value|)))
```

———-

7.4.77 defun coerceSpadFunValue2E

[objValUnwrap p462]
[coerceInteractive p658]
[mkObjWrap p461]
[$streamCount p956]
[$tracedSpadModemap p67]
[$OutputForm p631]

— **defun coerceSpadFunValue2E** —

```
(defun |coerceSpadFunValue2E| (|value|)
 (let (|$streamCount|)
  (declare (special |$streamCount| |$tracedSpadModemap| |$OutputForm|))
  (setq |$streamCount| 0)
  (|objValUnwrap|
   (|coerceInteractive|
    (mkObjWrap |value| (car |$tracedSpadModemap|))
    |$OutputForm|))))
```

———-

7.4.78 defun getMapSubNames

[get p??]
[union p??]
[getPreviousMapSubNames p90]
[unionq p??]
[$lastUntraced p61]
[$InteractiveFrame p34]
[$traceNames p66]

— **defun getMapSubNames** —

```
(defun getMapSubNames (arg)
 (let (lmm subs)
 (declare (special |$traceNames| |$lastUntraced| |$InteractiveFrame|))
  (setq subs nil)
  (dolist (mapname arg)
   (when (setq lmm (|get| mapname '|localModemap| |$InteractiveFrame|))
     (setq subs
      (append
        (do ((t2 lmm (cdr t2)) (t1 nil) (|mm| nil))
            ((or (atom t2)
                 (progn (setq |mm| (CAR t2)) nil)) (nreverse0 t1))
           (setq t1 (cons (cons mapname (cadr |mm|)) t1)))
       subs))))
  (|union| subs
    (|getPreviousMapSubNames|
      (union |$traceNames| |$lastUntraced| :test #'eq)))))
```

———-

7.4.79 defun spadTrace,g

— defun spadTrace,g —

```
(defun |spadTrace,g| (x)
  (if (stringp x) (intern x) x))
```

————-

7.4.80 defun spadTrace,isTraceable

[seq p??]
[exit p??]
[gensymp p??]
[reportSpadTrace p125]
[bpiname p??]

— defun spadTrace,isTraceable —

```
(defun |spadTrace,isTraceable| (x domain)
 (prog (n |functionSlot|)
  (return
   (seq
    (progn
     (setq n (caddr x))
     x
     (seq
      (if (atom (elt domain n)) (exit nil))
      (setq |functionSlot| (car (elt domain n)))
      (if (gensymp |functionSlot|)
       (exit (seq (|reportSpadTrace| '|Already Traced| x) (exit nil))))
      (if (null (bpiname |functionSlot|))
       (exit
        (seq
         (|reportSpadTrace| '|No function for| x)
         (exit nil))))
      (exit t)))))))
```

————-

7.4.81 defun spadTrace

[refvecp p??]
[aldorTrace p??]
[isDomainOrPackage p94]
[userError p??]
[seq p??]
[exit p??]
[spadTrace,g p116]
[getOption p100]

[removeOption p88]
[opOf p??]
[assoc p??]
[flattenOperationAlist p94]
[getOperationAlistFromLisplib p119]
[spadTrace,isTraceable p116]
[as-insert p??]
[bpiname p??]
[spadTraceAlias p124]
[subTypes p89]
[constructSubst p651]
[bpitrace p120]
[rplac p??]
[printDashedLine p??]
[reportSpadTrace p125]
[setletprintflag p??]
[spadReply p100]
[$tracedModemap p??]
[$fromSpadTrace p61]
[$letAssoc p61]
[$reportSpadTrace p125]
[$traceNoisely p66]
[$traceNames p66]

— **defun spadTrace** —

```
(defun |spadTrace| (domain options)
 (let (listOfOperations listOfVariables listOfBreakVars anyifTrue
       domainId currentEntry currentAlist opStructureList triple
       sigSlotNumberAlist fn alias tracedModemap dn1 fgg tf)
   (declare (special |$fromSpadTrace|))
   (setq |$fromSpadTrace| t)
   (cond
    ((null (|isDomainOrPackage| domain))
      (|userError| "bad argument to trace"))
    (t
      (setq listOfOperations
       (loop for x in (getOption 'ops options)
        collect (if (stringp x) (intern x) x)))
      (when (setq listOfVariables (getOption 'vars options))
       (setq options (removeOption 'vars options)))
      (when (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 (cdr currentEntry))
      (setq opStructureList
        (flattenOperationAlist (|getOperationAlistFromLisplib| domainId)))
;     new form is (<op> <signature> <slotNumber> <condition> <kind>)
      (setq sigSlotNumberAlist
       (loop for arg in opStructureList
```

```
   when
    (and (eq (fifth arg) 'elt)
         (or anyifTrue (member (car arg) listOfOperations))
         (integerp (third arg))
         (|spadTrace,isTraceable|
           (setq triple (list (first arg) (second arg) (third arg)))
           domain))
   collect triple))
(when listOfVariables
  (loop for arg in sigSlotNumberAlist do
    (setq fn (car (elt domain (third arg))))
    (setq |$letAssoc|
      (as-insert (bpiname fn) listOfVariables |$letAssoc|))))
(when listOfBreakVars
  (loop for arg in sigSlotNumberAlist do
    (setq fn (car (elt domain (third arg))))
    (setq |$letAssoc|
      (as-insert (bpiname fn) (list (cons 'break listOfBreakVars))
        |$letAssoc|))))
(loop for pair in sigSlotNumberAlist do
  (setq alias (|spadTraceAlias| domainId (first pair) (third pair)))
  (setq tracedModemap
    (|subTypes| (second pair) (|constructSubst| (elt domain 0))))
  (setq dn1 (car (elt domain (third pair))))
  (setq fgg #'|newGoGet|)
  (setq tf
   (if (equal dn1 fgg)
     (|goGetTracerHelper| (elt domain (third pair)) fgg
         pair alias options tracedModemap)
     (bpitrace dn1 alias tracedModemap options)))
  (nconc pair (list listOfVariables (car (elt domain (third pair)))))
  (|rplac| (car (elt domain (third pair))) tf))
(setq sigSlotNumberAlist
  (loop for x in sigSlotNumberAlist
    when (cdddr x)
    collect x))
(when |$reportSpadTrace|
  (when |$traceNoisely| (|printDashedLine|))
  (loop for x in (|orderBySlotNumber| sigSlotNumberAlist) do
    (|reportSpadTrace| 'tracing x)))
(cond
 (currentEntry
   (|rplac| (cdr currentEntry) (append sigSlotNumberAlist currentAlist)))
 (t
   (setq |$traceNames|
     (cons (cons domain sigSlotNumberAlist) |$traceNames|))
   (|spadReply|)))))))
```

———-

7.4.82 defun getOperationAlistFromLisplib

[getdatabase p1070]
[addConsDB p??]
[markUnique p119]

— **defun getOperationAlistFromLisplib** —

```
(defun |getOperationAlistFromLisplib| (x)
 (let (opalist consdb)
  (setq opalist (getdatabase x 'operationalist))
  (cond
   ((null opalist) opalist)
   ((eq (caar opalist) '|$unique|) (cdr opalist))
   (t
    (setq consdb '(nil t elt))
    ((lambda (arg1 arg2)
     (loop
      (cond
       ((or (atom arg1) (progn (setq arg2 (car arg1)) nil))
         (return nil))
       (t
        (and (consp arg2)
             ((lambda (items)
              (loop
               (cond
                ((atom items) (return nil))
                (t
                 (cond
                  ((consp (cdar items))
                   (cond
                    ((consp (cddar items))
                      (cond
                       ((and (consp (cdddar items))
                             (eq (cdr (cdddar items)) nil)) nil)
                       (t (rplacd (cddar items) (cddr consdb)))))
                    (t (rplacd (cdar items) (cdr consdb)))))
                  (t (rplacd (car items) consdb)))
                 (rplaca items (car items))))
               (setq items (cdr items))))
              (cdr arg2)))))
      (setq arg1 (cdr arg1))))
     opalist nil)
    (and opalist (markUnique opalist))))))
```

———-

7.4.83 defun markUnique

— **defun markUnique 0** —

```
(defun markUnique (x)
```

```
 (let (u)
  (setq u (car x))
  (rplaca x '(|$unique|))
  (rplacd x (cons u (cdr x)))
  (cdr x)))
```

———-

7.4.84 defun bpitrace

[trace3 p72]

— **defun bpitrace** —

```
(defun bpitrace (binDef alias modemap options)
  (trace3 (gensym) modemap (cons (list 'alias alias) options) binDef))
```

———-

7.4.85 defun traceDomainLocalOps

[sayMSG p40]

— **defun traceDomainLocalOps** —

```
(defun |traceDomainLocalOps| ()
  (|sayMSG| '("  The )local option has been withdrawn"))
  (|sayMSG| '("  Use )ltr to trace local functions.")))
```

———-

7.4.86 defun untraceDomainLocalOps

[sayMSG p40]

— **defun untraceDomainLocalOps** —

```
(defun |untraceDomainLocalOps| ()
  (|sayMSG| '("  The )local option has been withdrawn"))
  (|sayMSG| '("  Use )ltr to trace local functions.")))
```

———-

7.4.87 defun traceDomainConstructor

[getOption p100]
[seq p??]
[exit p??]
[spadTrace p116]

[concat p1107]
[embed p??]
[mkq p??]
[loadFunctor p1095]
[traceDomainLocalOps p120]
[$ConstructorCache p??]

— defun traceDomainConstructor —

```
(defun |traceDomainConstructor| (domainConstructor options)
 (prog (listOfLocalOps argl domain innerDomainConstructor)
 (declare (special |$ConstructorCache|))
  (return
   (seq
    (progn
     (|loadFunctor| domainConstructor)
     (setq listOfLocalOps (getOption 'local options))
     (when listOfLocalOps (|traceDomainLocalOps|))
     (cond
      ((and listOfLocalOps (null (getOption 'ops options))) nil)
      (t
       (do ((t2 (hget |$ConstructorCache| domainConstructor) (cdr t2))
            (t3 nil))
           ((or (atom t2)
                (progn (setq t3 (car t2)) nil)
                (progn
                 (progn
                  (setq argl (car t3))
                  (setq domain (cddr t3)) t3)
                 nil))
               nil)
         (seq
          (exit
           (|spadTrace| domain options))))
       (setq |$traceNames| (cons domainConstructor |$traceNames|))
       (setq innerDomainConstructor
         (intern (concat domainConstructor ";")))
       (cond
        ((fboundp innerDomainConstructor)
          (setq domainConstructor innerDomainConstructor)))
       (embed domainConstructor
        (cons 'lambda
         (cons
          (cons '&rest
           (cons 'args nil))
          (cons
           (cons 'prog
            (cons
             (cons 'domain nil)
             (cons
              (cons 'setq
               (cons 'domain
                (cons
                 (cons 'apply (cons domainConstructor
```

```
            (cons 'args nil))) nil)))
        (cons
         (cons '|spadTrace|
          (cons 'domain
           (cons (mkq options) nil)))
         (cons (cons 'return (cons 'domain nil)) nil)))))
     nil)))))))))))
```

———-

7.4.88 defun untraceDomainConstructor,keepTraced?

[seq p??]
[qcar p??]
[isDomainOrPackage p94]
[boot-equal p??]
[devaluate p??]
[exit p??]
[/untrace,0 p??]

— **defun untraceDomainConstructor,keepTraced?** —

```
(defun |untraceDomainConstructor,keepTraced?| (df domainConstructor)
 (prog (dc)
  (return
   (seq
    (if (and
         (and
          (and (consp df) (progn (setq dc (qcar df)) t))
           (|isDomainOrPackage| dc))
          (boot-equal (ifcar (|devaluate| dc)) domainConstructor))
     (exit (seq (|/UNTRACE,0| (cons dc nil)) (exit nil))))
    (exit t)))))
```

———-

7.4.89 defun untraceDomainConstructor

[untraceDomainConstructor,keepTraced? p122]
[unembed p??]
[seq p??]
[exit p??]
[concat p1107]
[delete p??]
[$traceNames p66]

— **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|)))))))
```

———-

7.4.90 defun mapLetPrint

[getAliasIfTracedMapParameter p123]
[getBpiNameIfTracedMap p124]
[letPrint p95]

— **defun mapLetPrint** —

```
(defun |mapLetPrint| (x val currentFunction)
  (setq x (|getAliasIfTracedMapParameter| x currentFunction))
  (setq currentFunction (|getBpiNameIfTracedMap| currentFunction))
  (|letPrint| x val currentFunction))
```

———-

7.4.91 defun getAliasIfTracedMapParameter

[isSharpVarWithNum p96]
[get p??]
[exit p??]
[string2pint-n p??]
[substring p293]
[pname p1106]
[seq p??]
[$InteractiveFrame p34]

— **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|
           (-
            (string2pint-n (substring (pname x) 1 nil) 1) 1))))))
     (t x))))))
```

———-

7.4.92 defun getBpiNameIfTracedMap

[get p??]
[exit p??]
[seq p??]
[$InteractiveFrame p34]
[$traceNames p66]

— **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))))))
```

———-

7.4.93 defun spadTraceAlias

[internl p??]

— **defun spadTraceAlias** —

```
(defun |spadTraceAlias| (domainid op n)
 (internl domainid (intern "." "boot") op '|,| (princ-to-string n)))
```

———-

7.4.94 defun reportSpadTrace

[qcar p??]
[sayBrightly p??]
[$traceNoisely p66]

— 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 op
          (cons '|:|
            (cons (CDR sig)
             (cons '| -> |
              (cons (car sig)
               (cons '| in slot |
                (cons n nil)))))))))
      (setq |namePart| nil)
      (setq |tracePart|
       (cond
        ((and (consp |t|) (progn (setq y (qcar |t|)) t) (null (null y)))
          (cond
           ((eq y '|all|)
             (cons '|all| (cons '|vars| nil)))
           (t (cons '| vars: | (cons y nil)))))
        (t nil)))
      (|sayBrightly| (append |msg| (append |namePart| |tracePart|)))))))))
```

———-

7.4.95 defun /tracereply

[qcar p??]
[isDomainOrPackage p94]
[devaluate p??]
[seq p??]
[exit p??]

[$traceNames p66]

— 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)))))))))
```

———-

7.4.96 defun spadUntrace

[isDomainOrPackage p94]
[userError p??]
[getOption p100]
[devaluate p??]
[assoc p??]
[sayMSG p40]
[bright p??]
[prefix2String p??]
[bpiname p??]
[remover p101]
[setletprintflag p??]
[bpiuntrace p??]
[rplac p??]
[seq p??]
[exit p??]
[spadReply p100]
[$letAssoc p61]
[$traceNames p66]

— defun spadUntrace —

```
(defun |spadUntrace| (domain options)
```

```
(prog (anyiftrue listofoperations domainid |pair| sigslotnumberalist
       op sig n |lv| |bpiPointer| tracename alias |assocPair|
       |newSigSlotNumberAlist|)
(declare (special |$letAssoc| |$traceNames|))
 (return
  (seq
   (cond
    ((null (|isDomainOrPackage| domain))
      (|userError| "bad argument to untrace"))
    (t
     (setq anyiftrue (null options))
     (setq listofoperations (getOption '|ops:| options))
     (setq domainid (|devaluate| domain))
     (cond
      ((null (setq |pair| (|assoc| domain |$traceNames|)))
        (|sayMSG|
         (cons "   No functions in"
          (append
           (|bright| (|prefix2String| domainid))
           (cons "are now traced." nil)))))
      (t
       (setq sigslotnumberalist (cdr |pair|))
       (do ((t0 sigslotnumberalist (cdr t0)) (|pair| nil))
           ((or (atom t0)
                (progn (setq |pair| (car t0)) nil)
                (progn
                 (progn
                  (setq op (car |pair|))
                  (setq sig (cadr |pair|))
                  (setq n (caddr |pair|))
                  (setq |lv| (cadddr |pair|))
                  (setq |bpiPointer| (car (cddddr |pair|)))
                  (setq tracename (cadr (cddddr |pair|)))
                  (setq alias (caddr (cddddr |pair|)))
                  |pair|)
                 nil))
                nil)
        (seq
         (exit
          (cond
           ((or anyiftrue (member op listofoperations))
             (progn
               (bpiuntrace tracename alias)
               (rplac (car (elt domain n)) |bpiPointer|)
               (rplac (cdddr |pair|) nil)
               (cond
                ((setq |assocPair|
                    (|assoc| (bpiname |bpiPointer|) |$letAssoc|))
                  (setq |$letAssoc| (remover |$letAssoc| |assocPair|))
                  (cond
                   ((null |$letAssoc|) (setletprintflag nil))
                   (t nil)))
                (t nil))))))))
       (setq |newSigSlotNumberAlist|
```

```
         (prog (t1)
          (setq t1 nil)
          (return
           (do ((t2 sigslotnumberalist (cdr t2)) (x nil))
               ((or (atom t2) (progn (setq x (car t2)) nil)) (nreverse0 t1))
             (seq
              (exit
               (cond ((cdddr x) (setq t1 (cons x t1)))))))))
        (cond
         (|newSigSlotNumberAlist|
          (rplac (cdr |pair|) |newSigSlotNumberAlist|))
         (t
          (setq |$traceNames| (remove domain |$traceNames| :key #'car))
          (|spadReply|))))))
```

———-

7.4.97 defun prTraceNames,fn

[seq p??]
[qcar p??]
[qcdr p??]
[isDomainOrPackage p94]
[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)))))
```

———-

7.4.98 defun prTraceNames

[seq p??]
[exit p??]
[prTraceNames,fn p128]
[$traceNames p66]

— **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)))
```

———-

7.4.99 defun addTraceItem

[constructor? p??]
[isDomain p??]
[devaluate p??]
[isDomainOrPackage p94]
[$constructors p60]
[$domains p60]
[$packages p64]

— **defun addTraceItem** —

```
(defun |addTraceItem| (|d|)
 (declare (special |$constructors| |$domains| |$packages|))
 (cond
  ((|constructor?| |d|)
    (setq |$constructors| (cons |d| |$constructors|)))
  ((|isDomainOrPackage| |d|)
    (setq |$packages| (cons (|devaluate| |d|) |$packages|)))))
```

———-

7.4.100 defun ?t

[isgenvar p??]
[get p??]
[sayMSG p40]
[bright p??]
[rassocSub p91]
[qcar p??]
[qcdr p??]
[isDomainOrPackage p94]
[isDomain p??]
[reportSpadTrace p125]
[take p??]
[sayBrightly p??]
[devaluate p??]
[$mapSubNameAlist p61]
[$InteractiveFrame p34]
[$traceNames p66]

— defun ?t —

```
(defun |?t| ()
 (let (llm d suffix l)
  (declare (special |$traceNames| |$InteractiveFrame| |$mapSubNameAlist|))
  (if (null |$traceNames|)
   (|sayMSG| (|bright| "nothing is traced"))
   (progn
    (dolist (x |$traceNames|)
     (cond
      ((and (atom x) (null (isgenvar x)))
       (progn
        (cond
         ((setq llm (|get| x '|localModemap| |$InteractiveFrame|))
           (setq x (list (cadar llm)))))
        (|sayMSG|
         `("Function" ,@(|bright| (|rassocSub| x |$mapSubNameAlist|))
           "traced"))))))
    (dolist (x |$traceNames|)
     (cond
      ((and (consp x)
            (progn (setq d (qcar x)) (setq l (qcdr x)) t)
            (|isDomainOrPackage| d))
       (progn
        (setq suffix "package")
        (|sayBrightly|
         `("   Functions traced in " ,suffix ,(|devaluate| d) ":"))
        (dolist (x (|orderBySlotNumber| l))
          (|reportSpadTrace| '|   | (TAKE 4 x)))
        (terpri)))))))))
```

———-

7.4.101 defun Handle traced function entry

[sayBrightlyNT2 p??]
[coerceTraceArgs2E p112]
[spadsysnamep p??]
[monitorPrintArgs p131]
[$TraceFlag p65]
[$traceStream p67]
[$monitorDepth p62]
[$monitorFunDepth p62]
[$monitorCaller p62]
[$mathTrace p62]

— defun monitorEnter —

```
(defun monitorEnter (tracecode c type name name1)
 (let (|$TraceFlag| cArgs)
  (declare (special |$TraceFlag| |$traceStream| |$monitorDepth|
            |$monitorFunDepth| |$monitorCaller| |$mathTrace|))
```

```
  (setq |$TraceFlag| nil)
  (cond
   ((equal tracecode "000") nil)
   (t
     (tab 0 |$traceStream|)
     (princ (make-string (- |$monitorDepth| 1) :initial-element #\space)
            |$traceStream|)
     (prin1 |$monitorFunDepth| |$traceStream|)
     (format |$traceStream| "<enter ~a " (symbol-name name1))
     (unless (eql c 0)
      (if (eq type 'macro)
        (print " expanded" |$traceStream|)
        (progn
         (print " from " |$traceStream|)
         (prin1 |$monitorCaller| |$traceStream|))))
     (setq cArgs (|coerceTraceArgs2E| name1 name |$monitorArgs|))
     (when (spadsysnamep (symbol-name name))
       (setq cArgs (nreverse (reverse cArgs))))
     (monitorPrintArgs cArgs tracecode (get name '/transform))
     (unless |$mathTrace| (terpri |$traceStream|))))))
```

———-

7.4.102 defun Print the arguments to a traced function

Print the arguments to a traced function. [monitorPrint p132]
[monitorPrintRest p133]
[monitorPrintArg p132]
[mkq p**??**]
[prinmathor0 p133]
[$traceStream p67]
[$mathTrace p62]

— **defun monitorPrintArgs** —

```
(defun monitorPrintArgs (l code trans)
 (let (n)
  (cond
   ((eql (digit-char-p (elt code 2)) 0) nil)
   ((eql (digit-char-p (elt code 2)) 9)
    (cond
     (trans
      (loop for x in l for y in (cdr trans) do
       (cond
        ((eq y '*)
          (princ "\\ " |$traceStream|)
          (monitorPrint x |$traceStream|))
        ((eq y '&)
          (princ "\\\\" |$traceStream|)
          (terpri |$traceStream|)
          (print x |$traceStream|))
        ((null y)
```

```
          (princ "! " |$traceStream|))
         (t
          (princ "! " |$traceStream|)
          (monitorPrint (eval (subst (mkq x) '* y)) |$traceStream|)))))
     (t
       (princ ": " |$traceStream|)
       (unless (atom l)
          (when |$mathTrace| (terpri |$traceStream|))
          (monitorPrint (car l) |$traceStream|)
          (setq l (cdr l)))
       (loop for el in l do (monitorPrintRest el)))))
   (t
     (loop for istep from 2 to (1- (length code)) do
       (setq n (digit-char-p (elt code istep)))
       (unless (eql n 0)
         (princ "\\" |$traceStream|)
         (prinmathor0 n |$traceStream|)
         (princ ": " |$traceStream|)
         (monitorPrintArg l n)))))))
```

———-

7.4.103 defun monitorPrintArg

[monitorPrint p132]
[$traceStream p67]

— **defun monitorPrintArg** —

```
(defun monitorPrintArg (l n)
  (loop for el in l for k from 1 to n do
    (when (= k n) (monitorPrint el |$traceStream|))))
```

———-

7.4.104 defun monitorPrint

[smallEnough p135]
[limitedPrint1 p135]
[prinmathor0 p133]
[$monitorPretty p63]

— **defun monitorPrint** —

```
(defun monitorPrint (x tracestr)
 (cond
  ((null (smallEnough x)) (limitedPrint1 x tracestr))
  (|$monitorPretty| (prettyprint x tracestr))
  (t (prinmathor0 x tracestr))))
```

———-

7.4.105 defun monitorPrintRest

[smallEnough p135]
[monitorBlanks p??]
[prinmathor0 p133]
[$traceStream p67]
[$monitorDepth p62]
[$monitorPretty p63]
[$mathTrace p62]

— **defun monitorPrintRest** —

```
(defun monitorPrintRest (x)
 (cond
  ((null (smallEnough x))
    (terpri |$traceStream|)
    (monitorBlanks (+ |$monitorDepth| 1))
    (princ "\\" |$traceStream|)
    (print x |$traceStream|))
  (t
    (unless |$mathTrace| (princ "\\" |$traceStream|))
    (if |$monitorPretty|
     (prettyprint x |$traceStream|)
     (prinmathor0 x |$traceStream|)))))
```

———-

7.4.106 defun prinmathor0

[maprinSpecial p??]
[outputTran2 p??]
[$mathTrace p62]
[$monitorDepth p62]

— **defun prinmathor0** —

```
(defun prinmathor0 (x tracestr)
 (if |$mathTrace|
  (|maprinSpecial| (|outputTran2| x) |$monitorDepth| 80)
  (prin1 x tracestr)))
```

———-

7.4.107 defun Handle traced function exit

[tab p??]
[monitorPrintValue p134]
[coerceTraceFunValue2E p114]
[$TraceFlag p65]
[$traceStream p67]
[$monitorFunDepth p62]

[$mathTrace p62]
[$monitorValue p63]

— **defun monitorExit** —

```
(defun monitorExit (tracecode name name1 v timernam evalTime)
 (let (|$TraceFlag|)
  (declare (special |$TraceFlag|))
  (setq |$TraceFlag| nil)
  (cond
   ((equal tracecode "000") nil)
   (t
    (tab 0 |$traceStream|)
    (princ (make-string (- |$monitorDepth| 1) :initial-element #\space)
           |$traceStream|)
    (prin1 |$monitorFunDepth| |$traceStream|)
    (format |$traceStream| ">exit ~a " (symbol-name name1))
    (when timernam (format |$traceStream| "(~a sec)" (/ evalTime 60.0)))
    (when (eql v 1)
      (monitorPrintValue
       (|coerceTraceFunValue2E| name1 name |$monitorValue|)
       name1))
    (unless |$mathTrace| (terpri |$traceStream|))))))
```

———-

7.4.108 defun monitorPrintValue

[eqcar p??]
[mkq p??]
[smallEnough p135]
[limitedPrint1 p135]
[prinmathor0 p133]
[$traceStream p67]
[$monitorPretty p63]
[$mathTrace p62]

— **defun monitorPrintValue** —

```
(defun monitorPrintValue (val name)
 (let (u)
  (setq u (get name '/transform))
  (cond
   (u
    (cond
     ((eqcar u '&)
       (format |$traceStream| "//~a~%" val))
     (t
       (format |$traceStream| "! ~a~%" (eval (subst (mkq val) '* (car u)))))))
   (t
     (princ ": " |$traceStream|)
     (cond
      ((null (smallEnough val)) (limitedPrint1 val |$traceStream|))
```

```
      (|$monitorPretty| (prettyprint val |$traceStream|))
      (t
        (when |$mathTrace| (terpri |$traceStream|))
        (prinmathor0 val |$traceStream|)))))))
```

———-

7.4.109 defun limitedPrint1

[*print-level* p??]
[*print-length* p??]

— **defun limitedPrint1 0** —

```
(defun limitedPrint1 (form stream)
 (let ((*print-level* 4) (*print-length* 4))
  (prin1 form stream)
  (terpri stream)))
```

———-

7.4.110 defun smallEnough

— **defun smallEnough** —

```
(defun smallEnough (x)
  (if |$traceSize|
   (< (smallEnoughCount x 0 |$traceSize|) |$traceSize|)
   t))
```

———-

7.4.111 defun How big is an object?

We need to know how many elements there are in the object x. The n argument should be 0 for a top-level call and m is the maximum length allowed.

— **defun smallEnoughCount 0** —

```
(defun smallEnoughCount (x n m)
 (cond
  ((null (< n m)) n)
  ((simple-vector-p x)
   (loop for i from 0 to (1- (length x)) while (< n m)
      do (setq n (smallEnoughCount (elt x i) (+ n 1) m)))
   n)
  ((atom x) n)
  (t
   (setq n (smallEnoughCount (car x) (+ n 1) m))
   (if (null (< n m)) n (smallEnoughCount (cdr x) n m)))))
```

———-

7.4.112 defun tracelet

[stupidIsSpadFunction p101]
[bpiname p??]
[lassoc p??]
[union p??]
[isGenvar p111]
[gensymp p??]
[compileBoot p101]
[delete p??]
[$traceletflag p66]
[$QuickLet p64]
[$letAssoc p61]
[$traceletFunctions p66]

— **defun tracelet** —

```
(defun |tracelet| (fn binDef vars)
 (let (|$QuickLet| $traceletflag l)
  (declare (special |$QuickLet| $traceletflag |$letAssoc|
            |$traceletFunctions|))
  (when (and binDef (|stupidIsSpadFunction| binDef))
    (when (compiled-function-p binDef) (setq fn (bpiname binDef))))
  (cond ((eq fn '|Undef|) nil)
  (t
    (setq vars
      (cond
       ((eq vars '|all|) '|all|)
       ((setq l (lassoc fn |$letAssoc|)) (|union| vars l))
       (t vars)))
    (setq |$letAssoc| (cons (cons fn vars) |$letAssoc|))
    (setq $traceletflag t)
    (setq |$QuickLet| nil)
    (when (and (null (member fn |$traceletFunctions|))
               (null (isGenvar fn))
               (compiled-function-p (symbol-function fn))
               (null (|stupidIsSpadFunction| fn))
               (null (gensymp fn)))
      (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
      (|compileBoot| fn)
      (setq |$traceletFunctions| (|delete| fn |$traceletFunctions|)))))))
```

———-

7.4.113 defun breaklet

[gensymp p??]
[stupidIsSpadFunction p101]
[bpiname p??]

[lassoc p??]
[assoc p??]
[union p??]
[setletprintflag p??]
[compileBoot p101]
[delete p??]
[$QuickLet p64]
[$letAssoc p61]
[$traceletFunctions p66]

— **defun breaklet** —

```
; vars is "all" or a list of variables
; $letAssoc ==> (.. (=fn .. (BREAK . all))) OR (.. (=fn .. (BREAK . vl)))
(defun |breaklet| (fn binDef vars)
 (let (|$QuickLet| pair fnEntry)
  (declare (special |$QuickLet| |$letAssoc| |$traceletFunctions|))
  (when (and binDef (|stupidIsSpadFunction| binDef))
    (when (compiled-function-p binDef) (setq fn (bpiname binDef))))
  (cond
   ((eq fn '|Undef|) nil)
   (t
    (setq fnEntry (lassoc fn |$letAssoc|))
    (setq vars
     (if (setq pair (|assoc| 'break fnEntry))
       (|union| vars (cdr pair))
       vars))
     (setq |$letAssoc|
      (cond
       ((null fnEntry)
          (cons (cons fn (list (cons 'break vars))) |$letAssoc|))
       (pair (rplacd pair vars) |$letAssoc|)))
     (setq |$QuickLet| nil)
     (when (and (null (member fn |$traceletFunctions|))
                (null (|stupidIsSpadFunction| fn))
                (null (gensymp fn)))
       (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
       (|compileBoot| fn)
       (setq |$traceletFunctions| (|delete| fn |$traceletFunctions|)))))))
```

———-

7.4.114 defun break

[MONITOR,EVALTRAN p??]
[sayBrightly p??]
[/breakcondition p??]

— **defun break** —

```
(defun |break| (msg)
 (let (condition)
```

```
(declare (special /breakcondition))
 (setq condition (|MONITOR,EVALTRAN| /breakcondition nil))
 (when (eval condition) (|sayBrightly| msg))))
```

Chapter 8

Exposure groups

8.1 Functions to manipulate exposure

8.1.1 Expose a group

Note that `$localExposureData` is a vector of lists. It consists of [exposed groups,exposed constructors,hidden constructors]

[object2String p??]
[qcar p??]
[displayExposedGroups p146]
[sayMSG p40]
[displayExposedConstructors p146]
[displayHiddenConstructors p147]
[clearClams p??]
[getalist p??]
[sayKeyedMsg p39]
[member p1108]
[msort p??]
[specialChar p1043]
[namestring p1102]
[pathname p1103]
[sayAsManyPerLineAsPossible p??]
[$globalExposureGroupAlist p148]
[$localExposureData p147]
[$interpreterFrameName p34]
[$linelength p936]

— **defun setExposeAddGroup** —

```
(defun |setExposeAddGroup| (arg)
 "Expose a group"
 (declare (special |$globalExposureGroupAlist| |$localExposureData|
                   |$interpreterFrameName| $linelength))
 (if (null arg)
  (progn
```

```
  (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " The group Option ")
  (|displayExposedGroups|)
  (|sayMSG| " ")
  (|sayAsManyPerLineAsPossible|
   (mapcar #'(lambda (x) (|object2String| (first x)))
     |$globalExposureGroupAlist|)))
 (dolist (x arg)
   (when (consp x) (setq x (qcar x)))
   (cond
    ((eq x '|all|)
     (setf (elt |$localExposureData| 0)
       (mapcar #'first |$globalExposureGroupAlist|))
     (setf (elt |$localExposureData| 1) nil)
     (setf (elt |$localExposureData| 2) nil)
     (|displayExposedGroups|)
     (|sayMSG| " ")
     (|displayExposedConstructors|)
     (|sayMSG| " ")
     (|displayHiddenConstructors|)
     (|clearClams|))
    ((null (getalist |$globalExposureGroupAlist| x))
     (|sayKeyedMsg| "%1 is not a known exposure group name." (cons x nil)))
    ((|member| x (elt |$localExposureData| 0))
     (|sayKeyedMsg| "%1 is already an exposure group for frame %2"
       (list x |$interpreterFrameName|)))
    (t
     (setf (elt |$localExposureData| 0)
       (msort (cons x (elt |$localExposureData| 0))))
     (|sayKeyedMsg| "%1 is now an exposure group for frame %2"
       (list x |$interpreterFrameName|))
     (|clearClams|))))))
```

———-

8.1.2 The top level set expose command handler

[displayExposedGroups p146]
[sayMSG p40]
[displayExposedConstructors p146]
[displayHiddenConstructors p147]
[sayKeyedMsg p39]
[namestring p1102]
[pathname p1103]
[qcar p??]
[qcdr p??]
[selectOptionLC p728]
[setExposeAdd p141]
[setExposeDrop p143]
[setExpose p140]

— **defun setExpose** —

```
(defun |setExpose| (arg)
 "The top level set expose command handler"
 (let (fnargs fn)
   (cond
    ((eq arg '|%initialize%|))
    ((eq arg '|%display%|) "...")
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|displayExposedGroups|)
     (|sayMSG| " ")
     (|displayExposedConstructors|)
     (|sayMSG| " ")
     (|displayHiddenConstructors|)
     (|sayMSG| " "))
    ((and (consp arg)
          (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
          (setq fn (|selectOptionLC| fn '(|add| |drop|) nil)))
     (cond
      ((eq fn '|add|) (|setExposeAdd| fnargs))
      ((eq fn '|drop|) (|setExposeDrop| fnargs))
      (t nil)))
    (t (|setExpose| nil)))))
```

———-

8.1.3 The top level set expose add command handler

[specialChar p1043]
[displayExposedGroups p146]
[sayMSG p40]
[displayExposedConstructors p146]
[sayKeyedMsg p39]
[qcar p??]
[qcdr p??]
[selectOptionLC p728]
[setExposeAddGroup p139]
[setExposeAddConstr p142]
[setExposeAdd p141]
[$linelength p936]

— **defun setExposeAdd** —

```
(defun |setExposeAdd| (arg)
 "The top level set expose add command handler"
 (declare (special $linelength))
 (let (fnargs fn)
   (cond
    ((null arg)
     (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " The add Option ")
     (|displayExposedGroups|)
     (|sayMSG| " ")
     (|displayExposedConstructors|)
     (|sayMSG| " ")
```

```
    (|sayKeyedMsg|
     (format nil
      "When )set expose add is followed by no arguments, the information ~
       you now see is displayed. ~
       The arguments group and constructor are used to specify ~
       exposure groups or an explicit constructor to be added to the local ~
       frame exposure data. Issue ~
       %ceon )set expose add group %ceoff or ~
       %ceon )set expose add constructor %ceoff ~
       for more information.")
      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)))))
```

———-

8.1.4 The top level set expose add constructor handler

[unabbrev p??]
[qcar p??]
[getdatabase p1070]
[sayKeyedMsg p39]
[member p1108]
[delete p??]
[msort p??]
[clearClams p??]
[specialChar p1043]
[displayExposedConstructors p146]
[$linelength p936]
[$localExposureData p147]
[$interpreterFrameName p34]

— **defun setExposeAddConstr** —

```
(defun |setExposeAddConstr| (arg)
 "The top level set expose add constructor handler"
 (declare (special $linelength |$localExposureData| |$interpreterFrameName|))
 (if (null arg)
   (progn
    (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " The constructor Option ")
    (|displayExposedConstructors|))
   (dolist (x arg)
     (setq x (|unabbrev| x))
     (when (consp x) (setq x (qcar x)))
     (cond
      ((null (getdatabase x 'constructorkind))
```

```
      (|sayKeyedMsg|
       (format nil "%1 is not a known constructor. ~
          You can make the constructor known to the system by loading it.")
        (list x)))
     ((|member| x (elt |$localExposureData| 1))
      (|sayKeyedMsg| "%1 is already explicitly exposed in frame %2"
        (list x |$interpreterFrameName| )))
     (t
      (when (|member| x (elt |$localExposureData| 2))
        (setf (elt |$localExposureData| 2)
         (|delete| x (elt |$localExposureData| 2))))
      (setf (elt |$localExposureData| 1)
       (msort (cons x (elt |$localExposureData| 1))))
      (|clearClams|)
      (|sayKeyedMsg| "%1 is now explicitly exposed in frame %2"
       (list x |$interpreterFrameName| )))))))
```

———-

8.1.5 The top level set expose drop handler

[specialChar p1043]
[displayHiddenConstructors p147]
[sayMSG p40]
[sayKeyedMsg p39]
[qcar p??]
[qcdr p??]
[selectOptionLC p728]
[setExposeDropGroup p144]
[setExposeDropConstr p145]
[setExposeDrop p143]
[$linelength p936]

— **defun setExposeDrop** —

```
(defun |setExposeDrop| (arg)
 "The top level set expose drop handler"
 (declare (special $linelength))
 (let (fnargs fn)
   (cond
    ((null arg)
     (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " The drop Option ")
     (|displayHiddenConstructors|)
     (|sayMSG| " ")
     (|sayKeyedMsg|
      (format nil "When )set expose drop is followed by no arguments, the ~
       information you now see is displayed. The arguments group and ~
       constructor are used to specify exposure groups or an explicit ~
       constructor to be dropped from the local frame exposure data. Issue ~
       %ceon )set expose drop group %ceoff or %ceon )set expose drop ~
       constructor %ceoff for more information.")
      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)))))
```

———-

8.1.6 The top level set expose drop group handler

[qcar p??]
[displayExposedGroups p146]
[sayMSG p40]
[displayExposedConstructors p146]
[displayHiddenConstructors p147]
[clearClams p??]
[member p1108]
[delete p??]
[sayKeyedMsg p39]
[getalist p??]
[specialChar p1043]
[$linelength p936]
[$localExposureData p147]
[$interpreterFrameName p34]
[$globalExposureGroupAlist p148]

— **defun setExposeDropGroup** —

```
(defun |setExposeDropGroup| (arg)
 "The top level set expose drop group handler"
 (declare (special $linelength |$localExposureData| |$interpreterFrameName|
                   |$globalExposureGroupAlist|))
 (if (null arg)
  (progn
   (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " The group Option ")
   (|sayKeyedMsg|
    (format nil "When followed by one or more exposure group names, this ~
                 option allows you to remove those groups from the local ~
                 frame exposure data.")
    nil)
   (|sayMSG| " ")
   (|displayExposedGroups|))
  (dolist (x arg)
    (when (consp x) (setq x (qcar x)))
    (cond
     ((eq x '|all|)
      (setf (elt |$localExposureData| 0) nil)
      (setf (elt |$localExposureData| 1) nil)
      (setf (elt |$localExposureData| 2) nil)
```

```
      (|displayExposedGroups|)
      (|sayMSG| " ")
      (|displayExposedConstructors|)
      (|sayMSG| " ")
      (|displayHiddenConstructors|)
      (|clearClams|))
     ((|member| x (elt |$localExposureData| 0))
      (setf (elt |$localExposureData| 0)
       (|delete| x (elt |$localExposureData| 0)))
      (|clearClams|)
      (|sayKeyedMsg| "%1 is no longer an exposure group for frame %2"
        (list x |$interpreterFrameName| )))
     ((getalist |$globalExposureGroupAlist| x)
      (|sayKeyedMsg| "%1 is already an exposure group for frame %2"
        (list x |$interpreterFrameName| )))
     (t (|sayKeyedMsg| "%1 is not a known exposure group name." (list x )))))))
```

———-

8.1.7 The top level set expose drop constructor handler

[unabbrev p??]
[qcar p??]
[getdatabase p1070]
[sayKeyedMsg p39]
[member p1108]
[delete p??]
[msort p??]
[clearClams p??]
[specialChar p1043]
[sayMSG p40]
[displayExposedConstructors p146]
[displayHiddenConstructors p147]
[$linelength p936]
[$localExposureData p147]
[$interpreterFrameName p34]

— **defun setExposeDropConstr** —

```
(defun |setExposeDropConstr| (arg)
 "The top level set expose drop constructor handler"
 (declare (special $linelength |$localExposureData| |$interpreterFrameName|))
 (if (null arg)
  (progn
   (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " The constructor Option ")
   (|sayKeyedMsg|
    (format nil "When followed by one or more constructor names, this option ~
                 allows you to explicitly hide constructors in this frame.")
    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|
       (format nil "%1 is not a known constructor. ~
        You can make the constructor known to the system by loading it.")
       (list x)))
     ((|member| x (elt |$localExposureData| 2))
      (|sayKeyedMsg| "%1 is already explicitly hidden in frame %2"
       (list x |$interpreterFrameName|)))
     (t
      (when (|member| x (elt |$localExposureData| 1))
       (setf (elt |$localExposureData| 1)
        (|delete| x (elt |$localExposureData| 1))))
      (setf (elt |$localExposureData| 2)
        (msort (cons x (elt |$localExposureData| 2))))
      (|clearClams|)
      (|sayKeyedMsg| "%1 is now explicitly hidden in frame %2"
       (list x |$interpreterFrameName|)))))))
```

———-

8.1.8 Display exposed groups

[sayKeyedMsg p39]
[$interpreterFrameName p34]
[$localExposureData p147]

— **defun displayExposedGroups** —

```
(defun |displayExposedGroups| ()
 "Display exposed groups"
 (declare (special |$interpreterFrameName| |$localExposureData|))
 (|sayKeyedMsg|
  (format nil "The following groups are explicitly exposed in the current ~
              frame (called %1 ):")
  (list |$interpreterFrameName|))
 (if  (null (elt |$localExposureData| 0))
  (format t "~v:@<~a~>~%" (- $linelength 2) " there are no exposed groups ")
  (dolist (c (elt |$localExposureData| 0))
     (format t "~v:@<~a~>~%" (- $linelength 2) c))))
```

———-

8.1.9 Display exposed constructors

[sayKeyedMsg p39]
[$localExposureData p147]

— **defun displayExposedConstructors** —

```
(defun |displayExposedConstructors| ()
 "Display exposed constructors"
 (declare (special |$localExposureData|))
 (|sayKeyedMsg|
  "The following constructors are explicitly exposed in the current frame:"
  nil)
 (if  (null (elt |$localExposureData| 1))
  (format t "~v:@<~a~>~%" (- $linelength 2)
    "there are no explicitly exposed constructors")
  (dolist (c (elt |$localExposureData| 1))
    (format t "~v:@<~a~>~%" (- $linelength 2) c))))
```

———-

8.1.10 Display hidden constructors

[sayKeyedMsg p39]
[$localExposureData p147]

— **defun displayHiddenConstructors** —

```
(defun |displayHiddenConstructors| ()
 "Display hidden constructors"
 (declare (special |$localExposureData|))
 (|sayKeyedMsg|
  "The following constructors are explicitly hidden in the current frame:"
  nil)
 (if (null (elt |$localExposureData| 2))
  (format t "~v:@<~a~>~%" (- $linelength 2)
      "there are no explicitly hidden constructors")
   (dolist (c (elt |$localExposureData| 2))
     (format t "~v:@<~a~>~%" (- $linelength 2) c))))
```

———-

8.2 Exposure Data Structures

8.2.1 defvar $localExposureData

— **postvars** —

```
(defvar |$localExposureData|  (copy-seq |$localExposureDataDefault|))
```

———-

8.2.2 defvar $localExposureDataDefault

— **initvars** —

```
(defvar |$localExposureDataDefault|
  (vector
    ;;These groups will be exposed
   (list '|basic| '|categories| '|naglink| '|anna|)
    ;;These constructors will be explicitly exposed
   (list )
    ;;These constructors will be explicitly hidden
   (list )))
```

———-

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.

8.2.3 defvar $globalExposureGroupAlist

— **initvars** —

```
(defvar |$globalExposureGroupAlist|
 '(
 ;;define the groups |basic| |naglink| |anna| |categories| |Hidden| |defaults|
  (|basic|
   (|AffineAlgebraicSetComputeWithGroebnerBasis| . AFALGGRO)
   (|AffineAlgebraicSetComputeWithResultant| . AFALGRES)
   (|AffinePlane| . AFFPL)
   (|AffinePlaneOverPseudoAlgebraicClosureOfFiniteField| . AFFPLPS)
   (|AffineSpace| . AFFSP)
   (|AlgebraicManipulations| . ALGMANIP)
   (|AlgebraicNumber| . AN)
   (|AlgFactor| . ALGFACT)
   (|AlgebraicMultFact| . ALGMFACT)
   (|AlgebraPackage| . ALGPKG)
   (|AlgebraGivenByStructuralConstants| . ALGSC)
   (|Any| . ANY)
   (|AnyFunctions1| . ANY1)
   (|ApplicationProgramInterface| . API)
   (|ArrayStack| . ASTACK)
   (|AssociatedJordanAlgebra| . JORDAN)
   (|AssociatedLieAlgebra| . LIE)
   (|AttachPredicates| . PMPRED)
   (|AxiomServer| . AXSERV)
   (|BalancedBinaryTree| . BBTREE)
   (|BasicStochasticDifferential| . BSD)
   (|BasicOperator| . BOP)
   (|BasicOperatorFunctions1| . BOP1)
   (|Bezier| . BEZIER)
   (|BinaryExpansion| . BINARY)
   (|BinaryFile| . BINFILE)
```

```
(|BinarySearchTree| . BSTREE)
(|BinaryTournament| . BTOURN)
(|BinaryTree| . BTREE)
(|Bits| . BITS)
(|BlasLevelOne| . BLAS1)
(|BlowUpPackage| . BLUPPACK)
(|BlowUpWithHamburgerNoether| . BLHN)
(|BlowUpWithQuadTrans| . BLQT)
(|Boolean| . BOOLEAN)
(|CardinalNumber| . CARD)
(|CartesianTensor| . CARTEN)
(|CartesianTensorFunctions2| . CARTEN2)
(|Cell| .  CELL)
(|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)
(|CylindricalAlgebraicDecompositionPackage| . CAD)
(|CylindricalAlgebraicDecompositionUtilities| . CADU)
(|Database| . DBASE)
(|DataList| . DLIST)
(|DecimalExpansion| . DECIMAL)
(|DenavitHartenbergMatrix| . DHMATRIX)
(|Dequeue| . DEQUEUE)
(|DesingTree| . DSTREE)
(|DesingTreePackage| . DTP)
(|DiophantineSolutionPackage| . DIOSP)
(|DirichletRing| . DIRRING)
(|DirectProductFunctions2| . DIRPROD2)
(|DisplayPackage| . DISPLAY)
(|DistinctDegreeFactorize| . DDFACT)
(|Divisor| . DIV)
(|DoubleFloat| . DFLOAT)
(|DoubleFloatMatrix| . DFMAT)
(|DoubleFloatVector| . DFVEC)
(|DoubleFloatSpecialFunctions| . DFSFUN)
(|DrawComplex| . DRAWCX)
(|DrawNumericHack| . DRAWHACK)
(|DrawOption| . DROPT)
(|EigenPackage| . EP)
```

```
 (|ElementaryFunctionDefiniteIntegration| . DEFINTEF)
 (|ElementaryFunctionLODESolver| . LODEEF)
 (|ElementaryFunctionODESolver| . ODEEF)
 (|ElementaryFunctionSign| . SIGNEF)
 (|ElementaryFunctionStructurePackage| . EFSTRUC)
 (|Equation| . EQ)
 (|EquationFunctions2| . EQ2)
 (|ErrorFunctions| . ERROR)
 (|EuclideanGroebnerBasisPackage| . GBEUCLID)
 (|Exit| . EXIT)
 (|Export3D| . EXP3D)
 (|Expression| . EXPR)
 (|ExpressionFunctions2| . EXPR2)
 (|ExpressionSolve| . EXPRSOL)
 (|ExpressionSpaceFunctions2| . ES2)
 (|ExpressionSpaceODESolver| . EXPRODE)
 (|ExpressionToOpenMath| . OMEXPR)
 (|ExpressionToUnivariatePowerSeries| . EXPR2UPS)
 (|Factored| . FR)
 (|FactoredFunctions2| . FR2)
(|FactorisationOverPseudoAlgebraicClosureOfAlgExtOfRationalNumber| . FACTEXT)
 (|FactorisationOverPseudoAlgebraicClosureOfRationalNumber| . FACTRN)
 (|File| . FILE)
 (|FileName| . FNAME)
 (|FiniteAbelianMonoidRingFunctions2| . FAMR2)
 (|FiniteDivisorFunctions2| . FDIV2)
 (|FiniteField| . FF)
 (|FiniteFieldFactorization| . FFFACTOR)
 (|FiniteFieldFactorizationWithSizeParseBySideEffect| . FFFACTSE)
 (|FiniteFieldCyclicGroup| . FFCG)
 (|FiniteFieldPolynomialPackage2| . FFPOLY2)
 (|FiniteFieldNormalBasis| . FFNB)
 (|FiniteFieldHomomorphisms| . FFHOM)
 (|FiniteFieldSquareFreeDecomposition| . FFSQFR)
 (|FiniteLinearAggregateFunctions2| . FLAGG2)
 (|FiniteLinearAggregateSort| . FLASORT)
 (|FiniteSetAggregateFunctions2| . FSAGG2)
 (|FlexibleArray| . FARRAY)
 (|Float| . FLOAT)
 (|FloatingRealPackage| . FLOATRP)
 (|FloatingComplexPackage| . FLOATCP)
 (|FloatSpecialFunctions| . FSFUN)
 (|FourierSeries| . FSERIES)
 (|Fraction| . FRAC)
 (|FractionalIdealFunctions2| . FRIDEAL2)
 (|FractionFreeFastGaussian| . FFFG)
 (|FractionFreeFastGaussianFractions| . FFFGF)
 (|FractionFunctions2| . FRAC2)
 (|FreeNilpotentLie| . FNLA)
 (|FullPartialFractionExpansion| . FPARFRAC)
 (|FunctionFieldCategoryFunctions2| . FFCAT2)
 (|FunctionSpaceAssertions| . PMASSFS)
 (|FunctionSpaceAttachPredicates| . PMPREDFS)
 (|FunctionSpaceComplexIntegration| . FSCINT)
```

```
(|FunctionSpaceFunctions2| . FS2)
(|FunctionSpaceIntegration| . FSINT)
(|FunctionSpacePrimitiveElement| . FSPRMELT)
(|FunctionSpaceSum| . SUMFS)
(|GaussianFactorizationPackage| . GAUSSFAC)
(|GeneralPackageForAlgebraicFunctionField| . GPAFF)
(|GeneralUnivariatePowerSeries| . GSERIES)
(|GenerateUnivariatePowerSeries| . GENUPS)
(|Graphviz| . GRAPHVIZ)
(|GnuDraw| . GDRAW)
(|GraphicsDefaults| . GRDEF)
(|GroebnerPackage| . GB)
(|GroebnerFactorizationPackage| . GBF)
(|Guess| . GUESS)
(|GuessAlgebraicNumber| . GUESSAN)
(|GuessFinite| . GUESSF)
(|GuessFiniteFunctions| . GUESSF1)
(|GuessInteger| . GUESSINT)
(|GuessOption| . GOPT)
(|GuessPolynomial| . GUESSP)
(|GuessUnivariatePolynomial| . GUESSUP)
(|HallBasis| . HB)
(|Heap| . HEAP)
(|HexadecimalExpansion| . HEXADEC)
(|HTMLFormat| . HTMLFORM)
(|IdealDecompositionPackage| . IDECOMP)
(|IndexCard| . ICARD)
(|InfClsPt| . ICP)
(|InfiniteProductCharacteristicZero| . INFPROD0)
(|InfiniteProductFiniteField| . INPRODFF)
(|InfiniteProductPrimeField| . INPRODPF)
(|InfiniteTuple| . ITUPLE)
(|InfiniteTupleFunctions2| . ITFUN2)
(|InfiniteTupleFunctions3| . ITFUN3)
(|InfinitlyClosePoint| . INFCLSPT)
(|InfinitlyClosePointOverPseudoAlgebraicClosureOfFiniteField| . INFCLSPS)
(|Infinity| . INFINITY)
(|Integer| . INT)
(|IntegerCombinatoricFunctions| . COMBINAT)
(|IntegerLinearDependence| . ZLINDEP)
(|IntegerNumberTheoryFunctions| . INTHEORY)
(|IntegerPrimesPackage| . PRIMES)
(|IntegerRetractions| . INTRET)
(|IntegerRoots| . IROOT)
(|IntegrationResultFunctions2| . IR2)
(|IntegrationResultRFToFunction| . IRRF2F)
(|IntegrationResultToFunction| . IR2F)
(|InterfaceGroebnerPackage| . INTERGB)
(|InterpolateFormsPackage| . INTFRSP)
(|IntersectionDivisorPackage| . INTDIVP)
(|Interval| . INTRVL)
(|InventorDataSink| . IVDATA)
(|InventorViewPort| . IVVIEW)
(|InventorRenderPackage| . IVREND)
```

```
(|InverseLaplaceTransform| . INVLAPLA)
(|IrrRepSymNatPackage| . IRSN)
(|KernelFunctions2| . KERNEL2)
(|KeyedAccessFile| . KAFILE)
(|LaplaceTransform| . LAPLACE)
(|LazardMorenoSolvingPackage| . LAZM3PK)
(|Library| . LIB)
(|LieSquareMatrix| . LSQM)
(|LinearOrdinaryDifferentialOperator| . LODO)
(|LinearSystemMatrixPackage| . LSMP)
(|LinearSystemMatrixPackage1| . LSMP1)
(|LinearSystemFromPowerSeriesPackage| . LISYSER)
(|LinearSystemPolynomialPackage| . LSPP)
(|List| . LIST)
(|LinesOpPack| . LOP)
(|ListFunctions2| . LIST2)
(|ListFunctions3| . LIST3)
(|ListToMap| . LIST2MAP)
(|LocalParametrizationOfSimplePointPackage| . LPARSPT)
(|MakeFloatCompiledFunction| . MKFLCFN)
(|MakeFunction| . MKFUNC)
(|MakeRecord| . MKRECORD)
(|MappingPackage1| . MAPPKG1)
(|MappingPackage2| . MAPPKG2)
(|MappingPackage3| . MAPPKG3)
(|MappingPackage4| . MAPPKG4)
(|MathMLFormat| . MMLFORM)
(|Matrix| . MATRIX)
(|MatrixCategoryFunctions2| . MATCAT2)
(|MatrixCommonDenominator| . MCDEN)
(|MatrixLinearAlgebraFunctions| . MATLIN)
(|MatrixManipulation| . MAMA)
(|MergeThing| . MTHING)
(|ModularDistinctDegreeFactorizer| . MDDFACT)
(|ModuleOperator| . MODOP)
(|MonoidRingFunctions2| . MRF2)
(|MoreSystemCommands| . MSYSCMD)
(|MPolyCatFunctions2| . MPC2)
(|MPolyCatRationalFunctionFactorizer| . MPRFF)
(|Multiset| . MSET)
(|MultivariateFactorize| . MULTFACT)
(|MultivariatePolynomial| . MPOLY)
(|MultFiniteFactorize| . MFINFACT)
(|MyUnivariatePolynomial| . MYUP)
(|MyExpression| . MYEXPR)
(|NeitherSparseOrDensePowerSeries| . NSDPS)
(|NewtonPolygon| . NPOLYGON)
(|NoneFunctions1| . NONE1)
(|NonNegativeInteger| . NNI)
(|NottinghamGroup| . NOTTING)
(|NormalizationPackage| . NORMPK)
(|NormInMonogenicAlgebra| . NORMMA)
(|NumberTheoreticPolynomialFunctions| . NTPOLFN)
(|Numeric| . NUMERIC)
```

```
(|NumericalOrdinaryDifferentialEquations| . NUMODE)
(|NumericalQuadrature| . NUMQUAD)
(|NumericComplexEigenPackage| . NCEP)
(|NumericRealEigenPackage| . NREP)
(|NumericContinuedFraction| . NCNTFRAC)
(|Octonion| . OCT)
(|OctonionCategoryFunctions2| . OCTCT2)
(|OneDimensionalArray| . ARRAY1)
(|OneDimensionalArrayFunctions2| . ARRAY12)
(|OnePointCompletion| . ONECOMP)
(|OnePointCompletionFunctions2| . ONECOMP2)
(|OpenMathConnection| . OMCONN)
(|OpenMathDevice| . OMDEV)
(|OpenMathEncoding| . OMENC)
(|OpenMathError| . OMERR)
(|OpenMathErrorKind| . OMERRK)
(|OpenMathPackage| . OMPKG)
(|OpenMathServerPackage| . OMSERVER)
(|OperationsQuery| . OPQUERY)
(|OrderedCompletion| . ORDCOMP)
(|OrderedCompletionFunctions2| . ORDCOMP2)
(|OrdinaryDifferentialRing| . ODR)
(|OrdSetInts| . OSI)
(|OrthogonalPolynomialFunctions| . ORTHPOL)
(|OutputPackage| . OUT)
(|PackageForAlgebraicFunctionField| . PAFF)
(|PackageForAlgebraicFunctionFieldOverFiniteField| . PAFFFF)
(|PackageForPoly| . PFORP)
(|PadeApproximantPackage| . PADEPAC)
(|Palette| . PALETTE)
(|PartialFraction| . PFR)
(|PatternFunctions2| . PATTERN2)
(|ParametricPlaneCurve| . PARPCURV)
(|ParametricSpaceCurve| . PARSCURV)
(|ParametricSurface| . PARSURF)
(|ParametricPlaneCurveFunctions2| . PARPC2)
(|ParametricSpaceCurveFunctions2| . PARSC2)
(|ParametricSurfaceFunctions2| . PARSU2)
(|ParametrizationPackage| . PARAMP)
(|PartitionsAndPermutations| . PARTPERM)
(|PatternMatch| . PATMATCH)
(|PatternMatchAssertions| . PMASS)
(|PatternMatchResultFunctions2| . PATRES2)
(|PendantTree| . PENDTREE)
(|Permanent| . PERMAN)
(|PermutationGroupExamples| . PGE)
(|PermutationGroup| . PERMGRP)
(|Permutation| . PERM)
(|Pi| . HACKPI)
(|PiCoercions| . PICOERCE)
(|Places| . PLACES)
(|PlacesOverPseudoAlgebraicClosureOfFiniteField| . PLACESPS)
(|Plcs| . PLCS)
(|PointFunctions2| . PTFUNC2)
```

```
(|PolyGroebner| . PGROEB)
(|Polynomial| . POLY)
(|PolynomialAN2Expression| . PAN2EXPR)
(|PolynomialComposition| . PCOMP)
(|PolynomialDecomposition| . PDECOMP)
(|PolynomialFunctions2| . POLY2)
(|PolynomialIdeals| . IDEAL)
(|PolynomialPackageForCurve| . PLPKCRV)
(|PolynomialToUnivariatePolynomial| . POLY2UP)
(|PositiveInteger| . PI)
(|PowerSeriesLimitPackage| . LIMITPS)
(|PrimeField| . PF)
(|PrimitiveArrayFunctions2| . PRIMARR2)
(|PrintPackage| . PRINT)
(|ProjectiveAlgebraicSetPackage| . PRJALGPK)
(|ProjectivePlane| . PROJPL)
(|ProjectivePlaneOverPseudoAlgebraicClosureOfFiniteField| . PROJPLPS)
(|ProjectiveSpace| . PROJSP)
(|PseudoAlgebraicClosureOfAlgExtOfRationalNumber| . PACEXT)
(|QuadraticForm| . QFORM)
(|QuasiComponentPackage| . QCMPACK)
(|Quaternion| . QUAT)
(|QuaternionCategoryFunctions2| . QUATCT2)
(|QueryEquation| . QEQUAT)
(|Queue| . QUEUE)
(|QuotientFieldCategoryFunctions2| . QFCAT2)
(|RadicalEigenPackage| . REP)
(|RadicalSolvePackage| . SOLVERAD)
(|RadixExpansion| . RADIX)
(|RadixUtilities| . RADUTIL)
(|RandomNumberSource| . RANDSRC)
(|RationalFunction| . RF)
(|RationalFunctionDefiniteIntegration| . DEFINTRF)
(|RationalFunctionFactor| . RFFACT)
(|RationalFunctionFactorizer| . RFFACTOR)
(|RationalFunctionIntegration| . INTRF)
(|RationalFunctionLimitPackage| . LIMITRF)
(|RationalFunctionSign| . SIGNRF)
(|RationalFunctionSum| . SUMRF)
(|RationalRetractions| . RATRET)
(|RealClosure| . RECLOS)
(|RealPolynomialUtilitiesPackage| . POLUTIL)
(|RealZeroPackage| . REAL0)
(|RealZeroPackageQ| . REAL0Q)
(|RecurrenceOperator| . RECOP)
(|RectangularMatrixCategoryFunctions2| . RMCAT2)
(|RegularSetDecompositionPackage| . RSDCMPK)
(|RegularTriangularSet| . REGSET)
(|RegularTriangularSetGcdPackage| . RSETGCD)
(|RepresentationPackage1| . REP1)
(|RepresentationPackage2| . REP2)
(|ResolveLatticeCompletion| . RESLATC)
(|RewriteRule| . RULE)
(|RightOpenIntervalRootCharacterization| . ROIRC)
```

```
(|RomanNumeral| . ROMAN)
(|RootsFindingPackage| . RFP)
(|Ruleset| . RULESET)
(|ScriptFormulaFormat| . FORMULA)
(|ScriptFormulaFormat1| . FORMULA1)
(|Segment| . SEG)
(|SegmentBinding| . SEGBIND)
(|SegmentBindingFunctions2| . SEGBIND2)
(|SegmentFunctions2| . SEG2)
(|Set| . SET)
(|SimpleAlgebraicExtensionAlgFactor| . SAEFACT)
(|SimpleCell| . SCELL)
(|SimplifyAlgebraicNumberConvertPackage| . SIMPAN)
(|SingleInteger| . SINT)
(|SmithNormalForm| . SMITH)
(|SparseEchelonMatrix| . SEM)
(|SparseUnivariatePolynomialExpressions| . SUPEXPR)
(|SparseUnivariatePolynomialFunctions2| . SUP2)
(|SpecialOutputPackage| . SPECOUT)
(|SquareFreeRegularSetDecompositionPackage| . SRDCMPK)
(|SquareFreeRegularTriangularSet| . SREGSET)
(|SquareFreeRegularTriangularSetGcdPackage| . SFRGCD)
(|SquareFreeQuasiComponentPackage| . SFQCMPK)
(|Stack| . STACK)
(|Stream| . STREAM)
(|StreamFunctions1| . STREAM1)
(|StreamFunctions2| . STREAM2)
(|StreamFunctions3| . STREAM3)
(|StreamTensor| . STNSR)
(|StochasticDifferential| . SD)
(|String| . STRING)
(|SturmHabichtPackage| . SHP)
(|Symbol| . SYMBOL)
(|SymmetricGroupCombinatoricFunctions| . SGCF)
(|SystemSolvePackage| . SYSSOLP)
(|SAERationalFunctionAlgFactor| . SAERFFC)
(|Tableau| . TABLEAU)
(|TaylorSeries| . TS)
(|TaylorSolve| . UTSSOL)
(|TexFormat| . TEX)
(|TexFormat1| . TEX1)
(|TextFile| . TEXTFILE)
(|ThreeDimensionalViewport| . VIEW3D)
(|ThreeSpace| . SPACE3)
(|Timer| . TIMER)
(|TopLevelDrawFunctions| . DRAW)
(|TopLevelDrawFunctionsForAlgebraicCurves| . DRAWCURV)
(|TopLevelDrawFunctionsForCompiledFunctions| . DRAWCFUN)
(|TopLevelDrawFunctionsForPoints| . DRAWPT )
(|TopLevelThreeSpace| . TOPSP)
(|TranscendentalManipulations| . TRMANIP)
(|TransSolvePackage| . SOLVETRA)
(|Tree| . TREE)
(|TrigonometricManipulations| . TRIGMNIP)
```

```
  (|UnivariateLaurentSeriesFunctions2| . ULS2)
  (|UnivariateFormalPowerSeries| . UFPS)
  (|UnivariateFormalPowerSeriesFunctions| . UFPS1)
  (|UnivariatePolynomial| . UP)
  (|UnivariatePolynomialCategoryFunctions2| . UPOLYC2)
  (|UnivariatePolynomialCommonDenominator| . UPCDEN)
  (|UnivariatePolynomialFunctions2| . UP2)
  (|UnivariatePolynomialMultiplicationPackage| . UPMP)
  (|UnivariateTaylorSeriesCZero| . UTSZ)
  (|UnivariatePuiseuxSeriesFunctions2| . UPXS2)
  (|UnivariateTaylorSeriesFunctions2| . UTS2)
  (|UniversalSegment| . UNISEG)
  (|UniversalSegmentFunctions2| . UNISEG2)
  (|UserDefinedVariableOrdering| . UDVO)
  (|U8Matrix| . U8MAT)
  (|U16Matrix| . U16MAT)
  (|U32Matrix| . U32MAT)
  (|U8Vector| . U8VEC)
  (|U16Vector| . U16VEC)
  (|U32Vector| . U32VEC)
  (|U32VectorPolynomialOperations| . POLYVEC)
  (|Vector| . VECTOR)
  (|VectorFunctions2| . VECTOR2)
  (|ViewDefaultsPackage| . VIEWDEF)
  (|Void| . VOID)
  (|WuWenTsunTriangularSet| . WUTSET))
(|naglink|
  (|Asp1| . ASP1)
  (|Asp4| . ASP4)
  (|Asp6| . ASP6)
  (|Asp7| . ASP7)
  (|Asp8| . ASP8)
  (|Asp9| . ASP9)
  (|Asp10| . ASP10)
  (|Asp12| . ASP12)
  (|Asp19| . ASP19)
  (|Asp20| . ASP20)
  (|Asp24| . ASP24)
  (|Asp27| . ASP27)
  (|Asp28| . ASP28)
  (|Asp29| . ASP29)
  (|Asp30| . ASP30)
  (|Asp31| . ASP31)
  (|Asp33| . ASP33)
  (|Asp34| . ASP34)
  (|Asp35| . ASP35)
  (|Asp41| . ASP41)
  (|Asp42| . ASP42)
  (|Asp49| . ASP49)
  (|Asp50| . ASP50)
  (|Asp55| . ASP55)
  (|Asp73| . ASP73)
  (|Asp74| . ASP74)
  (|Asp77| . ASP77)
```

```
  (|Asp78| . ASP78)
  (|Asp80| . ASP80)
  (|FortranCode| . FC)
  (|FortranCodePackage1| . FCPAK1)
  (|FortranExpression| . FEXPR)
  (|FortranMachineTypeCategory| . FMTC)
  (|FortranMatrixCategory| . FMC)
  (|FortranMatrixFunctionCategory| . FMFUN)
  (|FortranOutputStackPackage| . FOP)
  (|FortranPackage| . FORT)
  (|FortranProgramCategory| . FORTCAT)
  (|FortranProgram| . FORTRAN)
  (|FortranFunctionCategory| . FORTFN)
  (|FortranScalarType| . FST)
  (|FortranType| . FT)
  (|FortranTemplate| . FTEM)
  (|FortranVectorFunctionCategory| . FVFUN)
  (|FortranVectorCategory| . FVC)
  (|MachineComplex| . MCMPLX)
  (|MachineFloat| . MFLOAT)
  (|MachineInteger| . MINT)
  (|MultiVariableCalculusFunctions| . MCALCFN)
  (|NagDiscreteFourierTransformInterfacePackage| . NAGDIS)
  (|NagEigenInterfacePackage| . NAGEIG)
  (|NAGLinkSupportPackage| . NAGSP)
  (|NagOptimisationInterfacePackage| . NAGOPT)
  (|NagQuadratureInterfacePackage| . NAGQUA)
  (|NagResultChecks| . NAGRES)
  (|NagSpecialFunctionsInterfacePackage| . NAGSPE)
  (|NagPolynomialRootsPackage| . NAGC02)
  (|NagRootFindingPackage| . NAGC05)
  (|NagSeriesSummationPackage| . NAGC06)
  (|NagIntegrationPackage| . NAGD01)
  (|NagOrdinaryDifferentialEquationsPackage| . NAGD02)
  (|NagPartialDifferentialEquationsPackage| . NAGD03)
  (|NagInterpolationPackage| . NAGE01)
  (|NagFittingPackage| . NAGE02)
  (|NagOptimisationPackage| . NAGE04)
  (|NagMatrixOperationsPackage| . NAGF01)
  (|NagEigenPackage| . NAGF02)
  (|NagLinearEquationSolvingPackage| . NAGF04)
  (|NagLapack| . NAGF07)
  (|NagSpecialFunctionsPackage| . NAGS)
  (|PackedHermitianSequence| . PACKED)
  (|Result| . RESULT)
  (|SimpleFortranProgram| . SFORT)
  (|Switch| . SWITCH)
  (|SymbolTable| . SYMTAB)
  (|TemplateUtilities| . TEMUTL)
  (|TheSymbolTable| . SYMS)
  (|ThreeDimensionalMatrix| . M3D))
 (|anna|
  (|AnnaNumericalIntegrationPackage| . INTPACK)
  (|AnnaNumericalOptimizationPackage| . OPTPACK)
```

```
  (|AnnaOrdinaryDifferentialEquationPackage| . ODEPACK)
  (|AnnaPartialDifferentialEquationPackage| . PDEPACK)
  (|AttributeButtons| . ATTRBUT)
  (|BasicFunctions| . BFUNCT)
  (|d01ajfAnnaType| . D01AJFA)
  (|d01akfAnnaType| . D01AKFA)
  (|d01alfAnnaType| . D01ALFA)
  (|d01amfAnnaType| . D01AMFA)
  (|d01anfAnnaType| . D01ANFA)
  (|d01apfAnnaType| . D01APFA)
  (|d01aqfAnnaType| . D01AQFA)
  (|d01asfAnnaType| . D01ASFA)
  (|d01fcfAnnaType| . D01FCFA)
  (|d01gbfAnnaType| . D01GBFA)
  (|d01AgentsPackage| . D01AGNT)
  (|d01TransformFunctionType| . D01TRNS)
  (|d01WeightsPackage| . D01WGTS)
  (|d02AgentsPackage| . D02AGNT)
  (|d02bbfAnnaType| . D02BBFA)
  (|d02bhfAnnaType| . D02BHFA)
  (|d02cjfAnnaType| . D02CJFA)
  (|d02ejfAnnaType| . D02EJFA)
  (|d03AgentsPackage| . D03AGNT)
  (|d03eefAnnaType| . D03EEFA)
  (|d03fafAnnaType| . D03FAFA)
  (|e04AgentsPackage| . E04AGNT)
  (|e04dgfAnnaType| . E04DGFA)
  (|e04fdfAnnaType| . E04FDFA)
  (|e04gcfAnnaType| . E04GCFA)
  (|e04jafAnnaType| . E04JAFA)
  (|e04mbfAnnaType| . E04MBFA)
  (|e04nafAnnaType| . E04NAFA)
  (|e04ucfAnnaType| . E04UCFA)
  (|ExpertSystemContinuityPackage| . ESCONT)
  (|ExpertSystemContinuityPackage1| . ESCONT1)
  (|ExpertSystemToolsPackage| . ESTOOLS)
  (|ExpertSystemToolsPackage1| . ESTOOLS1)
  (|ExpertSystemToolsPackage2| . ESTOOLS2)
  (|NumericalIntegrationCategory| . NUMINT)
  (|NumericalIntegrationProblem| . NIPROB)
  (|NumericalODEProblem| . ODEPROB)
  (|NumericalOptimizationCategory| . OPTCAT)
  (|NumericalOptimizationProblem| . OPTPROB)
  (|NumericalPDEProblem| . PDEPROB)
  (|ODEIntensityFunctionsTable| . ODEIFTBL)
  (|IntegrationFunctionsTable| . INTFTBL)
  (|OrdinaryDifferentialEquationsSolverCategory| . ODECAT)
  (|PartialDifferentialEquationsSolverCategory| . PDECAT)
  (|RoutinesTable| . ROUTINE))
 (|categories|
  (|AbelianGroup| . ABELGRP)
  (|AbelianMonoid| . ABELMON)
  (|AbelianMonoidRing| . AMR)
  (|AbelianSemiGroup| . ABELSG)
```

```
(|AdditiveValuationAttribute| . ATADDVA)
(|AffineSpaceCategory| . AFSPCAT)
(|Aggregate| . AGG)
(|Algebra| . ALGEBRA)
(|AlgebraicallyClosedField| . ACF)
(|AlgebraicallyClosedFunctionSpace| . ACFS)
(|ApproximateAttribute| . ATAPPRO)
(|ArbitraryExponentAttribute| . ATARBEX)
(|ArbitraryPrecisionAttribute| . ATARBPR)
(|ArcHyperbolicFunctionCategory| . AHYP)
(|ArcTrigonometricFunctionCategory| . ATRIG)
(|AssociationListAggregate| . ALAGG)
(|AttributeRegistry| . ATTREG)
(|BagAggregate| . BGAGG)
(|BasicType| . BASTYPE)
(|BiModule| . BMODULE)
(|BinaryRecursiveAggregate| . BRAGG)
(|BinaryTreeCategory| . BTCAT)
(|BitAggregate| . BTAGG)
(|BlowUpMethodCategory| . BLMETCT)
(|CachableSet| . CACHSET)
(|CancellationAbelianMonoid| . CABMON)
(|CanonicalAttribute| . ATCANON)
(|CanonicalClosedAttribute| . ATCANCL)
(|CanonicalUnitNormalAttribute| . ATCUNOR)
(|CentralAttribute| . ATCENRL)
(|CharacteristicNonZero| . CHARNZ)
(|CharacteristicZero| . CHARZ)
(|CoercibleTo| . KOERCE)
(|Collection| . CLAGG)
(|CombinatorialFunctionCategory| . CFCAT)
(|CombinatorialOpsCategory| . COMBOPC)
(|CommutativeRing| . COMRING)
(|CommutativeStarAttribute| . ATCS)
(|Comparable| . COMPAR)
(|ComplexCategory| . COMPCAT)
(|ConvertibleTo| . KONVERT)
(|DequeueAggregate| . DQAGG)
(|DesingTreeCategory| . DSTRCAT)
(|Dictionary| . DIAGG)
(|DictionaryOperations| . DIOPS)
(|DifferentialExtension| . DIFEXT)
(|DifferentialPolynomialCategory| . DPOLCAT)
(|DifferentialRing| . DIFRING)
(|DifferentialVariableCategory| . DVARCAT)
(|DirectProductCategory| . DIRPCAT)
(|DivisionRing| . DIVRING)
(|DivisorCategory| . DIVCAT)
(|DoublyLinkedAggregate| . DLAGG)
(|ElementaryFunctionCategory| . ELEMFUN)
(|Eltable| . ELTAB)
(|EltableAggregate| . ELTAGG)
(|EntireRing| . ENTIRER)
(|EuclideanDomain| . EUCDOM)
```

```
(|Evalable| . EVALAB)
(|ExpressionSpace| . ES)
(|ExtensibleLinearAggregate| . ELAGG)
(|ExtensionField| . XF)
(|Field| . FIELD)
(|FieldOfPrimeCharacteristic| . FPC)
(|Finite| . FINITE)
(|FileCategory| . FILECAT)
(|FileNameCategory| . FNCAT)
(|FiniteAbelianMonoidRing| . FAMR)
(|FiniteAggregateAttribute| . ATFINAG)
(|FiniteAlgebraicExtensionField| . FAXF)
(|FiniteDivisorCategory| . FDIVCAT)
(|FiniteFieldCategory| . FFIELDC)
(|FiniteLinearAggregate| . FLAGG)
(|FiniteRankNonAssociativeAlgebra| . FINAALG)
(|FiniteRankAlgebra| . FINRALG)
(|FiniteSetAggregate| . FSAGG)
(|FloatingPointSystem| . FPS)
(|FramedAlgebra| . FRAMALG)
(|FramedNonAssociativeAlgebra| . FRNAALG)
(|FramedNonAssociativeAlgebraFunctions2| . FRNAAF2)
(|FreeAbelianMonoidCategory| . FAMONC)
(|FreeLieAlgebra| . FLALG)
(|FreeModuleCat| . FMCAT)
(|FullyEvalableOver| . FEVALAB)
(|FullyLinearlyExplicitRingOver| . FLINEXP)
(|FullyPatternMatchable| . FPATMAB)
(|FullyRetractableTo| . FRETRCT)
(|FunctionFieldCategory| . FFCAT)
(|FunctionSpace| . FS)
(|GcdDomain| . GCDDOM)
(|GradedAlgebra| . GRALG)
(|GradedModule| . GRMOD)
(|Group| . GROUP)
(|HomogeneousAggregate| . HOAGG)
(|HyperbolicFunctionCategory| . HYPCAT)
(|IndexedAggregate| . IXAGG)
(|IndexedDirectProductCategory| . IDPC)
(|InfinitlyClosePointCategory| . INFCLCT)
(|InnerEvalable| . IEVALAB)
(|IntegerNumberSystem| . INS)
(|IntegralDomain| . INTDOM)
(|IntervalCategory| . INTCAT)
(|KeyedDictionary| . KDAGG)
(|JacobiIdentityAttribute| . ATJACID)
(|LazyRepresentativeAttribute| . ATLR)
(|LazyStreamAggregate| . LZSTAGG)
(|LeftAlgebra| . LALG)
(|LeftModule| . LMODULE)
(|LeftOreRing| . LORER)
(|LeftUnitaryAttribute| . ATLUNIT)
(|LieAlgebra| . LIECAT)
(|LinearAggregate| . LNAGG)
```

```
(|LinearlyExplicitRingOver| . LINEXP)
(|LinearOrdinaryDifferentialOperatorCategory| . LODOCAT)
(|LiouvillianFunctionCategory| . LFCAT)
(|ListAggregate| . LSAGG)
(|LocalPowerSeriesCategory| . LOCPOWC)
(|Logic| . LOGIC)
(|ModularAlgebraicGcdOperations| . MAGCDOC)
(|MatrixCategory| . MATCAT)
(|Module| . MODULE)
(|Monad| . MONAD)
(|MonadWithUnit| . MONADWU)
(|Monoid| . MONOID)
(|MonogenicAlgebra| . MONOGEN)
(|MonogenicLinearOperator| . MLO)
(|MultiDictionary| . MDAGG)
(|MultiplicativeValuationAttribute| . ATMULVA)
(|MultisetAggregate| . MSETAGG)
(|MultivariateTaylorSeriesCategory| . MTSCAT)
(|NonAssociativeAlgebra| . NAALG)
(|NonAssociativeRing| . NASRING)
(|NonAssociativeRng| . NARNG)
(|NormalizedTriangularSetCategory| . NTSCAT)
(|NotherianAttribute| . ATNOTHR)
(|NullSquareAttribute| . ATNULSQ)
(|NoZeroDivisorsAttribute| . ATNZDIV)
(|Object| . OBJECT)
(|OctonionCategory| . OC)
(|OneDimensionalArrayAggregate| . A1AGG)
(|OpenMath| . OM)
(|OrderedAbelianGroup| . OAGROUP)
(|OrderedAbelianMonoid| . OAMON)
(|OrderedAbelianMonoidSup| . OAMONS)
(|OrderedAbelianSemiGroup| . OASGP)
(|OrderedCancellationAbelianMonoid| . OCAMON)
(|OrderedFinite| . ORDFIN)
(|OrderedIntegralDomain| . OINTDOM)
(|OrderedMonoid| . ORDMON)
(|OrderedMultisetAggregate| . OMSAGG)
(|OrderedRing| . ORDRING)
(|OrderedSet| . ORDSET)
(|PAdicIntegerCategory| . PADICCT)
(|PartialDifferentialRing| . PDRING)
(|PartiallyOrderedSetAttribute| . ATPOSET)
(|PartialTranscendentalFunctions| . PTRANFN)
(|Patternable| . PATAB)
(|PatternMatchable| . PATMAB)
(|PermutationCategory| . PERMCAT)
(|PlacesCategory| . PLACESC)
(|PlottablePlaneCurveCategory| . PPCURVE)
(|PlottableSpaceCurveCategory| . PSCURVE)
(|PointCategory| . PTCAT)
(|PolynomialCategory| . POLYCAT)
(|PolynomialFactorizationExplicit| . PFECAT)
(|PolynomialSetCategory| . PSETCAT)
```

```
(|PowerSeriesCategory| . PSCAT)
(|PrimitiveFunctionCategory| . PRIMCAT)
(|PrincipalIdealDomain| . PID)
(|PriorityQueueAggregate| . PRQAGG)
(|ProjectiveSpaceCategory| . PRSPCAT)
(|PseudoAlgebraicClosureofAlgExtOfRationalNumberCategory| . PACEXTC)
(|PseudoAlgebraicClosureOfFiniteField| . PACOFF)
(|PseudoAlgebraicClosureOfFiniteFieldCategory| . PACFFC)
(|PseudoAlgebraicClosureOfPerfectFieldCategory| . PACPERC)
(|PseudoAlgebraicClosureOfRationalNumber| . PACRAT)
(|PseudoAlgebraicClosureOfRationalNumberCategory| . PACRATC)
(|QuaternionCategory| . QUATCAT)
(|QueueAggregate| . QUAGG)
(|QuotientFieldCategory| . QFCAT)
(|RadicalCategory| . RADCAT)
(|RealClosedField| . RCFIELD)
(|RealConstant| . REAL)
(|RealNumberSystem| . RNS)
(|RealRootCharacterizationCategory| . RRCC)
(|RectangularMatrixCategory| . RMATCAT)
(|RecursiveAggregate| . RCAGG)
(|RecursivePolynomialCategory| . RPOLCAT)
(|RegularChain| . RGCHAIN)
(|RegularTriangularSetCategory| . RSETCAT)
(|RetractableTo| . RETRACT)
(|RightModule| . RMODULE)
(|Ring| . RING)
(|RightUnitaryAttribute| . ATRUNIT)
(|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)
(|TriangularSetCategory| . TSETCAT)
(|TrigonometricFunctionCategory| . TRIGCAT)
(|TwoDimensionalArrayCategory| . ARR2CAT)
(|Type| . TYPE)
```

```
  (|UnaryRecursiveAggregate| . URAGG)
  (|UniqueFactorizationDomain| . UFD)
  (|UnitsKnownAttribute| . ATUNIKN)
  (|UnivariateLaurentSeriesCategory| . ULSCAT)
  (|UnivariateLaurentSeriesConstructorCategory| . ULSCCAT)
  (|UnivariatePolynomialCategory| . UPOLYC)
  (|UnivariatePowerSeriesCategory| . UPSCAT)
  (|UnivariatePuiseuxSeriesCategory| . UPXSCAT)
  (|UnivariatePuiseuxSeriesConstructorCategory| . UPXSCCA)
  (|UnivariateSkewPolynomialCategory| . OREPCAT)
  (|UnivariateTaylorSeriesCategory| . UTSCAT)
  (|VectorCategory| . VECTCAT)
  (|VectorSpace| . VSPACE)
  (|XAlgebra| . XALG)
  (|XFreeAlgebra| . XFALG)
  (|XPolynomialsCat| . XPOLYC)
  (|ZeroDimensionalSolvePackage| . ZDSOLVE))
 (|Hidden|
  (|AlgebraicFunction| . AF)
  (|AlgebraicFunctionField| . ALGFF)
  (|AlgebraicHermiteIntegration| . INTHERAL)
  (|AlgebraicIntegrate| . INTALG)
  (|AlgebraicIntegration| . INTAF)
  (|AnonymousFunction| . ANON)
  (|AntiSymm| . ANTISYM)
  (|ApplyRules| . APPRULE)
  (|ApplyUnivariateSkewPolynomial| . APPLYORE)
  (|ArrayStack| . ASTACK)
  (|AssociatedEquations| . ASSOCEQ)
  (|AssociationList| . ALIST)
  (|Automorphism| . AUTOMOR)
  (|BalancedFactorisation| . BALFACT)
  (|BalancedPAdicInteger| . BPADIC)
  (|BalancedPAdicRational| . BPADICRT)
  (|BezoutMatrix| . BEZOUT)
  (|BoundIntegerRoots| . BOUNDZRO)
  (|BrillhartTests| . BRILL)
  (|ChangeOfVariable| . CHVAR)
  (|CharacteristicPolynomialInMonogenicalAlgebra| . CPIMA)
  (|ChineseRemainderToolsForIntegralBases| . IBACHIN)
  (|CoerceVectorMatrixPackage| . CVMP)
  (|CombinatorialFunction| . COMBF)
  (|CommonOperators| . COMMONOP)
  (|CommuteUnivariatePolynomialCategory| . COMMUPC)
  (|ComplexIntegerSolveLinearPolynomialEquation| . CINTSLPE)
  (|ComplexPattern| . COMPLPAT)
  (|ComplexPatternMatch| . CPMATCH)
  (|ComplexRootFindingPackage| . CRFP)
  (|ConstantLODE| . ODECONST)
  (|CyclicStreamTools| . CSTTOOLS)
  (|CyclotomicPolynomialPackage| . CYCLOTOM)
  (|DefiniteIntegrationTools| . DFINTTLS)
  (|DegreeReductionPackage| . DEGRED)
  (|DeRhamComplex| . DERHAM)
```

```
(|DifferentialSparseMultivariatePolynomial| . DSMP)
(|DirectProduct| . DIRPROD)
(|DirectProductMatrixModule| . DPMM)
(|DirectProductModule| . DPMO)
(|DiscreteLogarithmPackage| . DLP)
(|DistributedMultivariatePolynomial| . DMP)
(|DoubleResultantPackage| . DBLRESP)
(|DrawOptionFunctions0| . DROPT0)
(|DrawOptionFunctions1| . DROPT1)
(|ElementaryFunction| . EF)
(|ElementaryFunctionsUnivariateLaurentSeries| . EFULS)
(|ElementaryFunctionsUnivariatePuiseuxSeries| . EFUPXS)
(|ElementaryIntegration| . INTEF)
(|ElementaryRischDE| . RDEEF)
(|ElementaryRischDESystem| . RDEEFS)
(|EllipticFunctionsUnivariateTaylorSeries| . ELFUTS)
(|EqTable| . EQTBL)
(|EuclideanModularRing| . EMR)
(|EvaluateCycleIndicators| . EVALCYC)
(|ExponentialExpansion| . EXPEXPAN)
(|ExponentialOfUnivariatePuiseuxSeries| . EXPUPXS)
(|ExpressionSpaceFunctions1| . ES1)
(|ExpressionTubePlot| . EXPRTUBE)
(|ExtAlgBasis| . EAB)
(|FactoredFunctions| . FACTFUNC)
(|FactoredFunctionUtilities| . FRUTIL)
(|FactoringUtilities| . FACUTIL)
(|FGLMIfCanPackage| . FGLMICPK)
(|FindOrderFinite| . FORDER)
(|FiniteDivisor| . FDIV)
(|FiniteFieldCyclicGroupExtension| . FFCGX)
(|FiniteFieldCyclicGroupExtensionByPolynomial| . FFCGP)
(|FiniteFieldExtension| . FFX)
(|FiniteFieldExtensionByPolynomial| . FFP)
(|FiniteFieldFunctions| . FFF)
(|FiniteFieldNormalBasisExtension| . FFNBX)
(|FiniteFieldNormalBasisExtensionByPolynomial| . FFNBP)
(|FiniteFieldPolynomialPackage| . FFPOLY)
(|FiniteFieldSolveLinearPolynomialEquation| . FFSLPE)
(|FormalFraction| . FORMAL)
(|FourierComponent| . FCOMP)
(|FractionalIdeal| . FRIDEAL)
(|FramedModule| . FRMOD)
(|FreeAbelianGroup| . FAGROUP)
(|FreeAbelianMonoid| . FAMONOID)
(|FreeGroup| . FGROUP)
(|FreeModule| . FM)
(|FreeModule1| . FM1)
(|FreeMonoid| . FMONOID)
(|FunctionalSpecialFunction| . FSPECF)
(|FunctionCalled| . FUNCTION)
(|FunctionFieldIntegralBasis| . FFINTBAS)
(|FunctionSpaceReduce| . FSRED)
(|FunctionSpaceToUnivariatePowerSeries| . FS2UPS)
```

```
(|FunctionSpaceToExponentialExpansion| . FS2EXPXP)
(|FunctionSpaceUnivariatePolynomialFactor| . FSUPFACT)
(|GaloisGroupFactorizationUtilities| . GALFACTU)
(|GaloisGroupFactorizer| . GALFACT)
(|GaloisGroupPolynomialUtilities| . GALPOLYU)
(|GaloisGroupUtilities| . GALUTIL)
(|GeneralHenselPackage| . GHENSEL)
(|GeneralDistributedMultivariatePolynomial| . GDMP)
(|GeneralPolynomialGcdPackage| . GENPGCD)
(|GeneralSparseTable| . GSTBL)
(|GenericNonAssociativeAlgebra| . GCNAALG)
(|GenExEuclid| . GENEEZ)
(|GeneralizedMultivariateFactorize| . GENMFACT)
(|GeneralModulePolynomial| . GMODPOL)
(|GeneralPolynomialSet| . GPOLSET)
(|GeneralTriangularSet| . GTSET)
(|GenUFactorize| . GENUFACT)
(|GenusZeroIntegration| . INTG0)
(|GosperSummationMethod| . GOSPER)
(|GraphImage| . GRIMAGE)
(|GrayCode| . GRAY)
(|GroebnerInternalPackage| . GBINTERN)
(|GroebnerSolve| . GROEBSOL)
(|GuessOptionFunctions0| . GOPT0)
(|HashTable| . HASHTBL)
(|Heap| . HEAP)
(|HeuGcd| . HEUGCD)
(|HomogeneousDistributedMultivariatePolynomial| . HDMP)
(|HyperellipticFiniteDivisor| . HELLFDIV)
(|IncrementingMaps| . INCRMAPS)
(|IndexedBits| . IBITS)
(|IndexedDirectProductAbelianGroup| . IDPAG)
(|IndexedDirectProductAbelianMonoid| . IDPAM)
(|IndexedDirectProductObject| . IDPO)
(|IndexedDirectProductOrderedAbelianMonoid| . IDPOAM)
(|IndexedDirectProductOrderedAbelianMonoidSup| . IDPOAMS)
(|IndexedExponents| . INDE)
(|IndexedFlexibleArray| . IFARRAY)
(|IndexedList| . ILIST)
(|IndexedMatrix| . IMATRIX)
(|IndexedOneDimensionalArray| . IARRAY1)
(|IndexedString| . ISTRING)
(|IndexedTwoDimensionalArray| . IARRAY2)
(|IndexedVector| . IVECTOR)
(|InnerAlgFactor| . IALGFACT)
(|InnerAlgebraicNumber| . IAN)
(|InnerCommonDenominator| . ICDEN)
(|InnerFiniteField| . IFF)
(|InnerFreeAbelianMonoid| . IFAMON)
(|InnerIndexedTwoDimensionalArray| . IIARRAY2)
(|InnerMatrixLinearAlgebraFunctions| . IMATLIN)
(|InnerMatrixQuotientFieldFunctions| . IMATQF)
(|InnerModularGcd| . INMODGCD)
(|InnerMultFact| . INNMFACT)
```

```
(|InnerNormalBasisFieldFunctions| . INBFF)
(|InnerNumericEigenPackage| . INEP)
(|InnerNumericFloatSolvePackage| . INFSP)
(|InnerPAdicInteger| . IPADIC)
(|InnerPolySign| . INPSIGN)
(|InnerPolySum| . ISUMP)
(|InnerPrimeField| . IPF)
(|InnerSparseUnivariatePowerSeries| . ISUPS)
(|InnerTable| . INTABL)
(|InnerTaylorSeries| . ITAYLOR)
(|InnerTrigonometricManipulations| . ITRIGMNP)
(|InputForm| . INFORM)
(|InputFormFunctions1| . INFORM1)
(|IntegerBits| . INTBIT)
(|IntegerFactorizationPackage| . INTFACT)
(|IntegerMod| . ZMOD)
(|IntegerSolveLinearPolynomialEquation| . INTSLPE)
(|IntegralBasisPolynomialTools| . IBPTOOLS)
(|IntegralBasisTools| . IBATOOL)
(|IntegrationResult| . IR)
(|IntegrationTools| . INTTOOLS)
(|InternalPrintPackage| . IPRNTPK)
(|InternalRationalUnivariateRepresentationPackage| . IRURPK)
(|IrredPolyOverFiniteField| . IRREDFFX)
(|Kernel| . KERNEL)
(|Kovacic| . KOVACIC)
(|LaurentPolynomial| . LAUPOL)
(|LeadingCoefDetermination| . LEADCDET)
(|LexTriangularPackage| . LEXTRIPK)
(|LieExponentials| . LEXP)
(|LiePolynomial| . LPOLY)
(|LinearDependence| . LINDEP)
(|LinearOrdinaryDifferentialOperatorFactorizer| . LODOF)
(|LinearOrdinaryDifferentialOperator1| . LODO1)
(|LinearOrdinaryDifferentialOperator2| . LODO2)
(|LinearOrdinaryDifferentialOperatorsOps| . LODOOPS)
(|LinearPolynomialEquationByFractions| . LPEFRAC)
(|LinGroebnerPackage| . LGROBP)
(|LiouvillianFunction| . LF)
(|ListMonoidOps| . LMOPS)
(|ListMultiDictionary| . LMDICT)
(|LocalAlgebra| . LA)
(|Localize| . LO)
(|LyndonWord| . LWORD)
(|Magma| . MAGMA)
(|MakeBinaryCompiledFunction| . MKBCFUNC)
(|MakeCachableSet| . MKCHSET)
(|MakeUnaryCompiledFunction| . MKUCFUNC)
(|MappingPackageInternalHacks1| . MAPHACK1)
(|MappingPackageInternalHacks2| . MAPHACK2)
(|MappingPackageInternalHacks3| . MAPHACK3)
(|MeshCreationRoutinesForThreeDimensions| . MESH)
(|ModMonic| . MODMON)
(|ModularField| . MODFIELD)
```

```
(|ModularHermitianRowReduction| . MHROWRED)
(|ModularRing| . MODRING)
(|ModuleMonomial| . MODMONOM)
(|MoebiusTransform| . MOEBIUS)
(|MonoidRing| . MRING)
(|MonomialExtensionTools| . MONOTOOL)
(|MPolyCatPolyFactorizer| . MPCPF)
(|MPolyCatFunctions3| . MPC3)
(|MRationalFactorize| . MRATFAC)
(|MultipleMap| . MMAP)
(|MultivariateLifting| . MLIFT)
(|MultivariateSquareFree| . MULTSQFR)
(|HomogeneousDirectProduct| . HDP)
(|NewSparseMultivariatePolynomial| . NSMP)
(|NewSparseUnivariatePolynomial| . NSUP)
(|NewSparseUnivariatePolynomialFunctions2| . NSUP2)
(|NonCommutativeOperatorDivision| . NCODIV)
(|NewtonInterpolation| . NEWTON)
(|None| . NONE)
(|NonLinearFirstOrderODESolver| . NODE1)
(|NonLinearSolvePackage| . NLINSOL)
(|NormRetractPackage| . NORMRETR)
(|NPCoef| . NPCOEF)
(|NumberFormats| . NUMFMT)
(|NumberFieldIntegralBasis| . NFINTBAS)
(|NumericTubePlot| . NUMTUBE)
(|ODEIntegration| . ODEINT)
(|ODETools| . ODETOOLS)
(|Operator| . OP)
(|OppositeMonogenicLinearOperator| . OMLO)
(|OrderedDirectProduct| . ODP)
(|OrderedFreeMonoid| . OFMONOID)
(|OrderedVariableList| . OVAR)
(|OrderingFunctions| . ORDFUNS)
(|OrderlyDifferentialPolynomial| . ODPOL)
(|OrderlyDifferentialVariable| . ODVAR)
(|OrdinaryWeightedPolynomials| . OWP)
(|OutputForm| . OUTFORM)
(|PadeApproximants| . PADE)
(|PAdicInteger| . PADIC)
(|PAdicRational| . PADICRAT)
(|PAdicRationalConstructor| . PADICRC)
(|PAdicWildFunctionFieldIntegralBasis| . PWFFINTB)
(|ParadoxicalCombinatorsForStreams| . YSTREAM)
(|ParametricLinearEquations| . PLEQN)
(|PartialFractionPackage| . PFRPAC)
(|Partition| . PRTITION)
(|Pattern| . PATTERN)
(|PatternFunctions1| . PATTERN1)
(|PatternMatchFunctionSpace| . PMFS)
(|PatternMatchIntegerNumberSystem| . PMINS)
(|PatternMatchIntegration| . INTPM)
(|PatternMatchKernel| . PMKERNEL)
(|PatternMatchListAggregate| . PMLSAGG)
```

```
(|PatternMatchListResult| . PATLRES)
(|PatternMatchPolynomialCategory| . PMPLCAT)
(|PatternMatchPushDown| . PMDOWN)
(|PatternMatchQuotientFieldCategory| . PMQFCAT)
(|PatternMatchResult| . PATRES)
(|PatternMatchSymbol| . PMSYM)
(|PatternMatchTools| . PMTOOLS)
(|PlaneAlgebraicCurvePlot| . ACPLOT)
(|Plot| . PLOT)
(|PlotFunctions1| . PLOT1)
(|PlotTools| . PLOTTOOL)
(|Plot3D| . PLOT3D)
(|PoincareBirkhoffWittLyndonBasis| . PBWLB)
(|Point| . POINT)
(|PointsOfFiniteOrder| . PFO)
(|PointsOfFiniteOrderRational| . PFOQ)
(|PointsOfFiniteOrderTools| . PFOTOOLS)
(|PointPackage| . PTPACK)
(|PolToPol| . POLTOPOL)
(|PolynomialCategoryLifting| . POLYLIFT)
(|PolynomialCategoryQuotientFunctions| . POLYCATQ)
(|PolynomialFactorizationByRecursion| . PFBR)
(|PolynomialFactorizationByRecursionUnivariate| . PFBRU)
(|PolynomialGcdPackage| . PGCD)
(|PolynomialInterpolation| . PINTERP)
(|PolynomialInterpolationAlgorithms| . PINTERPA)
(|PolynomialNumberTheoryFunctions| . PNTHEORY)
(|PolynomialRing| . PR)
(|PolynomialRoots| . POLYROOT)
(|PolynomialSetUtilitiesPackage| . PSETPK)
(|PolynomialSolveByFormulas| . SOLVEFOR)
(|PolynomialSquareFree| . PSQFR)
(|PrecomputedAssociatedEquations| . PREASSOC)
(|PrimitiveArray| . PRIMARR)
(|PrimitiveElement| . PRIMELT)
(|PrimitiveRatDE| . ODEPRIM)
(|PrimitiveRatRicDE| . ODEPRRIC)
(|Product| . PRODUCT)
(|PseudoRemainderSequence| . PRS)
(|PseudoLinearNormalForm| . PSEUDLIN)
(|PureAlgebraicIntegration| . INTPAF)
(|PureAlgebraicLODE| . ODEPAL)
(|PushVariables| . PUSHVAR)
(|QuasiAlgebraicSet| . QALGSET)
(|QuasiAlgebraicSet2| . QALGSET2)
(|RadicalFunctionField| . RADFF)
(|RandomDistributions| . RDIST)
(|RandomFloatDistributions| . RFDIST)
(|RandomIntegerDistributions| . RIDIST)
(|RationalFactorize| . RATFACT)
(|RationalIntegration| . INTRAT)
(|RationalInterpolation| . RINTERP)
(|RationalLODE| . ODERAT)
(|RationalRicDE| . ODERTRIC)
```

```
(|RationalUnivariateRepresentationPackage| . RURPK)
(|RealSolvePackage| . REALSOLV)
(|RectangularMatrix| . RMATRIX)
(|ReducedDivisor| . RDIV)
(|ReduceLODE| . ODERED)
(|ReductionOfOrder| . REDORDER)
(|Reference| . REF)
(|RepeatedDoubling| . REPDB)
(|RepeatedSquaring| . REPSQ)
(|ResidueRing| . RESRING)
(|RetractSolvePackage| . RETSOL)
(|RuleCalled| . RULECOLD)
(|SetOfMIntegersInOneToN| . SETMN)
(|SExpression| . SEX)
(|SExpressionOf| . SEXOF)
(|SequentialDifferentialPolynomial| . SDPOL)
(|SequentialDifferentialVariable| . SDVAR)
(|SimpleAlgebraicExtension| . SAE)
(|SingletonAsOrderedSet| . SAOS)
(|SortedCache| . SCACHE)
(|SortPackage| . SORTPAK)
(|SparseMultivariatePolynomial| . SMP)
(|SparseMultivariateTaylorSeries| . SMTS)
(|SparseTable| . STBL)
(|SparseUnivariatePolynomial| . SUP)
(|SparseUnivariateSkewPolynomial| . ORESUP)
(|SparseUnivariateLaurentSeries| . SULS)
(|SparseUnivariatePuiseuxSeries| . SUPXS)
(|SparseUnivariateTaylorSeries| . SUTS)
(|SplitHomogeneousDirectProduct| . SHDP)
(|SplittingNode| . SPLNODE)
(|SplittingTree| . SPLTREE)
(|SquareMatrix| . SQMATRIX)
(|Stack| . STACK)
(|StorageEfficientMatrixOperations| . MATSTOR)
(|StreamInfiniteProduct| . STINPROD)
(|StreamTaylorSeriesOperations| . STTAYLOR)
(|StreamTranscendentalFunctions| . STTF)
(|StreamTranscendentalFunctionsNonCommutative| . STTFNC)
(|StringTable| . STRTBL)
(|SubResultantPackage| . SUBRESP)
(|SubSpace| . SUBSPACE)
(|SubSpaceComponentProperty| . COMPPROP)
(|SuchThat| . SUCH)
(|SupFractionFactorizer| . SUPFRACF)
(|SymmetricFunctions| . SYMFUNC)
(|SymmetricPolynomial| . SYMPOLY)
(|SystemODESolver| . ODESYS)
(|Table| . TABLE)
(|TableauxBumpers| . TABLBUMP)
(|TabulatedComputationPackage| . TBCMPPK)
(|TangentExpansions| . TANEXP)
(|ToolsForSign| . TOOLSIGN)
(|TranscendentalHermiteIntegration| . INTHERTR)
```

```
  (|TranscendentalIntegration| . INTTR)
  (|TranscendentalRischDE| . RDETR)
  (|TranscendentalRischDESystem| . RDETRS)
  (|TransSolvePackageService| . SOLVESER)
  (|TriangularMatrixOperations| . TRIMAT)
  (|TubePlot| . TUBE)
  (|TubePlotTools| . TUBETOOL)
  (|Tuple| . TUPLE)
  (|TwoDimensionalArray| . ARRAY2)
  (|TwoDimensionalPlotClipping| . CLIP)
  (|TwoDimensionalViewport| . VIEW2D)
  (|TwoFactorize| . TWOFACT)
  (|UnivariateFactorize| . UNIFACT)
  (|UnivariateLaurentSeries| . ULS)
  (|UnivariateLaurentSeriesConstructor| . ULSCONS)
  (|UnivariatePolynomialDecompositionPackage| . UPDECOMP)
  (|UnivariatePolynomialDivisionPackage| . UPDIVP)
  (|UnivariatePolynomialSquareFree| . UPSQFREE)
  (|UnivariatePuiseuxSeries| . UPXS)
  (|UnivariatePuiseuxSeriesConstructor| . UPXSCONS)
  (|UnivariatePuiseuxSeriesWithExponentialSingularity| . UPXSSING)
  (|UnivariateSkewPolynomial| . OREUP)
  (|UnivariateSkewPolynomialCategoryOps| . OREPCTO)
  (|UnivariateTaylorSeries| . UTS)
  (|UnivariateTaylorSeriesODESolver| . UTSODE)
  (|UserDefinedPartialOrdering| . UDPO)
  (|UTSodetools| . UTSODETL)
  (|Variable| . VARIABLE)
  (|ViewportPackage| . VIEW)
  (|WeierstrassPreparation| . WEIER)
  (|WeightedPolynomials| . WP)
  (|WildFunctionFieldIntegralBasis| . WFFINTBS)
  (|XDistributedPolynomial| . XDPOLY)
  (|XExponentialPackage| . XEXPPKG)
  (|XPBWPolynomial| . XPBWPOLY)
  (|XPolynomial| . XPOLY)
  (|XPolynomialRing| . XPR)
  (|XRecursivePolynomial| . XRPOLY))
 (|defaults|
  (|AbelianGroup&| . ABELGRP-)
  (|AbelianMonoid&| . ABELMON-)
  (|AbelianMonoidRing&| . AMR-)
  (|AbelianSemiGroup&| . ABELSG-)
  (|Aggregate&| . AGG-)
  (|Algebra&| . ALGEBRA-)
  (|AlgebraicallyClosedField&| . ACF-)
  (|AlgebraicallyClosedFunctionSpace&| . ACFS-)
  (|ArcTrigonometricFunctionCategory&| . ATRIG-)
  (|BagAggregate&| . BGAGG-)
  (|BasicType&| . BASTYPE-)
  (|BinaryRecursiveAggregate&| . BRAGG-)
  (|BinaryTreeCategory&| . BTCAT-)
  (|BitAggregate&| . BTAGG-)
  (|Collection&| . CLAGG-)
```

```
(|ComplexCategory&| . COMPCAT-)
(|Dictionary&| . DIAGG-)
(|DictionaryOperations&| . DIOPS-)
(|DifferentialExtension&| . DIFEXT-)
(|DifferentialPolynomialCategory&| . DPOLCAT-)
(|DifferentialRing&| . DIFRING-)
(|DifferentialVariableCategory&| . DVARCAT-)
(|DirectProductCategory&| . DIRPCAT-)
(|DivisionRing&| . DIVRING-)
(|ElementaryFunctionCategory&| . ELEMFUN-)
(|EltableAggregate&| . ELTAGG-)
(|EuclideanDomain&| . EUCDOM-)
(|Evalable&| . EVALAB-)
(|ExpressionSpace&| . ES-)
(|ExtensibleLinearAggregate&| . ELAGG-)
(|ExtensionField&| . XF-)
(|Field&| . FIELD-)
(|FieldOfPrimeCharacteristic&| . FPC-)
(|FiniteAbelianMonoidRing&| . FAMR-)
(|FiniteAlgebraicExtensionField&| . FAXF-)
(|FiniteDivisorCategory&| . FDIVCAT-)
(|FiniteFieldCategory&| . FFIELDC-)
(|FiniteLinearAggregate&| . FLAGG-)
(|FiniteSetAggregate&| . FSAGG-)
(|FiniteRankAlgebra&| . FINRALG-)
(|FiniteRankNonAssociativeAlgebra&| . FINAALG-)
(|FloatingPointSystem&| . FPS-)
(|FramedAlgebra&| . FRAMALG-)
(|FramedNonAssociativeAlgebra&| . FRNAALG-)
(|FullyEvalableOver&| . FEVALAB-)
(|FullyLinearlyExplicitRingOver&| . FLINEXP-)
(|FullyRetractableTo&| . FRETRCT-)
(|FunctionFieldCategory&| . FFCAT-)
(|FunctionSpace&| . FS-)
(|GcdDomain&| . GCDDOM-)
(|GradedAlgebra&| . GRALG-)
(|GradedModule&| . GRMOD-)
(|Group&| . GROUP-)
(|HomogeneousAggregate&| . HOAGG-)
(|HyperbolicFunctionCategory&| . HYPCAT-)
(|IndexedAggregate&| . IXAGG-)
(|InnerEvalable&| . IEVALAB-)
(|IntegerNumberSystem&| . INS-)
(|IntegralDomain&| . INTDOM-)
(|KeyedDictionary&| . KDAGG-)
(|LazyStreamAggregate&| . LZSTAGG-)
(|LeftAlgebra&| . LALG-)
(|LieAlgebra&| . LIECAT-)
(|LinearAggregate&| . LNAGG-)
(|ListAggregate&| . LSAGG-)
(|Logic&| . LOGIC-)
(|LinearOrdinaryDifferentialOperatorCategory&| . LODOCAT-)
(|MatrixCategory&| . MATCAT-)
(|Module&| . MODULE-)
```

```
(|Monad&| . MONAD-)
(|MonadWithUnit&| . MONADWU-)
(|Monoid&| . MONOID-)
(|MonogenicAlgebra&| . MONOGEN-)
(|NonAssociativeAlgebra&| . NAALG-)
(|NonAssociativeRing&| . NASRING-)
(|NonAssociativeRng&| . NARNG-)
(|OctonionCategory&| . OC-)
(|OneDimensionalArrayAggregate&| . A1AGG-)
(|OrderedRing&| . ORDRING-)
(|OrderedSet&| . ORDSET-)
(|PartialDifferentialRing&| . PDRING-)
(|PolynomialCategory&| . POLYCAT-)
(|PolynomialFactorizationExplicit&| . PFECAT-)
(|PolynomialSetCategory&| . PSETCAT-)
(|PowerSeriesCategory&| . PSCAT-)
(|QuaternionCategory&| . QUATCAT-)
(|QuotientFieldCategory&| . QFCAT-)
(|RadicalCategory&| . RADCAT-)
(|RealClosedField&| . RCFIELD-)
(|RealNumberSystem&| . RNS-)
(|RealRootCharacterizationCategory&| . RRCC-)
(|RectangularMatrixCategory&| . RMATCAT-)
(|RecursiveAggregate&| . RCAGG-)
(|RecursivePolynomialCategory&| . RPOLCAT-)
(|RegularTriangularSetCategory&| . RSETCAT-)
(|RetractableTo&| . RETRACT-)
(|Ring&| . RING-)
(|SemiGroup&| . SGROUP-)
(|SetAggregate&| . SETAGG-)
(|SetCategory&| . SETCAT-)
(|SquareMatrixCategory&| . SMATCAT-)
(|StreamAggregate&| . STAGG-)
(|StringAggregate&| . SRAGG-)
(|TableAggregate&| . TBAGG-)
(|TranscendentalFunctionCategory&| . TRANFUN-)
(|TriangularSetCategory&| . TSETCAT-)
(|TrigonometricFunctionCategory&| . TRIGCAT-)
(|TwoDimensionalArrayCategory&| . ARR2CAT-)
(|UnaryRecursiveAggregate&| . URAGG-)
(|UniqueFactorizationDomain&| . UFD-)
(|UnivariateLaurentSeriesConstructorCategory&| . ULSCCAT-)
(|UnivariatePolynomialCategory&| . UPOLYC-)
(|UnivariatePowerSeriesCategory&| . UPSCAT-)
(|UnivariatePuiseuxSeriesConstructorCategory&| . UPXSCCA-)
(|UnivariateSkewPolynomialCategory&| . OREPCAT-)
(|UnivariateTaylorSeriesCategory&| . UTSCAT-)
(|VectorCategory&| . VECTCAT-)
(|VectorSpace&| . VSPACE-))))
```

———-

Chapter 9

The global variables

9.0.4 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:

9.0.5 defvar creditlist

— **initvars** —

```
(defvar creditlist '(
"An alphabetical listing of contributors to AXIOM:"
"Michael Albaugh        Cyril Alberga          Roy Adler"
"Christian Aistleitner  Richard Anderson       George Andrews"
"Jerry Archibald        S.J. Atkins            Jeremy Avigad"
"Henry Baker            Martin Baker           Stephen Balzac"
"Yurij Baransky         David R. Barton        Thomas Baruchel"
"Gerald Baumgartner     Gilbert Baumslag       Michael Becker"
"Nelson H. F. Beebe     Jay Belanger           David Bindel"
"Fred Blair             Vladimir Bondarenko    Mark Botch"
"Raoul Bourquin         Alexandre Bouyer       Karen Braman"
"Wolfgang Brehm         Peter A. Broadbery     Martin Brock"
"Manuel Bronstein       Christopher Brown      Stephen Buchwald"
"Florian Bundschuh      Luanne Burns           William Burge"
"Ralph Byers            Quentin Carpent        Pierre Casteran"
"Robert Cavines         Bruce Char             Ondrej Certik"
"Tzu-Yi Chen            Bobby Cheng            Cheekai Chin"
"David V. Chudnovsky    Gregory V. Chudnovsky  Mark Clements"
"Roland Coeurjoly       James Cloos            Jia Zhao Cong"
"Josh Cohen             Christophe Conil       Don Coppersmith"
"George Corliss         Robert Corless         Gary Cornell"
"Meino Cramer           Karl Crary             Jeremy Du Croz"
"David Cyganski         Nathaniel Daly         Timothy Daly Sr."
"Timothy Daly Jr.       James H. Davenport     David Day"
```

```
"James Demmel            Didier Deshommes       Michael Dewar"
"Inderjit Dhillon        Jack Dongarra          Jean Della Dora"
"Gabriel Dos Reis        Claire DiCrescendo     Sam Dooley"
"Nicolas James Doye      Zlatko Drmac           Lionel Ducos"
"Iain Duff               Lee Duhem              Martin Dunstan"
"Brian Dupee             Dominique Duval        Robert Edwards"
"Hans-Dieter Ehrich      Heow Eide-Goodman      Lars Erickson"
"Mark Fahey              Richard Fateman        Bertfried Fauser"
"Stuart Feldman          John Fletcher          Brian Ford"
"Albrecht Fortenbacher   George Frances         Constantine Frangos"
"Timothy Freeman         Korrinn Fu             Marc Gaetano"
"Rudiger Gebauer         Van de Geijn           Kathy Gerber"
"Patricia Gianni         Gustavo Goertkin       Samantha Goldrich"
"Holger Gollan           Teresa Gomez-Diaz      Laureano Gonzalez-Vega"
"Stephen Gortler         Johannes Grabmeier     Matt Grayson"
"Klaus Ebbe Grue         James Griesmer         Vladimir Grinberg"
"Oswald Gschnitzer       Ming Gu                Jocelyn Guidry"
"Gaetan Hache            Steve Hague            Satoshi Hamaguchi"
"Sven Hammarling         Mike Hansen            Richard Hanson"
"Richard Harke           Bill Hart              Vilya Harvey"
"Martin Hassner          Arthur S. Hathaway     Dan Hatton"
"Waldek Hebisch          Karl Hegbloom          Ralf Hemmecke"
"Henderson               Antoine Hersen         Nicholas J. Higham"
"Hoon Hong               Roger House            Gernot Hueber"
"Pietro Iglio            Alejandro Jakubi       Richard Jenks"
"Bo Kagstrom             William Kahan          Kyriakos Kalorkoti"
"Kai Kaminski            Grant Keady            Wilfrid Kendall"
"Tony Kennedy            David Kincaid          Keshav Kini"

"Ted Kosan               Paul Kosinski          Igor Kozachenko"
"Fred Krogh              Klaus Kusche           Bernhard Kutzler"
"Tim Lahey               Larry Lambe            Kaj Laurson"
"Charles Lawson          George L. Legendre     Franz Lehner"
"Frederic Lehobey        Michel Levaud          Howard Levy"
"J. Lewis                Ren-Cang Li            Rudiger Loos"
"Craig Lucas             Michael Lucks          Richard Luczak"
"Camm Maguire            Francois Maltey        William Martin"
"Osni Marques            Alasdair McAndrew      Bob McElrath"
"Michael McGettrick      Edi Meier              Ian Meikle"
"David Mentre            Jonathan Millen        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"
"Frank Pfenning          Jose Alfredo Portes    E. Quintana-Orti"
"Gregorio Quintana-Orti  Beresford Parlett      A. Petitet"
"Andre Platzer           Peter Poromaas         Claude Quitte"
"Arthur C. Ralfs         Norman Ramsey          Anatoly Raportirenko"
"Guilherme Reis          Huan Ren               Albert D. Rich"
```

```
"Michael Richardson     Jason Riedy             Renaud Rioboo"
"Jean Rivlin            Nicolas Robidoux        Simon Robinson"
"Raymond Rogers         Michael Rothstein       Martin Rubey"
"Jeff Rutter            Philip Santas           David Saunders"
"Alfred Scheerhorn      William Schelter        Gerhard Schneider"
"Martin Schoenert       Marshall Schor          Frithjof Schulze"
"Fritz Schwarz          Steven Segletes         V. Sima"
"Nick Simicich          William Sit             Elena Smirnova"
"Jacob Nyffeler Smith   Matthieu Sozeau         Ken Stanley"
"Jonathan Steinbach     Fabio Stumbo            Christine Sundaresan"
"Klaus Sutner           Robert Sutor            Moss E. Sweedler"
"Eugene Surowitz        Yong Kiam Tan           Max Tegmark"
"T. Doug Telford        James Thatcher          Laurent Thery"
"Balbir Thomas          Mike Thomas             Dylan Thurston"
"Francoise Tisseur      Steve Toleque           Raymond Toy"
"Barry Trager           Themos T. Tsikas        Gregory Vanuxem"
"Kresimir Veselic       Christof Voemel         Bernhard Wall"
"Stephen Watt           Andreas Weber           Jaap Weel"
"Juergen Weiss          M. Weller               Mark Wegman"
"James Wen              Thorsten Werther        Michael Wester"
"R. Clint Whaley        James T. Wheeler        John M. Wiley"
"Berhard Will           Clifton J. Williamson   Stephen Wilson"
"Shmuel Winograd        Robert Wisbauer         Sandra Wityak"
"Waldemar Wiwianka      Knut Wolf               Yanyang Xiao"
"Liu Xiaojun            Clifford Yapp           David Yun"
"Qian Yun               Vadim Zhytnikov         Richard Zippel"
"Evelyn Zoernack        Bruno Zuercher          Dan Zwillinger"))
```

———-

The `$current-directory` variable is set to the current directory at startup. This is used by the `)cd` function and some of the compile routines. This is the result of the (p296) get-current-directory function. This variable is used to set `*default-pathname-defaults*`. The (p299) reroot function resets it to `$spadroot`.

An example of a runtime value is:

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

9.0.6 defvar $current-directory

— **initvars** —

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

———-

The `$directory-list` is a runtime list of absolute pathnames. This list is generated by (p299) reroot from the list of relative paths held in the variable `$relative-directory-list`. Each entry will be prefixed by `$spadroot`.

An example of a runtime value is:

```
$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/")
```

9.0.7 defvar $directory-list

— **initvars** —

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

———-

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

See the function "makeInitialModemapFrame" (12.3.15 p 296).

An example of a runtime value is:

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

9.0.8 defvar $InitialModemapFrame

— **initvars** —

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

———-

The `$library-directory-list` variable is the system-wide search path for library files. (p299) reroot prepends the `$spadroot` variable to the `$relative-library-directory-list` variable.

An example of a runtime value is:

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

9.0.9 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
```

9.0.10 defvar $msgDatabaseName

— **initvars** —

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

———-

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

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

See the function "openserver" (33.0.23 p 1049).

An example of a runtime value is:

```
$openServerIfTrue = nil
```

9.0.11 defvar $openServerIfTrue

— **initvars** —

```
(defvar $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 (p299) reroot function to generate the absolute list of paths held in the variable `$directory-list`. Each entry will be prefixed by `$spadroot`.

An example of a runtime value is:

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

9.0.12 defvar $relative-directory-list

— **initvars** —

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

```
    "/doc/spadhelp/" ))
```

———-

The `$relative-library-directory-list` is a hand-generated list of directories containing algebra. The (p299) reroot function will prefix every path in this list with the value of the `$spadroot` variable to construct the `$library-directory-list` variable.

An example of a runtime value is:

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

9.0.13 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 (p295) initroot function uses this variable if the AXIOM shell variable is not set. The (p296) 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"
```

9.0.14 defvar $spadroot

— **initvars** —

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

———-

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

See the function "openserver" (33.0.23 p 1049).

An example of a runtime value is:

```
$SpadServer = nil
```

9.0.15 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” (33.0.23 p 1049).

An example of a runtime value is:

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

9.0.16 defvar $SpadServerName

— **initvars** —

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

———-

Chapter 10

Starting Axiom

This chapter details the internal processing behind an Axiom console session where the user types "1" and gets a result.

```
axiom -nox
                        AXIOM Computer Algebra System
                        Version: Axiom (August 2014)
             Timestamp: Friday September 12, 2014 at 06:24:14
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
   Visit http://axiom-developer.org for more information
------------------------------------------------------------------------------

   Re-reading interp.daase
   Re-reading operation.daase
   Re-reading category.daase
   Re-reading browse.daase
(1) ->
(1) -> 1

   (1)  1
                                                          Type: PositiveInteger
(2) ->
```

By working through this example we introduce, motivate, and explain how the interpreter works, where and why functions are called, how the system transitions from input strings to algebra, how the databases are used, and more.

If you plan to maintain or modify the interpreter this information is necessary. If you really want to know how Axiom works, this information is useful.

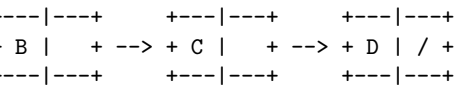
Each function call we describe has a link to the actual function so you can read the detailed code and see why it reacts as it does to the given input.

I've taken the liberty of adding comments that show the function signature. Some of the types only exist as unnamed data structures in the interpreter (e.g. "Server", which is really just a small integer). They are introduced without definition simply as a documentation aid

but may sometimes be defined a Common Lisp deftypes for performance reasons.

A Note on Common Lisp Circular Notation

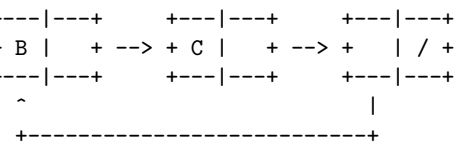
You may not be familiar with circular notation in Common Lisp. If a list contains a pointer back to itself or a sublist then the output would be an infinite stream. In order to prevent this the circular notation is used. So for a list X,



which is the list (A . (B . (C . (D . ())))). The printing rule says that if a period is followed by a parenthesis then both are suppressed. So this would print as (A B C D). But it could be that we execute

```
(rplaca (last X) (cdr X))
```

so the list now is



and now the list X is cicular. This prints as

```
(A . #0=(B C #0#))
```

As you can see the `#0=` introduces a unique label for the cons cell pointed at by (CDR A). We stored that address in the CAR of the last node. So the last node in the list uses the previously defined label with the notation `#0#`.

Circular notation is used extensively in Axiom since a lot of the structures are shared or self-referential. You have to be careful because, as a result of structure sharing, changing something in one place can change an apparently unrelated structure by side-effect.

Axiom starts by invoking a function value of the lisp symbol `*top-level-hook*` which is normally unbound. The normal function invocation path is:

```
axiom -nox

lisp
 -> restart
 -> |spad|
 -> |runspad|
 -> |ncTopLevel|
 -> |ncIntLoop|
 -> |intloop|
 -> |SpadInterpretStream|
 ->   mkprompt                  -- outputs "(1) ->" to the console
 ->   |intloopReadConsole|      -- the Read-Eval-Print loop function
 ->     |serverReadLine|        -- does the actual read to the console
 -> process the input and recursively call |intloopReadConsole|
```

SpadInterpretStream(p289) is called with a third arguments, **interactive?** set to **t** so it sets up an interactive loop to read from the console. The other two arguments are ignored on the main interpreter path.

SpadInterpretStream(p289) can also be called by the compiler, with the **interactive?** argument **nil** to read from a file. See bookvol9.

mkprompt(p301) puts one of several kinds of prompts on the screen. In the default case we include the step number. The return value is not used.

The **intloopReadConsole**(p290) function does tail-recursive calls to itself and never exits. It is the primary Read-Eval-Print-Loop (REPL).

intloopReadConsole(p290) reads the next line and calls one of three kinds of processors

1. **intnplisp**(p296) to handle)lisp input
2. **ncloopCommand**(p727) to handle)command input
3. **intloopProcessString**(p297) to handle everything else

There are only two ways out of the REPL, either using the command "**)fin**" which drops into lisp or closing the *standard-input* stream. If dropped into lisp, the top level loop can be restarted by calling **(restart)**.

intloopReadConsole takes 2 arguments. The first is a String **prefix** which is usually an empty string but might contain prior lines that ended with an underscore, the Axiom continuation character. The second is an Integer which will be the step number printed at the prompt.

10.1 An Overview of a Simple Input

Here we walk through details of Axiom's default behavior when handling a simple input, the number 1. Many details are skipped in order to provide a simple overview of the interpreter operation. Further details can be found at the specific functions.

Axiom is in **intloopReadConsole**(p290), the Read-Eval-Print-Loop (REPL) function and the user types "1".

```
  1> (|intloopReadConsole| "" 1)
       ; serverReadLine : Stream -> String
    2> (|serverReadLine| #<synonym stream to *TERMINAL-IO*>)
         ; is-console : Stream -> Boolean
      3> (IS-CONSOLE #<synonym stream to *TERMINAL-IO*>)
      <3 (IS-CONSOLE T)
         ; sockSendInt : (Purpose,Command) -> Integer
         ; Purpose 1 is SessionManager, Command 3 is EndOfOutput
         ; A return of 0 indicates success.
         ; see the socket types purpose list in bookvol7, chunk include/com.h
      3> (|sockSendInt| 1 3)
      <3 (|sockSendInt| 0)
         ; serverSwitch : Void -> Integer
         ; see server_switch in sockio.c
         ; this multiplexes the socket connection among front ends
         ; CallInterp is the constant 4 (see the table in sockio-c)
         ; CallInterp simply returns to the interpreter
      3> (|serverSwitch|)
1
      <3 (|serverSwitch| 4)
         ; the action for CallInterp is to call read-line
```

```
        ; read-line is defined in vmlisp.lisp
      3> (|read-line| #<synonym stream to *TERMINAL-IO*>)
      <3 (|read-line| "1" NIL)
    <2 (|serverReadLine| "1")
```

Axiom calls **serverReadLine**(p304) to read the integer from the console. First it calls **is-console** (bookvol9) to check that the console stream exists.

sockSendInt (see sockio.lisp, sockio-c.c) sends on socket 1 (**SessionManager**) a 3, meaning **EndOfOutput**, i.e. a newline.

serverSwitch (see sockio-c in bookvol7) multitasks among the different sockets and finds the interpreter socket is available, returning 4 (**CallInterp**) (see sockio-c commands sent table and bookvol8).

serverReadLine(p304) has a cond switch for action **$CallInterp**. In that case it calls **read-line** (see vmlisp.lisp) to read the input line and returns the result, in this case, the string "1".

```
    2> (|intloopPrefix?| ")fi" "1")
    <2 (|intloopPrefix?| NIL)
    2> (|intloopPrefix?| ")" "1")
    <2 (|intloopPrefix?| NIL)
    2> (CONCAT "" "1")
    <2 (CONCAT "1")
    2> (|ncloopEscaped| "1")
    <2 (|ncloopEscaped| NIL)
```

intloopReadConsole(p290) checks for various possible special kinds of input. Axiom returned a non-zero length string. Before processing it we need to check for the "**)fin**" command, which fails. We need to check for a leading "**)**", meaning it is some kind of command input, which fails. We might have an existing string in the **prefix** argument so we concatentate it to the input. The **prefix** might contain text from a previous continued line. Next we check whether the input line has a trailing underscore, meaning an Axiom line is being continued, and if so, we recurse in order to read the next line.

intloopPrefix?(p295) which will return NIL if there is no match of the prefix characters, otherwise it returns the string without any leading blanks.

None of these special cases occur with the input "1". Axiom calls **intloopProcessString**(p297) which calls **setCurrentLine**(p301) to add the input line to the history which is stored in **$currentLine**.

```
    2> (|intloopProcessString| "1" 1)
      3> (|setCurrentLine| "1")
      <3 (|setCurrentLine| ("1"))
```

· · · all the magic happens here · · ·

· · · and then **intloopProcessString** will eventually return the new step number 2. Then Axiom puts up a prompt and waits for further input.

```
    <2 (|intloopProcessString| 2)
    2> (MKPROMPT)
      3> (CONCAT "(" "2" ") -> ")
      <3 (CONCAT "(2) -> ")
    <2 (MKPROMPT "(2) -> ")
(2) ->
```

```
2> (|serverReadLine| #<synonym stream to *TERMINAL-IO*>)
  3> (IS-CONSOLE #<synonym stream to *TERMINAL-IO*>)
  <3 (IS-CONSOLE T)
  3> (|sockSendInt| 1 3)
  <3 (|sockSendInt| 0)
  3> (|serverSwitch|)
```

Now Axiom is ready for the next input.

10.2 Parsing the input

We now examine the magic portion above which has several phases. The first phase constructs a data structure called a Delay. This data structure is the core data structure of the "zipper" parser.

The "zipper" parser is unique to Axiom. It was invented by Bill Burge who did research in recursive techniques, including parsing. For insight, see his article on Stream Procesing Functions [Burg74].

10.2.1 Creating a Delay – incString

The **intloopProcessString**(p297) has the nested function call

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

which according to lisp semantics is processed inside out. First we examine the call to **incString**(p298) which is passed the input string "1".

The **incString**(p298) function gets the string from Axiom's input line, in this case "1" and constructs a set of nested function calls to process the input line.

```
3> (|incString| "1")
```

The **incString**(p298) function calls **Delay**(p356) which changes the function call into a simple list object prefixed by the symbol tag **nonnullstream**.

```
4> (|incLude| 0 ("1") 0 ("strings") (1))
  5> (|Delay| |incLude1|  (0 ("1") 0 ("strings") (1)))
  <5 (|Delay| (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1)))
<4 (|incLude| (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1)))
```

That result is passed to **incRenumber**(p329), which calls **incIgen**(p330) which returns a **Delay**(p356). It then calls **incZip**(p330) to "zips" together the function **incRenumberLine**(p331) and the two delays into a single delay. This gets put into a delay with **incZip1**(p330) as the function.

```
4> (|incRenumber|
       (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1)))
  5> (|incIgen| 0)
    6> (|Delay| |incIgen1| (0))
    <6 (|Delay| (|nonnullstream| |incIgen1| 0))
  <5 (|incIgen| (|nonnullstream| |incIgen1| 0))
```

```
    5> (|incZip| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
          (|nonnullstream| |incIgen1| 0))
      6> (|Delay| |incZip1| |incRenumberLine|)
      <6 (|Delay|
           (|nonnullstream| |incZip1| |incRenumberLine|
             (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
             (|nonnullstream| |incIgen1| 0)))
    <5 (|incZip|
         (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
           (|nonnullstream| |incIgen1| 0)))

  <4 (|incRenumber|
       (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0)))

<3 (|incString|
     (|nonnullstream| |incZip1| |incRenumberLine|
       (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
       (|nonnullstream| |incIgen1| 0)))
```

We are building a stream of functions and arguments stored in a delay structure which will eventually be evaluated. We continue this process with the call to **next**(p298) which builds a delay with the function **next1**(p298) and the current delay.

10.2.2 Creating a Delay – next

```
3> (|next| |lineoftoks|
     (|nonnullstream| |incZip1| |incRenumberLine|
       (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
       (|nonnullstream| |incIgen1| 0)))
  4> (|Delay| |next1|
       (|lineoftoks|
         (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
           (|nonnullstream| |incIgen1| 0))))
  <4 (|Delay| (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0))))
<3 (|next|
     (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0))))
```

10.2.3 Creating a Delay – ncloopParse

'We continue building a larger delay, this time with a call to **next**(p298) with the function argument **ncloopParse**(p297) and the existing delay.

```
3> (|next| |ncloopParse|
     (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0))))
  4> (|Delay| #0=|next1|
       (|ncloopParse|
         (|nonnullstream| #0# |lineoftoks|
           (|nonnullstream| |incZip1| |incRenumberLine|
             (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
             (|nonnullstream| |incIgen1| 0)))))
  <4 (|Delay|
       (|nonnullstream| #0=|next1| |ncloopParse|
         (|nonnullstream| #0# |lineoftoks|
           (|nonnullstream| |incZip1| |incRenumberLine|
             (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
             (|nonnullstream| |incIgen1| 0)))))
<3 (|next|
     (|nonnullstream| #0=|next1| |ncloopParse|
       (|nonnullstream| #0# |lineoftoks|
         (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
           (|nonnullstream| |incIgen1| 0)))))
```

Finally we call **intloopProcess**(p321) with the step number **stepno**, whether we are talking to the console **interactive** and the delay we just constructed **delay**

10.2.4 Evaluating a Delay – intloopProcess

At this point we have created a large delay. Now we begin to evaluate it.

```
3> (|intloopProcess| 1 T
    (|nonnullstream| #0=|next1| |ncloopParse|
      (|nonnullstream| #0# |lineoftoks|
        (|nonnullstream| |incZip1| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
          (|nonnullstream| |incIgen1| 0)))))
```

intloopProcess(p321) calls **StreamNull**(p555) which walks the delay applying the second value, which is a function, to the rest of the delay. Thus, all of the functions we packaged into the delay will be evaluated.

The result of each function call, e.g the result of calling **next1**(p298) will be a pair, which we call a ParsePair . The car of the ParsePair is rplaca'd into the delay and the cdr of the ParsePair is rplacd'd into the delay. So the delay is gradually reduced by each function call.

```
  4> (|StreamNull|
       (|nonnullstream| #0=|next1| |ncloopParse|
         (|nonnullstream| #0# |lineoftoks|
           (|nonnullstream| |incZip1| |incRenumberLine|
```

```
        (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
        (|nonnullstream| |incIgen1| 0)))))
```

Here we see the **next1**(p298) function being called from the delay. It immediately calls **StreamNull**(p555) to process the rest of the delay.

```
5> (|next1| |ncloopParse|
     (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0))))
  6> (|StreamNull|
       (|nonnullstream| |next1| |lineoftoks|
         (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
           (|nonnullstream| |incIgen1| 0))))
```

StreamNull(p555), now working on the inner portion of the delay, finds the function **next1**(p298) and calls it, which results in an immediate inner call to **StreamNull**(p555).

```
7> (|next1| |lineoftoks|
     (|nonnullstream| |incZip1| |incRenumberLine|
       (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
       (|nonnullstream| |incIgen1| 0)))
  8> (|StreamNull|
       (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0)))
```

Descending even further, the **StreamNull**(p555) finds **incZip1**(p330), which finds the function **incRenumberLine**(p331) and two delays.

```
9> (|incZip1|
     |incRenumberLine|
     (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
     (|nonnullstream| |incIgen1| 0))
```

incZip1(p330) invokes **StreamNull**(p555) on the first delay, which invokes **incLude1**(p336) on the rest of the delay.

```
10> (|StreamNull|
      (|nonnullstream| |incLude1| 0 ("1") 0
                                    ("strings") (1)))
```

incLude1(p336) unpacks the argument list and invokes **StreamNull**(p555) on the second argument `("1")` which is not the expected symbol **nonnullstream** so **StreamNull**(p555) immediately returns NIL.

```
11> (|incLude1| 0 ("1") 0 ("strings") (1))
12> (|StreamNull| ("1"))
<12 (|StreamNull| NIL)
```

Next, **incLude1**(p336) calls **incClassify**(p353) to which calls **incCommand?**(p353) which checks for a leading ")". Since there isn't one **incClassify**(p353) immediately returns a list of NIL, 0, and the empty string.

```
12> (|incClassify| "1")
13> (|incCommand?| "1")
<13 (|incCommand?| NIL)
```

```
<12 (|incClassify| (NIL 0 ""))

12> (|Skipping?| 1)
13> (|KeepPart?| 1)
<13 (|KeepPart?| T)
<12 (|Skipping?| NIL)

12> (|xlOK| 0 "1" 1 "strings")
13> (|xlOK1| 0 "1" "1" 1 "strings")
14> (INCLINE1 0 "1" "1" -1 1 "strings")
15> (|lnCreate| 0 "1" -1 1 "strings")
<15 (|lnCreate| (0 "1" -1 1 "strings"))
<14 (INCLINE1 (((0 "1" -1 1 "strings") . 1) . "1"))
<13 (|xlOK1| ((((0 "1" -1 1 "strings") . 1) . "1")
              (NIL |none|)))
<12 (|xlOK| ((((0 "1" -1 1 "strings") . 1) . "1")
             (NIL |none|)))

12> (|incLude| 0 NIL 1 ("strings") (1))
13> (|Delay| |incLude1| (0 NIL 1 ("strings") (1)))
<13 (|Delay|
      (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1)))
<12 (|incLude|
      (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1)))

<11 (|incLude1|
      (((((0 "1" -1 1 "strings") . 1) . "1") (NIL |none|))
      |nonnullstream| |incLude1| 0 NIL 1 ("strings") (1)))
<10 (|StreamNull| NIL)

10> (|StreamNull| (|nonnullstream| |incIgen1| 0))
11> (|incIgen1| 0)
12> (|incIgen| 1)
13> (|Delay| |incIgen1| (1))
<13 (|Delay| (|nonnullstream| |incIgen1| 1))
<12 (|incIgen| (|nonnullstream| |incIgen1| 1))
<11 (|incIgen1| (1 |nonnullstream| |incIgen1| 1))
<10 (|StreamNull| NIL)
10> (|incRenumberLine|
     ((((0 "1" -1 1 "strings") . 1) . "1") (NIL |none|)) 1)
11> (|incRenumberItem|
     (((0 "1" -1 1 "strings") . 1) . "1") 1)
12> (|lnSetGlobalNum| (0 "1" -1 1 "strings") 1)
<12 (|lnSetGlobalNum| 1)
<11 (|incRenumberItem| (((0 "1" 1 1 "strings") . 1) . "1"))
11> (|incHandleMessage|
      ((((0 "1" 1 1 "strings") . 1) . "1") (NIL |none|)))
<11 (|incHandleMessage| 0)
<10 (|incRenumberLine|
      (((0 "1" 1 1 "strings") . 1) . "1"))

10> (|incZip| |incRenumberLine|
      (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
      (|nonnullstream| |incIgen1| 1))
11> (|Delay| |incZip1|
```

```
          (|incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1)))
    <11 (|Delay|
          (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1)))
    <10 (|incZip|
         (|nonnullstream| |incZip1| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
          (|nonnullstream| |incIgen1| 1)))

  <9 (|incZip1|
      ((((0 "1" 1 1 "strings") . 1) . "1")
        |nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1)))
<8 (|StreamNull| NIL)

8> (|lineoftoks|
    ((((0 "1" 1 1 "strings") . 1) . "1")
      |nonnullstream| |incZip1| |incRenumberLine|
      (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
      (|nonnullstream| |incIgen1| 1)))
  9> (|nextline|
      ((((0 "1" 1 1 "strings") . 1) . "1")
        |nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1)))
    10> (|npNull|
         ((((0 "1" 1 1 "strings") . 1) . "1")
          |nonnullstream| |incZip1| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
          (|nonnullstream| |incIgen1| 1)))
    11> (|StreamNull|
         ((((0 "1" 1 1 "strings") . 1) . "1")
          |nonnullstream| |incZip1| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
          (|nonnullstream| |incIgen1| 1)))
    <11 (|StreamNull| NIL)
    <10 (|npNull| NIL)
    10> (STRPOSL " " "1" 0 T)
    <10 (STRPOSL 0)
  <9 (|nextline| T)
  9> (|scanIgnoreLine| "1" 0)
  <9 (|scanIgnoreLine| 0)
  9> (|incPrefix?| "command" 1 "1")
  <9 (|incPrefix?| NIL)
  9> (|scanToken|)
    10> (|startsComment?|)
    <10 (|startsComment?| NIL)
    10> (|startsNegComment?|)
    <10 (|startsNegComment?| NIL)
    10> (|punctuation?| 49)
    <10 (|punctuation?| NIL)
```

```
    10> (|digit?| #\1)
    11> (DIGITP #\1)
    <11 (DIGITP 1)
    <10 (|digit?| 1)
    10> (|scanNumber|)
    11> (|spleI| |digit?|)
    12> (|spleI1| |digit?| NIL)
    13> (|digit?| #\1)
    14> (DIGITP #\1)
    <14 (DIGITP 1)
    <13 (|digit?| 1)
    <12 (|spleI1| "1")
    <11 (|spleI| "1")
    11> (|lfinteger| "1")
    <11 (|lfinteger| (|integer| "1"))
    <10 (|scanNumber| (|integer| "1"))
    10> (|lnExtraBlanks| (0 "1" 1 1 "strings"))
    <10 (|lnExtraBlanks| 0)
    10> (|constoken|
          "1" (0 "1" 1 1 "strings") (|integer| "1") 0)
    11> (|ncPutQ|
         (|integer| . "1") |posn| ((0 "1" 1 1 "strings") . 0))
    12> (|ncAlist| (|integer| . "1"))
    <12 (|ncAlist| NIL)
    12> (|ncAlist| (|integer| . "1"))
    <12 (|ncAlist| NIL)
    12> (|ncTag| (|integer| . "1"))
    <12 (|ncTag| |integer|)
    <11 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
    <10 (|constoken|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) .
             "1"))
    10> (|dqUnit|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) .
             "1"))
    <10 (|dqUnit|
         (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) .
             "1")) . #0#))
  <9 (|scanToken|
       (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) .
             "1")) . #0#))
  9> (|dqAppend| NIL
      (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) .
             "1")) . #0#))
  <9 (|dqAppend|
      (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) .
             "1")) . #0#))
<8 (|lineoftoks|
    ((((#0=(
     ((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) . "1"))
         . #0#)
      (((#1# . 1) . "1") .
       #2=(|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
```

```
          (|nonnullstream| |incIgen1| 1)))))
     . #2#))

8> (|next| |lineoftoks|
    (|nonnullstream| |incZip1| |incRenumberLine|
     (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
     (|nonnullstream| |incIgen1| 1)))
  9> (|Delay| |next1|
      (|lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1))))
  <9 (|Delay|
      (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1))))
<8 (|next|
    (|nonnullstream| |next1| |lineoftoks|
     (|nonnullstream| |incZip1| |incRenumberLine|
      (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
      (|nonnullstream| |incIgen1| 1))))

8> (|incAppend|
    (((#0=((((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) . "1")) . #0#)
     (((#1# . 1) . "1") .
       #2=(|nonnullstream| |incZip1| |incRenumberLine|
       (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
       (|nonnullstream| |incIgen1| 1)))))
    (|nonnullstream| |next1| |lineoftoks| #2#))
  9> (|Delay| |incAppend1|
      ((((#1=((
        (|integer| (|posn| #2=(0 "1" 1 1 "strings") . 0)) .
           "1")) . #1#)
         (((#2# . 1) . "1") .
           #3=(|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1)))))
       (|nonnullstream| |next1| |lineoftoks| #3#)))
  <9 (|Delay|
       (|nonnullstream| |incAppend1|
        (((#1=
          ((((|integer| (|posn| #2=(0 "1" 1 1 "strings") . 0))
            . "1"))
             . #1#)
         (((#2# . 1) . "1") .
          #3=(|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1)))))
        (|nonnullstream| |next1| |lineoftoks| #3#)))
<8 (|incAppend|
    (|nonnullstream| |incAppend1|
     (((#1=((((|integer| (|posn| #2=(0 "1" 1 1 "strings") . 0)) . "1")) . #1#)
      (((#2# . 1) . "1") .
       #3=(|nonnullstream| |incZip1| |incRenumberLine|
```

```
            (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
            (|nonnullstream| |incIgen1| 1)))))
      (|nonnullstream| |next1| |lineoftoks| #3#)))

<7 (|next1|
    (|nonnullstream| |incAppend1|
     (((#1=(((|integer| (|posn| #2=(0 "1" 1 1 "strings") . 0))
       . "1"))
        . #1#) (((#2# . 1) . "1") .
         #3=(|nonnullstream| |incZip1| |incRenumberLine|
            (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
            (|nonnullstream| |incIgen1| 1)))))
     (|nonnullstream| |next1| |lineoftoks| #3#)))
7> (|incAppend1|
    (((#0=(((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) .
      "1"))
        . #0#) (((#1# . 1) . "1") .
     #2=(|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
         (|nonnullstream| |incIgen1| 1)))))
     (|nonnullstream| |next1| |lineoftoks| #2#))

  8> (|StreamNull|
      (((#0=(((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
        . "1"))
         . #0#) (((#1# . 1) . "1")
          |nonnullstream| |incZip1| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
          (|nonnullstream| |incIgen1| 1)))))
  <8 (|StreamNull| NIL)

  8> (|incAppend| NIL
      (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1))))
    9> (|Delay| |incAppend1|
        (NIL
         (|nonnullstream| |next1| |lineoftoks|
          (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1)))))
    <9 (|Delay|
        (|nonnullstream| |incAppend1| NIL
         (|nonnullstream| |next1| |lineoftoks|
          (|nonnullstream| |incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1)))))
  <8 (|incAppend| (|nonnullstream| |incAppend1| NIL
      (|nonnullstream| |next1| |lineoftoks|
       (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1)))))

<7 (|incAppend1|
    (((#0=(((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
```

```
        . "1")) . #0#) (((#1# . 1) . "1")
         . #2=(|nonnullstream| |incZip1| |incRenumberLine|
          (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
          (|nonnullstream| |incIgen1| 1))))
       |nonnullstream| |incAppend1| NIL
       (|nonnullstream| |next1| |lineoftoks| #2#)))
<6 (|StreamNull| NIL)

6> (|ncloopParse|
    (((#0=((((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
      . "1")) . #0#) (((#1# . 1) . "1")
       . #2=(|nonnullstream| |incZip1| |incRenumberLine|
       (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
       (|nonnullstream| |incIgen1| 1))))
     |nonnullstream| |incAppend1| NIL
     (|nonnullstream| |next1| |lineoftoks| #2#)))
  7> (|ncloopDQlines|
      (#0=((((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
       . "1")) . #0#) (((#1# . 1) . "1")
       |nonnullstream| |incZip1| |incRenumberLine|
       (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
       (|nonnullstream| |incIgen1| 1)))
    8> (|StreamNull|
        ((((0 "1" 1 1 "strings") . 1) . "1")
         |nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
         (|nonnullstream| |incIgen1| 1)))
    <8 (|StreamNull| NIL)
    8> (|tokPosn|
        ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
      9> (|ncAlist|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
      <9 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
    <8 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
    8> (|poGlobalLinePosn| ((0 "1" 1 1 "strings") . 0))
      9> (|poGetLineObject| ((0 "1" 1 1 "strings") . 0))
      <9 (|poGetLineObject| (0 "1" 1 1 "strings"))
      9> (|lnGlobalNum| (0 "1" 1 1 "strings"))
      <9 (|lnGlobalNum| 1)
    <8 (|poGlobalLinePosn| 1)
    8> (|poGlobalLinePosn| ((0 "1" 1 1 "strings") . 1))
      9> (|poGetLineObject| ((0 "1" 1 1 "strings") . 1))
      <9 (|poGetLineObject| (0 "1" 1 1 "strings"))
      9> (|lnGlobalNum| (0 "1" 1 1 "strings"))
      <9 (|lnGlobalNum| 1)
    <8 (|poGlobalLinePosn| 1)
    8> (|streamChop| 1
        ((((0 "1" 1 1 "strings") . 1) . "1")
         |nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
         (|nonnullstream| |incIgen1| 1)))
      9> (|StreamNull|
          ((((0 "1" 1 1 "strings") . 1) . "1")
           |nonnullstream| |incZip1| |incRenumberLine|
```

```
          (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
          (|nonnullstream| |incIgen1| 1)))
    <9 (|StreamNull| NIL)
    9> (|streamChop| 0
        (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
         (|nonnullstream| |incIgen1| 1)))
      10> (|StreamNull|
           (|nonnullstream| |incZip1| |incRenumberLine|
            (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
            (|nonnullstream| |incIgen1| 1)))
      11> (|incZip1| |incRenumberLine|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
           (|nonnullstream| |incIgen1| 1))
      12> (|StreamNull|
           (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1)))
      13> (|incLude1| 0 NIL 1 ("strings") (1))
      14> (|StreamNull| NIL)
      <14 (|StreamNull| T)
      14> (|Top?| 1)
      <14 (|Top?| T)
      <13 (|incLude1| (|nullstream|))
      <12 (|StreamNull| T)
      <11 (|incZip1| (|nullstream|))
      <10 (|StreamNull| T)
    <9 (|streamChop| (NIL NIL))
    9> (|ncloopPrefix?| ")command" "1")
    <9 (|ncloopPrefix?| NIL)
  <8 (|streamChop| (((((0 "1" 1 1 "strings") . 1) . "1")) NIL))
<7 (|ncloopDQlines| (((((0 "1" 1 1 "strings") . 1) . "1")) NIL))
7> (|dqToList|
    (#0=((
     (|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1")) . #0#))
<7 (|dqToList|
    ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
7> (|npParse|
    ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
  8> (|npFirstTok|)
    9> (|tokPart|
        ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
    <9 (|tokPart| "1")
  <8 (|npFirstTok| "1")
  8> (|npItem|)
    9> (|npQualDef|)
      10> (|npComma|)
      11> (|npTuple| |npQualifiedDefinition|)
      12> (|npListofFun|
           |npQualifiedDefinition|
           |npCommaBackSet|
           |pfTupleListOf|)
      13> (|npQualifiedDefinition|)
      14> (|npQualified| |npDefinitionOrStatement|)
      15> (|npDefinitionOrStatement|)
```

```
16> (|npBackTrack| |npGives| DEF |npDef|)
17> (|npState|)
<17 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
17> (|npGives|)
18> (|npBackTrack| |npExit| GIVES |npLambda|)
19> (|npState|)
<19 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
19> (|npExit|)
20> (|npBackTrack| |npAssign| EXIT |npPileExit|)
21> (|npState|)
<21 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
21> (|npAssign|)
22> (|npBackTrack| |npMDEF| BECOMES |npAssignment|)
23> (|npState|)
<23 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
23> (|npMDEF|)
24> (|npBackTrack| |npStatement| MDEF |npMDEFinition|)
25> (|npState|)
<25 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
25> (|npStatement|)
26> (|npExpress|)
27> (|npExpress1|)
28> (|npConditionalStatement|)
29> (|npConditional| |npQualifiedDefinition|)
30> (|npEqKey| IF)
<30 (|npEqKey| NIL)
<29 (|npConditional| NIL)
<28 (|npConditionalStatement| NIL)
28> (|npADD|)
29> (|npType|)
30> (|npMatch|)
31> (|npLeftAssoc| (IS ISNT) |npSuch|)
32> (|npSuch|)
33> (|npLeftAssoc| (BAR) |npLogical|)
34> (|npLogical|)
35> (|npLeftAssoc| (OR) |npDisjand|)
36> (|npDisjand|)
37> (|npLeftAssoc| (AND) |npDiscrim|)
38> (|npDiscrim|)
39> (|npLeftAssoc| (CASE HAS) |npQuiver|)
40> (|npQuiver|)
41> (|npRightAssoc| (ARROW LARROW) |npRelation|)
42> (|npState|)
<42 (|npState|
```

```
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
42> (|npRelation|)
43> (|npLeftAssoc|
     (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE)
     |npSynthetic|)
44> (|npSynthetic|)
45> (|npBy|)
46> (|npLeftAssoc| (BY) |npInterval|)
47> (|npInterval|)
48> (|npArith|)
49> (|npLeftAssoc| (MOD) |npSum|)
50> (|npSum|)
51> (|npLeftAssoc| (PLUS MINUS) |npTerm|)
52> (|npTerm|)
53> (|npInfGeneric| (MINUS PLUS))
54> (|npDDInfKey| (MINUS PLUS))
55> (|npInfKey| (MINUS PLUS))
<55 (|npInfKey| NIL)
55> (|npState|)
<55 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))))
55> (|npEqKey| |'|)
<55 (|npEqKey| NIL)
55> (|npRestore|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))))
56> (|npFirstTok|)
57> (|tokPart|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
<57 (|tokPart| "1")
<56 (|npFirstTok| "1")
<55 (|npRestore| T)
55> (|npEqKey| BACKQUOTE)
<55 (|npEqKey| NIL)
55> (|npRestore|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))))
56> (|npFirstTok|)
57> (|tokPart|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
<57 (|tokPart| "1")
<56 (|npFirstTok| "1")
<55 (|npRestore| T)
<54 (|npDDInfKey| NIL)
<53 (|npInfGeneric| NIL)
53> (|npRemainder|)
54> (|npLeftAssoc| (REM QUO) |npProduct|)
55> (|npProduct|)
56> (|npLeftAssoc|
      (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH
```

```
        SLASHBACKSLASH BACKSLASHSLASH)
      |npPower|)
57> (|npPower|)
58> (|npRightAssoc| (POWER CARAT) |npColon|)
59> (|npState|)
<59 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))))
59> (|npColon|)
60> (|npTypified|)
61> (|npApplication|)
62> (|npDotted| |npPrimary|)
63> (|npPrimary|)
64> (|npPrimary1|)
65> (|npEncAp| |npAtom1|)
66> (|npAtom1|)
67> (|npPDefinition|)
68> (|npParenthesized| |npDefinitionlist|)
69> (|npParenthesize| |(| |)| |npDefinitionlist|)
70> (|npEqKey| |(|)
<70 (|npEqKey| NIL)
<69 (|npParenthesize| NIL)
69> (|npParenthesize| |(\|| |\|)| |npDefinitionlist|)
70> (|npEqKey| |(\||)
<70 (|npEqKey| NIL)
<69 (|npParenthesize| NIL)
<68 (|npParenthesized| NIL)
<67 (|npPDefinition| NIL)
67> (|npName|)
68> (|npId|)
<68 (|npId| NIL)
68> (|npSymbolVariable|)
69> (|npState|)
<69 (|npState|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))))
69> (|npEqKey| BACKQUOTE)
<69 (|npEqKey| NIL)
69> (|npRestore|
     ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))))
70> (|npFirstTok|)
71> (|tokPart|
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
<71 (|tokPart| "1")
<70 (|npFirstTok| "1")
<69 (|npRestore| T)
<68 (|npSymbolVariable| NIL)
<67 (|npName| NIL)
67> (|npConstTok|)
68> (|tokType|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
```

```
69> (|ncTag|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
<69 (|ncTag| |integer|)
<68 (|tokType| |integer|)
68> (|npPush|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
<68 (|npPush|
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1")))
68> (|npNext|)
69> (|npFirstTok|)
70> (|tokPosn|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
71> (|ncAlist|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
<71 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<70 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
70> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
71> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
72> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<72 (|poNoPosition?| NIL)
<71 (|pfNoPosition?| NIL)
71> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
72> (|ncAlist| (ERROR . NOMORE))
<72 (|ncAlist| NIL)
72> (|ncAlist| (ERROR . NOMORE))
<72 (|ncAlist| NIL)
72> (|ncTag| (ERROR . NOMORE))
<72 (|ncTag| ERROR)
<71 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<70 (|tokConstruct|
      ((ERROR (|posn| (0 "1" 1 1 "strings") . 0))
        . NOMORE))
70> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<70 (|tokPart| NOMORE)
<69 (|npFirstTok| NOMORE)
<68 (|npNext| NOMORE)
<67 (|npConstTok| NOMORE)
67> (|npFromdom|)
68> (|npEqKey| $)
<68 (|npEqKey| NIL)
<67 (|npFromdom| T)
<66 (|npAtom1| T)
66> (|npAnyNo| |npEncl|)
67> (|npEncl|)
68> (|npBDefinition|)
69> (|npPDefinition|)
```

```
70> (|npParenthesized| |npDefinitionlist|)
71> (|npParenthesize| |(| |)| |npDefinitionlist|)
72> (|npEqKey| |(|)
<72 (|npEqKey| NIL)
<71 (|npParenthesize| NIL)
71> (|npParenthesize| |(\|| |\|)| |npDefinitionlist|)
72> (|npEqKey| |(\||)
<72 (|npEqKey| NIL)
<71 (|npParenthesize| NIL)
<70 (|npParenthesized| NIL)
<69 (|npPDefinition| NIL)
69> (|npBracketed| |npDefinitionlist|)
70> (|npParened| |npDefinitionlist|)
71> (|npEnclosed| |(| |)| |pfParen| |npDefinitionlist|)
72> (|npEqKey| |(|)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
71> (|npEnclosed| |(\|| |\|)| |pfParen| |npDefinitionlist|)
72> (|npEqKey| |(\||)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
<70 (|npParened| NIL)
70> (|npBracked| |npDefinitionlist|)
71> (|npEnclosed| [ ] |pfBracket| |npDefinitionlist|)
72> (|npEqKey| [)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
71> (|npEnclosed| |[\|| |\|]|
                  |pfBracketBar| |npDefinitionlist|)
72> (|npEqKey| |[\||)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
<70 (|npBracked| NIL)
70> (|npBraced| |npDefinitionlist|)
71> (|npEnclosed| { } |pfBrace| |npDefinitionlist|)
72> (|npEqKey| {)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
71> (|npEnclosed| |{\|| |\|}|
                  |pfBraceBar| |npDefinitionlist|)
72> (|npEqKey| |{\||)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
<70 (|npBraced| NIL)
70> (|npAngleBared| |npDefinitionlist|)
71> (|npEnclosed| |<\|| |\|>| |pfHide| |npDefinitionlist|)
72> (|npEqKey| |<\||)
<72 (|npEqKey| NIL)
<71 (|npEnclosed| NIL)
<70 (|npAngleBared| NIL)
<69 (|npBracketed| NIL)
<68 (|npBDefinition| NIL)
<67 (|npEncl| NIL)
<66 (|npAnyNo| T)
```

```
66> (|npFromdom|)
67> (|npEqKey| $)
<67 (|npEqKey| NIL)
<66 (|npFromdom| T)
<65 (|npEncAp| T)
<64 (|npPrimary1| T)
<63 (|npPrimary| T)
63> (|npAnyNo| |npSelector|)
64> (|npSelector|)
65> (|npEqKey| DOT)
<65 (|npEqKey| NIL)
<64 (|npSelector| NIL)
<63 (|npAnyNo| T)
<62 (|npDotted| T)
62> (|npApplication2|)
63> (|npDotted| |npPrimary1|)
64> (|npPrimary1|)
65> (|npEncAp| |npAtom1|)
66> (|npAtom1|)
67> (|npPDefinition|)
68> (|npParenthesized| |npDefinitionlist|)
69> (|npParenthesize| |(| |)| |npDefinitionlist|)
70> (|npEqKey| |(|)
<70 (|npEqKey| NIL)
<69 (|npParenthesize| NIL)
69> (|npParenthesize| |(\|| |\|)| |npDefinitionlist|)
70> (|npEqKey| |(\||)
<70 (|npEqKey| NIL)
<69 (|npParenthesize| NIL)
<68 (|npParenthesized| NIL)
<67 (|npPDefinition| NIL)
67> (|npName|)
68> (|npId|)
<68 (|npId| NIL)
68> (|npSymbolVariable|)
69> (|npState|)
<69 (|npState|
     (NIL ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
69> (|npEqKey| BACKQUOTE)
<69 (|npEqKey| NIL)
69> (|npRestore|
     (NIL ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
70> (|npFirstTok|)
71> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
72> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<72 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<71 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
71> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
72> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
```

```
73> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<73 (|poNoPosition?| NIL)
<72 (|pfNoPosition?| NIL)
72> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
73> (|ncAlist| (ERROR . NOMORE))
<73 (|ncAlist| NIL)
73> (|ncAlist| (ERROR . NOMORE))
<73 (|ncAlist| NIL)
73> (|ncTag| (ERROR . NOMORE))
<73 (|ncTag| ERROR)
<72 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<71 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
71> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<71 (|tokPart| NOMORE)
<70 (|npFirstTok| NOMORE)
<69 (|npRestore| T)
<68 (|npSymbolVariable| NIL)
<67 (|npName| NIL)
67> (|npConstTok|)
68> (|tokType|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
69> (|ncTag|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<69 (|ncTag| ERROR)
<68 (|tokType| ERROR)
68> (|npEqPeek| |'|)
<68 (|npEqPeek| NIL)
<67 (|npConstTok| NIL)
67> (|npDollar|)
68> (|npEqPeek| $)
<68 (|npEqPeek| NIL)
<67 (|npDollar| NIL)
67> (|npBDefinition|)
68> (|npPDefinition|)
69> (|npParenthesized| |npDefinitionlist|)
70> (|npParenthesize| |(| |)| |npDefinitionlist|)
71> (|npEqKey| |(|)
<71 (|npEqKey| NIL)
<70 (|npParenthesize| NIL)
70> (|npParenthesize| |(\|| |\|)| |npDefinitionlist|)
71> (|npEqKey| |(\||)
<71 (|npEqKey| NIL)
<70 (|npParenthesize| NIL)
<69 (|npParenthesized| NIL)
<68 (|npPDefinition| NIL)
68> (|npBracketed| |npDefinitionlist|)
69> (|npParened| |npDefinitionlist|)
70> (|npEnclosed| |(| |)| |pfParen| |npDefinitionlist|)
71> (|npEqKey| |(|)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
```

```
70> (|npEnclosed| |(\|| |\|)| |pfParen| |npDefinitionlist|)
71> (|npEqKey| |(\||)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
<69 (|npParened| NIL)
69> (|npBracked| |npDefinitionlist|)
70> (|npEnclosed| [ ] |pfBracket| |npDefinitionlist|)
71> (|npEqKey| [)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
70> (|npEnclosed| |[\|| |\|]|
                  |pfBracketBar| |npDefinitionlist|)
71> (|npEqKey| |[\||)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
<69 (|npBracked| NIL)
69> (|npBraced| |npDefinitionlist|)
70> (|npEnclosed| { } |pfBrace| |npDefinitionlist|)
71> (|npEqKey| {)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
70> (|npEnclosed| |{\|| |\|}|
                  |pfBraceBar| |npDefinitionlist|)
71> (|npEqKey| |{\||)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
<69 (|npBraced| NIL)
69> (|npAngleBared| |npDefinitionlist|)
70> (|npEnclosed| |<\|| |\|>| |pfHide| |npDefinitionlist|)
71> (|npEqKey| |<\||)
<71 (|npEqKey| NIL)
<70 (|npEnclosed| NIL)
<69 (|npAngleBared| NIL)
<68 (|npBracketed| NIL)
<67 (|npBDefinition| NIL)
<66 (|npAtom1| NIL)
<65 (|npEncAp| NIL)
65> (|npLet|)
66> (|npLetQualified| |npDefinitionOrStatement|)
67> (|npEqKey| LET)
<67 (|npEqKey| NIL)
<66 (|npLetQualified| NIL)
<65 (|npLet| NIL)
65> (|npFix|)
66> (|npEqKey| FIX)
<66 (|npEqKey| NIL)
<65 (|npFix| NIL)
65> (|npMacro|)
66> (|npEqKey| MACRO)
<66 (|npEqKey| NIL)
<65 (|npMacro| NIL)
65> (|npBPileDefinition|)
66> (|npPileBracketed| |npPileDefinitionlist|)
67> (|npEqKey| SETTAB)
```

```
<67 (|npEqKey| NIL)
<66 (|npPileBracketed| NIL)
<65 (|npBPileDefinition| NIL)
65> (|npDefn|)
66> (|npEqKey| DEFN)
<66 (|npEqKey| NIL)
<65 (|npDefn| NIL)
65> (|npRule|)
66> (|npEqKey| RULE)
<66 (|npEqKey| NIL)
<65 (|npRule| NIL)
<64 (|npPrimary1| NIL)
<63 (|npDotted| NIL)
<62 (|npApplication2| NIL)
<61 (|npApplication| T)
61> (|npAnyNo| |npTypeStyle|)
62> (|npTypeStyle|)
63> (|npCoerceTo|)
64> (|npTypedForm| COERCE |pfCoerceto|)
65> (|npEqKey| COERCE)
<65 (|npEqKey| NIL)
<64 (|npTypedForm| NIL)
<63 (|npCoerceTo| NIL)
63> (|npRestrict|)
64> (|npTypedForm| AT |pfRestrict|)
65> (|npEqKey| AT)
<65 (|npEqKey| NIL)
<64 (|npTypedForm| NIL)
<63 (|npRestrict| NIL)
63> (|npPretend|)
64> (|npTypedForm| PRETEND |pfPretend|)
65> (|npEqKey| PRETEND)
<65 (|npEqKey| NIL)
<64 (|npTypedForm| NIL)
<63 (|npPretend| NIL)
63> (|npColonQuery|)
64> (|npTypedForm| ATAT |pfRetractTo|)
65> (|npEqKey| ATAT)
<65 (|npEqKey| NIL)
<64 (|npTypedForm| NIL)
<63 (|npColonQuery| NIL)
<62 (|npTypeStyle| NIL)
<61 (|npAnyNo| T)
<60 (|npTypified| T)
60> (|npAnyNo| |npTagged|)
61> (|npTagged|)
62> (|npTypedForm1| COLON |pfTagged|)
63> (|npEqKey| COLON)
<63 (|npEqKey| NIL)
<62 (|npTypedForm1| NIL)
<61 (|npTagged| NIL)
<60 (|npAnyNo| T)
<59 (|npColon| T)
59> (|npInfGeneric| (POWER CARAT))
```

```
60> (|npDDInfKey| (POWER CARAT))
61> (|npInfKey| (POWER CARAT))
<61 (|npInfKey| NIL)
61> (|npState|)
<61 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
61> (|npEqKey| |'|)
<61 (|npEqKey| NIL)
61> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
62> (|npFirstTok|)
63> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
64> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<64 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<63 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
63> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
64> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
65> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<65 (|poNoPosition?| NIL)
<64 (|pfNoPosition?| NIL)
64> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
65> (|ncAlist| (ERROR . NOMORE))
<65 (|ncAlist| NIL)
65> (|ncAlist| (ERROR . NOMORE))
<65 (|ncAlist| NIL)
65> (|ncTag| (ERROR . NOMORE))
<65 (|ncTag| ERROR)
<64 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<63 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
63> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<63 (|tokPart| NOMORE)
<62 (|npFirstTok| NOMORE)
<61 (|npRestore| T)
61> (|npEqKey| BACKQUOTE)
<61 (|npEqKey| NIL)
61> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
62> (|npFirstTok|)
63> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
64> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
```

```
<64 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<63 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
63> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
64> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
65> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<65 (|poNoPosition?| NIL)
<64 (|pfNoPosition?| NIL)
64> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
65> (|ncAlist| (ERROR . NOMORE))
<65 (|ncAlist| NIL)
65> (|ncAlist| (ERROR . NOMORE))
<65 (|ncAlist| NIL)
65> (|ncTag| (ERROR . NOMORE))
<65 (|ncTag| ERROR)
<64 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<63 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
63> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<63 (|tokPart| NOMORE)
<62 (|npFirstTok| NOMORE)
<61 (|npRestore| T)
<60 (|npDDInfKey| NIL)
<59 (|npInfGeneric| NIL)
<58 (|npRightAssoc| T)
<57 (|npPower| T)
57> (|npInfGeneric|
     (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH
      SLASHBACKSLASH BACKSLASHSLASH))
58> (|npDDInfKey|
     (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH
      SLASHBACKSLASH BACKSLASHSLASH))
59> (|npInfKey|
     (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH
      SLASHBACKSLASH BACKSLASHSLASH))
<59 (|npInfKey| NIL)
59> (|npState|)
<59 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
59> (|npEqKey| |'|)
<59 (|npEqKey| NIL)
59> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
60> (|npFirstTok|)
61> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
62> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
```

```
<62 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<61 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
61> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
62> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
63> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<63 (|poNoPosition?| NIL)
<62 (|pfNoPosition?| NIL)
62> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
63> (|ncAlist| (ERROR . NOMORE))
<63 (|ncAlist| NIL)
63> (|ncAlist| (ERROR . NOMORE))
<63 (|ncAlist| NIL)
63> (|ncTag| (ERROR . NOMORE))
<63 (|ncTag| ERROR)
<62 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<61 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
61> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<61 (|tokPart| NOMORE)
<60 (|npFirstTok| NOMORE)
<59 (|npRestore| T)
59> (|npEqKey| BACKQUOTE)
<59 (|npEqKey| NIL)
59> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
60> (|npFirstTok|)
61> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
62> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<62 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<61 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
61> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
62> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
63> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<63 (|poNoPosition?| NIL)
<62 (|pfNoPosition?| NIL)
62> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
63> (|ncAlist| (ERROR . NOMORE))
<63 (|ncAlist| NIL)
63> (|ncAlist| (ERROR . NOMORE))
<63 (|ncAlist| NIL)
63> (|ncTag| (ERROR . NOMORE))
<63 (|ncTag| ERROR)
<62 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<61 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
```

```
61> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<61 (|tokPart| NOMORE)
<60 (|npFirstTok| NOMORE)
<59 (|npRestore| T)
<58 (|npDDInfKey| NIL)
<57 (|npInfGeneric| NIL)
<56 (|npLeftAssoc| T)
<55 (|npProduct| T)
55> (|npInfGeneric| (REM QUO))
56> (|npDDInfKey| (REM QUO))
57> (|npInfKey| (REM QUO))
<57 (|npInfKey| NIL)
57> (|npState|)
<57 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
57> (|npEqKey| |'|)
<57 (|npEqKey| NIL)
57> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
58> (|npFirstTok|)
59> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
60> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<60 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<59 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
59> (|tokConstruct| ERROR NOMORE (
     (0 "1" 1 1 "strings") . 0))
60> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
61> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<61 (|poNoPosition?| NIL)
<60 (|pfNoPosition?| NIL)
60> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
61> (|ncAlist| (ERROR . NOMORE))
<61 (|ncAlist| NIL)
61> (|ncAlist| (ERROR . NOMORE))
<61 (|ncAlist| NIL)
61> (|ncTag| (ERROR . NOMORE))
<61 (|ncTag| ERROR)
<60 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<59 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
59> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<59 (|tokPart| NOMORE)
<58 (|npFirstTok| NOMORE)
<57 (|npRestore| T)
57> (|npEqKey| BACKQUOTE)
```

```
<57 (|npEqKey| NIL)
57> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
58> (|npFirstTok|)
59> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
60> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<60 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<59 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
59> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
60> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
61> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<61 (|poNoPosition?| NIL)
<60 (|pfNoPosition?| NIL)
60> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
61> (|ncAlist| (ERROR . NOMORE))
<61 (|ncAlist| NIL)
61> (|ncAlist| (ERROR . NOMORE))
<61 (|ncAlist| NIL)
61> (|ncTag| (ERROR . NOMORE))
<61 (|ncTag| ERROR)
<60 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<59 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
59> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<59 (|tokPart| NOMORE)
<58 (|npFirstTok| NOMORE)
<57 (|npRestore| T)
<56 (|npDDInfKey| NIL)
<55 (|npInfGeneric| NIL)
<54 (|npLeftAssoc| T)
<53 (|npRemainder| T)
<52 (|npTerm| T)
52> (|npInfGeneric| (PLUS MINUS))
53> (|npDDInfKey| (PLUS MINUS))
54> (|npInfKey| (PLUS MINUS))
<54 (|npInfKey| NIL)
54> (|npState|)
<54 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
54> (|npEqKey| |'|)
<54 (|npEqKey| NIL)
54> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
```

```
55> (|npFirstTok|)
56> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
57> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<57 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<56 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
56> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
57> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
58> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<58 (|poNoPosition?| NIL)
<57 (|pfNoPosition?| NIL)
57> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncTag| (ERROR . NOMORE))
<58 (|ncTag| ERROR)
<57 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<56 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
56> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<56 (|tokPart| NOMORE)
<55 (|npFirstTok| NOMORE)
<54 (|npRestore| T)
54> (|npEqKey| BACKQUOTE)
<54 (|npEqKey| NIL)
54> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
55> (|npFirstTok|)
56> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
57> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<57 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<56 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
56> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
57> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
58> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<58 (|poNoPosition?| NIL)
<57 (|pfNoPosition?| NIL)
57> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
```

```
58> (|ncTag| (ERROR . NOMORE))
<58 (|ncTag| ERROR)
<57 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<56 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
56> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<56 (|tokPart| NOMORE)
<55 (|npFirstTok| NOMORE)
<54 (|npRestore| T)
<53 (|npDDInfKey| NIL)
<52 (|npInfGeneric| NIL)
<51 (|npLeftAssoc| T)
<50 (|npSum| T)
50> (|npInfGeneric| (MOD))
51> (|npDDInfKey| (MOD))
52> (|npInfKey| (MOD))
<52 (|npInfKey| NIL)
52> (|npState|)
<52 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
52> (|npEqKey| |'|)
<52 (|npEqKey| NIL)
52> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
53> (|npFirstTok|)
54> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
55> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<55 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<54 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
54> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
55> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
56> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<56 (|poNoPosition?| NIL)
<55 (|pfNoPosition?| NIL)
55> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
56> (|ncTag| (ERROR . NOMORE))
<56 (|ncTag| ERROR)
<55 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<54 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
54> (|tokPart|
```

```
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<54 (|tokPart| NOMORE)
<53 (|npFirstTok| NOMORE)
<52 (|npRestore| T)
52> (|npEqKey| BACKQUOTE)
<52 (|npEqKey| NIL)
52> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
53> (|npFirstTok|)
54> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
55> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<55 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<54 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
54> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
55> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
56> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<56 (|poNoPosition?| NIL)
<55 (|pfNoPosition?| NIL)
55> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
56> (|ncTag| (ERROR . NOMORE))
<56 (|ncTag| ERROR)
<55 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<54 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
54> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<54 (|tokPart| NOMORE)
<53 (|npFirstTok| NOMORE)
<52 (|npRestore| T)
<51 (|npDDInfKey| NIL)
<50 (|npInfGeneric| NIL)
<49 (|npLeftAssoc| T)
<48 (|npArith| T)
48> (|npSegment|)
49> (|npEqPeek| SEG)
<49 (|npEqPeek| NIL)
<48 (|npSegment| NIL)
<47 (|npInterval| T)
47> (|npInfGeneric| (BY))
48> (|npDDInfKey| (BY))
49> (|npInfKey| (BY))
<49 (|npInfKey| NIL)
49> (|npState|)
<49 (|npState|
```

```
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
49> (|npEqKey| |'|)
<49 (|npEqKey| NIL)
49> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
50> (|npFirstTok|)
51> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
52> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<52 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<51 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
51> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
52> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
53> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<53 (|poNoPosition?| NIL)
<52 (|pfNoPosition?| NIL)
52> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
53> (|ncAlist| (ERROR . NOMORE))
<53 (|ncAlist| NIL)
53> (|ncAlist| (ERROR . NOMORE))
<53 (|ncAlist| NIL)
53> (|ncTag| (ERROR . NOMORE))
<53 (|ncTag| ERROR)
<52 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<51 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
51> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<51 (|tokPart| NOMORE)
<50 (|npFirstTok| NOMORE)
<49 (|npRestore| T)
49> (|npEqKey| BACKQUOTE)
<49 (|npEqKey| NIL)
49> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
50> (|npFirstTok|)
51> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
52> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<52 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<51 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
51> (|tokConstruct| ERROR NOMORE ((0 "1" 1 1 "strings") .
       0))
52> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
```

```
53> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<53 (|poNoPosition?| NIL)
<52 (|pfNoPosition?| NIL)
52> (|ncPutQ|
     (ERROR . NOMORE) |posn| ((0 "1" 1 1 "strings") . 0))
53> (|ncAlist| (ERROR . NOMORE))
<53 (|ncAlist| NIL)
53> (|ncAlist| (ERROR . NOMORE))
<53 (|ncAlist| NIL)
53> (|ncTag| (ERROR . NOMORE))
<53 (|ncTag| ERROR)
<52 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<51 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
51> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<51 (|tokPart| NOMORE)
<50 (|npFirstTok| NOMORE)
<49 (|npRestore| T)
<48 (|npDDInfKey| NIL)
<47 (|npInfGeneric| NIL)
<46 (|npLeftAssoc| T)
<45 (|npBy| T)
45> (|npAmpersandFrom|)
46> (|npAmpersand|)
47> (|npEqKey| AMPERSAND)
<47 (|npEqKey| NIL)
<46 (|npAmpersand| NIL)
<45 (|npAmpersandFrom| NIL)
<44 (|npSynthetic| T)
44> (|npInfGeneric|
     (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE))
45> (|npDDInfKey|
     (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE))
46> (|npInfKey| (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE))
<46 (|npInfKey| NIL)
46> (|npState|)
<46 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
46> (|npEqKey| |'|)
<46 (|npEqKey| NIL)
46> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
47> (|npFirstTok|)
48> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
49> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<49 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<48 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
```

```
48> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
49> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
50> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<50 (|poNoPosition?| NIL)
<49 (|pfNoPosition?| NIL)
49> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
50> (|ncAlist| (ERROR . NOMORE))
<50 (|ncAlist| NIL)
50> (|ncAlist| (ERROR . NOMORE))
<50 (|ncAlist| NIL)
50> (|ncTag| (ERROR . NOMORE))
<50 (|ncTag| ERROR)
<49 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<48 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
48> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<48 (|tokPart| NOMORE)
<47 (|npFirstTok| NOMORE)
<46 (|npRestore| T)
46> (|npEqKey| BACKQUOTE)
<46 (|npEqKey| NIL)
46> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
47> (|npFirstTok|)
48> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
49> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<49 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<48 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
48> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
49> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
50> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<50 (|poNoPosition?| NIL)
<49 (|pfNoPosition?| NIL)
49> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
50> (|ncAlist| (ERROR . NOMORE))
<50 (|ncAlist| NIL)
50> (|ncAlist| (ERROR . NOMORE))
<50 (|ncAlist| NIL)
50> (|ncTag| (ERROR . NOMORE))
<50 (|ncTag| ERROR)
<49 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<48 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
48> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
```

```
<48 (|tokPart| NOMORE)
<47 (|npFirstTok| NOMORE)
<46 (|npRestore| T)
<45 (|npDDInfKey| NIL)
<44 (|npInfGeneric| NIL)
<43 (|npLeftAssoc| T)
<42 (|npRelation| T)
42> (|npInfGeneric| (ARROW LARROW))
43> (|npDDInfKey| (ARROW LARROW))
44> (|npInfKey| (ARROW LARROW))
<44 (|npInfKey| NIL)
44> (|npState|)
<44 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
44> (|npEqKey| |'|)
<44 (|npEqKey| NIL)
44> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
45> (|npFirstTok|)
46> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
47> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<47 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<46 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
46> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
47> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
48> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<48 (|poNoPosition?| NIL)
<47 (|pfNoPosition?| NIL)
47> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
48> (|ncAlist| (ERROR . NOMORE))
<48 (|ncAlist| NIL)
48> (|ncAlist| (ERROR . NOMORE))
<48 (|ncAlist| NIL)
48> (|ncTag| (ERROR . NOMORE))
<48 (|ncTag| ERROR)
<47 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<46 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
46> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<46 (|tokPart| NOMORE)
<45 (|npFirstTok| NOMORE)
<44 (|npRestore| T)
44> (|npEqKey| BACKQUOTE)
<44 (|npEqKey| NIL)
44> (|npRestore|
```

```
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
45> (|npFirstTok|)
46> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
47> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<47 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<46 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
46> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
47> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
48> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<48 (|poNoPosition?| NIL)
<47 (|pfNoPosition?| NIL)
47> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
48> (|ncAlist| (ERROR . NOMORE))
<48 (|ncAlist| NIL)
48> (|ncAlist| (ERROR . NOMORE))
<48 (|ncAlist| NIL)
48> (|ncTag| (ERROR . NOMORE))
<48 (|ncTag| ERROR)
<47 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<46 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
46> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<46 (|tokPart| NOMORE)
<45 (|npFirstTok| NOMORE)
<44 (|npRestore| T)
<43 (|npDDInfKey| NIL)
<42 (|npInfGeneric| NIL)
<41 (|npRightAssoc| T)
<40 (|npQuiver| T)
40> (|npInfGeneric| (CASE HAS))
41> (|npDDInfKey| (CASE HAS))
42> (|npInfKey| (CASE HAS))
<42 (|npInfKey| NIL)
42> (|npState|)
<42 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
42> (|npEqKey| |'|)
<42 (|npEqKey| NIL)
42> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
43> (|npFirstTok|)
44> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
```

```
45> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<45 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<44 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
44> (|tokConstruct| ERROR NOMORE
     ((0 "1" 1 1 "strings") . 0))
45> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
46> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<46 (|poNoPosition?| NIL)
<45 (|pfNoPosition?| NIL)
45> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
46> (|ncAlist| (ERROR . NOMORE))
<46 (|ncAlist| NIL)
46> (|ncAlist| (ERROR . NOMORE))
<46 (|ncAlist| NIL)
46> (|ncTag| (ERROR . NOMORE))
<46 (|ncTag| ERROR)
<45 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<44 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
44> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<44 (|tokPart| NOMORE)
<43 (|npFirstTok| NOMORE)
<42 (|npRestore| T)
42> (|npEqKey| BACKQUOTE)
<42 (|npEqKey| NIL)
42> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
43> (|npFirstTok|)
44> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
45> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<45 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<44 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
44> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
45> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
46> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<46 (|poNoPosition?| NIL)
<45 (|pfNoPosition?| NIL)
45> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
46> (|ncAlist| (ERROR . NOMORE))
<46 (|ncAlist| NIL)
46> (|ncAlist| (ERROR . NOMORE))
<46 (|ncAlist| NIL)
46> (|ncTag| (ERROR . NOMORE))
<46 (|ncTag| ERROR)
<45 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
```

```
<44 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
44> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<44 (|tokPart| NOMORE)
<43 (|npFirstTok| NOMORE)
<42 (|npRestore| T)
<41 (|npDDInfKey| NIL)
<40 (|npInfGeneric| NIL)
<39 (|npLeftAssoc| T)
<38 (|npDiscrim| T)
38> (|npInfGeneric| (AND))
39> (|npDDInfKey| (AND))
40> (|npInfKey| (AND))
<40 (|npInfKey| NIL)
40> (|npState|)
<40 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
40> (|npEqKey| |'|)
<40 (|npEqKey| NIL)
40> (|npRestore|
      (NIL
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
41> (|npFirstTok|)
42> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
43> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<43 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<42 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
42> (|tokConstruct| ERROR NOMORE ((0 "1" 1 1 "strings") . 0))
43> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
44> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<44 (|poNoPosition?| NIL)
<43 (|pfNoPosition?| NIL)
43> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
44> (|ncAlist| (ERROR . NOMORE))
<44 (|ncAlist| NIL)
44> (|ncAlist| (ERROR . NOMORE))
<44 (|ncAlist| NIL)
44> (|ncTag| (ERROR . NOMORE))
<44 (|ncTag| ERROR)
<43 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<42 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
42> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<42 (|tokPart| NOMORE)
<41 (|npFirstTok| NOMORE)
<40 (|npRestore| T)
```

```
40> (|npEqKey| BACKQUOTE)
<40 (|npEqKey| NIL)
40> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
41> (|npFirstTok|)
42> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
43> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<43 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<42 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
42> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
43> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
44> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<44 (|poNoPosition?| NIL)
<43 (|pfNoPosition?| NIL)
43> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
44> (|ncAlist| (ERROR . NOMORE))
<44 (|ncAlist| NIL)
44> (|ncAlist| (ERROR . NOMORE))
<44 (|ncAlist| NIL)
44> (|ncTag| (ERROR . NOMORE))
<44 (|ncTag| ERROR)
<43 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<42 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
42> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<42 (|tokPart| NOMORE)
<41 (|npFirstTok| NOMORE)
<40 (|npRestore| T)
<39 (|npDDInfKey| NIL)
<38 (|npInfGeneric| NIL)
<37 (|npLeftAssoc| T)
<36 (|npDisjand| T)
36> (|npInfGeneric| (OR))
37> (|npDDInfKey| (OR))
38> (|npInfKey| (OR))
<38 (|npInfKey| NIL)
38> (|npState|)
<38 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
38> (|npEqKey| |'|)
<38 (|npEqKey| NIL)
38> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
```

```
39> (|npFirstTok|)
40> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
41> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<41 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<40 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
40> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
41> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
42> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<42 (|poNoPosition?| NIL)
<41 (|pfNoPosition?| NIL)
41> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
42> (|ncAlist| (ERROR . NOMORE))
<42 (|ncAlist| NIL)
42> (|ncAlist| (ERROR . NOMORE))
<42 (|ncAlist| NIL)
42> (|ncTag| (ERROR . NOMORE))
<42 (|ncTag| ERROR)
<41 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<40 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
40> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<40 (|tokPart| NOMORE)
<39 (|npFirstTok| NOMORE)
<38 (|npRestore| T)
38> (|npEqKey| BACKQUOTE)
<38 (|npEqKey| NIL)
38> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
39> (|npFirstTok|)
40> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
41> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<41 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<40 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
40> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
41> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
42> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<42 (|poNoPosition?| NIL)
<41 (|pfNoPosition?| NIL)
41> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
42> (|ncAlist| (ERROR . NOMORE))
<42 (|ncAlist| NIL)
42> (|ncAlist| (ERROR . NOMORE))
<42 (|ncAlist| NIL)
```

```
42> (|ncTag| (ERROR . NOMORE))
<42 (|ncTag| ERROR)
<41 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<40 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
40> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<40 (|tokPart| NOMORE)
<39 (|npFirstTok| NOMORE)
<38 (|npRestore| T)
<37 (|npDDInfKey| NIL)
<36 (|npInfGeneric| NIL)
<35 (|npLeftAssoc| T)
<34 (|npLogical| T)
34> (|npInfGeneric| (BAR))
35> (|npDDInfKey| (BAR))
36> (|npInfKey| (BAR))
<36 (|npInfKey| NIL)
36> (|npState|)
<36 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
36> (|npEqKey| |'|)
<36 (|npEqKey| NIL)
36> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
37> (|npFirstTok|)
38> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
39> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<39 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<38 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
38> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
39> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
40> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<40 (|poNoPosition?| NIL)
<39 (|pfNoPosition?| NIL)
39> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
40> (|ncAlist| (ERROR . NOMORE))
<40 (|ncAlist| NIL)
40> (|ncAlist| (ERROR . NOMORE))
<40 (|ncAlist| NIL)
40> (|ncTag| (ERROR . NOMORE))
<40 (|ncTag| ERROR)
<39 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<38 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
38> (|tokPart|
```

```
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<38 (|tokPart| NOMORE)
<37 (|npFirstTok| NOMORE)
<36 (|npRestore| T)
36> (|npEqKey| BACKQUOTE)
<36 (|npEqKey| NIL)
36> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
37> (|npFirstTok|)
38> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
39> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<39 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<38 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
38> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
39> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
40> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<40 (|poNoPosition?| NIL)
<39 (|pfNoPosition?| NIL)
39> (|ncPutQ| (ERROR . NOMORE) |posn|
     ((0 "1" 1 1 "strings") . 0))
40> (|ncAlist| (ERROR . NOMORE))
<40 (|ncAlist| NIL)
40> (|ncAlist| (ERROR . NOMORE))
<40 (|ncAlist| NIL)
40> (|ncTag| (ERROR . NOMORE))
<40 (|ncTag| ERROR)
<39 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<38 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
38> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<38 (|tokPart| NOMORE)
<37 (|npFirstTok| NOMORE)
<36 (|npRestore| T)
<35 (|npDDInfKey| NIL)
<34 (|npInfGeneric| NIL)
<33 (|npLeftAssoc| T)
<32 (|npSuch| T)
32> (|npInfGeneric| (IS ISNT))
33> (|npDDInfKey| (IS ISNT))
34> (|npInfKey| (IS ISNT))
<34 (|npInfKey| NIL)
34> (|npState|)
<34 (|npState|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
34> (|npEqKey| |'|)
<34 (|npEqKey| NIL)
```

```
34> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1")))
35> (|npFirstTok|)
36> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
37> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<37 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<36 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
36> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
37> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
38> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<38 (|poNoPosition?| NIL)
<37 (|pfNoPosition?| NIL)
37> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
38> (|ncAlist| (ERROR . NOMORE))
<38 (|ncAlist| NIL)
38> (|ncAlist| (ERROR . NOMORE))
<38 (|ncAlist| NIL)
38> (|ncTag| (ERROR . NOMORE))
<38 (|ncTag| ERROR)
<37 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<36 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
36> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<36 (|tokPart| NOMORE)
<35 (|npFirstTok| NOMORE)
<34 (|npRestore| T)
34> (|npEqKey| BACKQUOTE)
<34 (|npEqKey| NIL)
34> (|npRestore|
     (NIL
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
35> (|npFirstTok|)
36> (|tokPosn|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
37> (|ncAlist|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<37 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<36 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
36> (|tokConstruct| ERROR NOMORE
      ((0 "1" 1 1 "strings") . 0))
37> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
38> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<38 (|poNoPosition?| NIL)
<37 (|pfNoPosition?| NIL)
37> (|ncPutQ| (ERROR . NOMORE) |posn|
      ((0 "1" 1 1 "strings") . 0))
```

```
38> (|ncAlist| (ERROR . NOMORE))
<38 (|ncAlist| NIL)
38> (|ncAlist| (ERROR . NOMORE))
<38 (|ncAlist| NIL)
38> (|ncTag| (ERROR . NOMORE))
<38 (|ncTag| ERROR)
<37 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<36 (|tokConstruct|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
36> (|tokPart|
     ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<36 (|tokPart| NOMORE)
<35 (|npFirstTok| NOMORE)
<34 (|npRestore| T)
<33 (|npDDInfKey| NIL)
<32 (|npInfGeneric| NIL)
<31 (|npLeftAssoc| T)
<30 (|npMatch| T)
30> (|npPop1|)
<30 (|npPop1|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
30> (|npWith|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
31> (|npEqKey| WITH)
<31 (|npEqKey| NIL)
<30 (|npWith| NIL)
30> (|npPush|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
<30 (|npPush|
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
<29 (|npType|
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
29> (|npPop1|)
<29 (|npPop1|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
29> (|npAdd|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
30> (|npEqKey| ADD)
<30 (|npEqKey| NIL)
<29 (|npAdd| NIL)
29> (|npPush|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1"))
<29 (|npPush|
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
<28 (|npADD|
```

```
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1")))
<27 (|npExpress1|
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1")))
27> (|npIterators|)
28> (|npForIn|)
29> (|npEqKey| FOR)
<29 (|npEqKey| NIL)
<28 (|npForIn| NIL)
28> (|npWhile|)
29> (|npAndOr| WHILE |npLogical| |pfWhile|)
30> (|npEqKey| WHILE)
<30 (|npEqKey| NIL)
<29 (|npAndOr| NIL)
<28 (|npWhile| NIL)
<27 (|npIterators| NIL)
<26 (|npExpress| T)
<25 (|npStatement| T)
25> (|npEqPeek| MDEF)
<25 (|npEqPeek| NIL)
<24 (|npBackTrack| T)
<23 (|npMDEF| T)
23> (|npEqPeek| BECOMES)
<23 (|npEqPeek| NIL)
<22 (|npBackTrack| T)
<21 (|npAssign| T)
21> (|npEqPeek| EXIT)
<21 (|npEqPeek| NIL)
<20 (|npBackTrack| T)
<19 (|npExit| T)
19> (|npEqPeek| GIVES)
<19 (|npEqPeek| NIL)
<18 (|npBackTrack| T)
<17 (|npGives| T)
17> (|npEqPeek| DEF)
<17 (|npEqPeek| NIL)
<16 (|npBackTrack| T)
<15 (|npDefinitionOrStatement| T)
15> (|npEqKey| WHERE)
<15 (|npEqKey| NIL)
<14 (|npQualified| T)
<13 (|npQualifiedDefinition| T)
13> (|npCommaBackSet|)
14> (|npEqKey| COMMA)
<14 (|npEqKey| NIL)
<13 (|npCommaBackSet| NIL)
<12 (|npListofFun| T)
<11 (|npTuple| T)
<10 (|npComma| T)
10> (|npPop1|)
<10 (|npPop1|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
```

```
        10> (|npPush|
             (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
               . "1")))
        <10 (|npPush|
             ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
               . "1"))))
       <9 (|npQualDef|
           ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
             . "1"))))
       9> (|npEqKey| SEMICOLON)
       <9 (|npEqKey| NIL)
       9> (|npPop1|)
       <9 (|npPop1|
           (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
       9> (|pfEnSequence|
           (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
       <9 (|pfEnSequence|
           ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
       9> (|npPush|
           ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
       <9 (|npPush|
           (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
     <8 (|npItem|
         (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
   <7 (|npParse|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
<6 (|ncloopParse|
    ((((((#0=(0 "1" 1 1 "strings") . 1) . "1"))
       ((|integer| (|posn| #0# . 0)) . "1")))
     |nonnullstream| |incAppend1| NIL
     (|nonnullstream| |next1| |lineoftoks| (|nullstream|))))

6> (|next| |ncloopParse|
    (|nonnullstream| |incAppend1| NIL
     (|nonnullstream| |next1| |lineoftoks| (|nullstream|))))
  7> (|Delay| #0=|next1|
      (|ncloopParse|
       (|nonnullstream| |incAppend1| NIL
        (|nonnullstream| #0# |lineoftoks| (|nullstream|)))))
  <7 (|Delay|
      (|nonnullstream| #0=|next1| |ncloopParse|
       (|nonnullstream| |incAppend1| NIL
        (|nonnullstream| #0# |lineoftoks| (|nullstream|)))))
<6 (|next|
    (|nonnullstream| #0=|next1| |ncloopParse|
     (|nonnullstream| |incAppend1| NIL
      (|nonnullstream| #0# |lineoftoks| (|nullstream|)))))

6> (|incAppend|
    (((((#0=(0 "1" 1 1 "strings") . 1) . "1"))
       ((|integer| (|posn| #0# . 0)) . "1")))
     (|nonnullstream| #1=|next1| |ncloopParse|
      (|nonnullstream| |incAppend1| NIL
       (|nonnullstream| #1# |lineoftoks| (|nullstream|)))))
  7> (|Delay| #0=|incAppend1|
```

```
        ((((((#2=(0 "1" 1 1 "strings") . 1) . "1"))
            ((|integer| (|posn| #2# . 0)) . "1")))
           (|nonnullstream| #3=|next1| |ncloopParse|
            (|nonnullstream| #0# NIL
             (|nonnullstream| #3# |lineoftoks| (|nullstream|))))))
    <7 (|Delay|
        (|nonnullstream| #0=|incAppend1|
         (((((#2=(0 "1" 1 1 "strings") . 1) . "1"))
            ((|integer| (|posn| #2# . 0)) . "1")))
           (|nonnullstream| #3=|next1| |ncloopParse|
            (|nonnullstream| #0# NIL
             (|nonnullstream| #3# |lineoftoks| (|nullstream|))))))
  <6 (|incAppend|
      (|nonnullstream| #0=|incAppend1|
       (((((#2=(0 "1" 1 1 "strings") . 1) . "1"))
          ((|integer| (|posn| #2# . 0)) . "1")))
         (|nonnullstream| #3=|next1| |ncloopParse|
          (|nonnullstream| #0# NIL
           (|nonnullstream| #3# |lineoftoks| (|nullstream|))))))
<5 (|next1|
    (|nonnullstream| #0=|incAppend1|
     (((((#2=(0 "1" 1 1 "strings") . 1) . "1"))
        ((|integer| (|posn| #2# . 0)) . "1")))
       (|nonnullstream| #3=|next1| |ncloopParse|
        (|nonnullstream| #0# NIL
         (|nonnullstream| #3# |lineoftoks| (|nullstream|))))))

5> (|incAppend1|
    (((((#0=(0 "1" 1 1 "strings") . 1) . "1"))
       ((|integer| (|posn| #0# . 0)) . "1")))
      (|nonnullstream| #1=|next1| |ncloopParse|
       (|nonnullstream| |incAppend1| NIL
        (|nonnullstream| #1# |lineoftoks| (|nullstream|)))))
  6> (|StreamNull|
      (((((#0=(0 "1" 1 1 "strings") . 1) . "1"))
         ((|integer| (|posn| #0# . 0)) . "1"))))
  <6 (|StreamNull| NIL)
  6> (|incAppend| NIL
      (|nonnullstream| #0=|next1| |ncloopParse|
       (|nonnullstream| |incAppend1| NIL
        (|nonnullstream| #0# |lineoftoks| (|nullstream|)))))
    7> (|Delay| #0=|incAppend1|
        (NIL
         (|nonnullstream| #2=|next1| |ncloopParse|
          (|nonnullstream| #0# NIL
           (|nonnullstream| #2# |lineoftoks| (|nullstream|))))))
    <7 (|Delay|
        (|nonnullstream| #0=|incAppend1| NIL
         (|nonnullstream| #2=|next1| |ncloopParse|
          (|nonnullstream| #0# NIL
           (|nonnullstream| #2# |lineoftoks| (|nullstream|))))))
  <6 (|incAppend|
      (|nonnullstream| #0=|incAppend1| NIL
       (|nonnullstream| #2=|next1| |ncloopParse|
```

```
          (|nonnullstream| #0# NIL
           (|nonnullstream| #2# |lineoftoks| (|nullstream|))))))
  <5 (|incAppend1|
      (((((#0=(0 "1" 1 1 "strings") . 1) . "1"))
        ((|integer| (|posn| #0# . 0)) . "1"))
        |nonnullstream| #1=|incAppend1| NIL
        (|nonnullstream| #3=|next1| |ncloopParse|
         (|nonnullstream| #1# NIL
          (|nonnullstream| #3# |lineoftoks| (|nullstream|))))))
<4 (|StreamNull| NIL)

4> (|pfAbSynOp?|
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1") |command|)
<4 (|pfAbSynOp?| NIL)

4> (|intloopSpadProcess| 1
    (((#0=(0 "1" 1 1 "strings") . 1) . "1"))
     ((|integer| (|posn| #0# . 0)) . "1") T)
  5> (|ncPutQ| (|carrier|) |stepNumber| 1)
    6> (|ncAlist| (|carrier|))
    <6 (|ncAlist| NIL)
    6> (|ncAlist| (|carrier|))
    <6 (|ncAlist| NIL)
    6> (|ncTag| (|carrier|))
    <6 (|ncTag| |carrier|)
  <5 (|ncPutQ| 1)
  5> (|ncPutQ| ((|carrier| (|stepNumber| . 1))) |messages| NIL)
    6> (|ncAlist| ((|carrier| (|stepNumber| . 1))))
    <6 (|ncAlist| ((|stepNumber| . 1)))
    6> (|ncAlist| ((|carrier| (|stepNumber| . 1))))
    <6 (|ncAlist| ((|stepNumber| . 1)))
    6> (|ncTag| ((|carrier| (|stepNumber| . 1))))
    <6 (|ncTag| |carrier|)
  <5 (|ncPutQ| NIL)
  5> (|ncPutQ|
      ((|carrier| (|messages|) (|stepNumber| . 1)))
      |lines| ((((0 "1" 1 1 "strings") . 1) . "1")))
    6> (|ncAlist| ((|carrier| (|messages|) (|stepNumber| . 1))))
    <6 (|ncAlist| ((|messages|) (|stepNumber| . 1)))
    6> (|ncAlist| ((|carrier| (|messages|) (|stepNumber| . 1))))
    <6 (|ncAlist| ((|messages|) (|stepNumber| . 1)))
    6> (|ncTag| ((|carrier| (|messages|) (|stepNumber| . 1))))
    <6 (|ncTag| |carrier|)
  <5 (|ncPutQ| ((((0 "1" 1 1 "strings") . 1) . "1")))
  5> (|intloopSpadProcess,interp|
      ((|carrier| (|lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
                  (|messages|) (|stepNumber| . 1)))
      ((|integer| (|posn| #0# . 0)) . "1") T)
    6> (|ncConversationPhase| |phParse|
        ((((|carrier| (|lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
                    (|messages|) (|stepNumber| . 1)))
          ((|integer| (|posn| #0# . 0)) . "1")))
      7> (|phParse|
          ((|carrier| (|lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
                    (|messages|) (|stepNumber| . 1)))
```

```
      ((|integer| (|posn| #0# . 0)) . "1"))
    8> (|ncPutQ|
        ((|carrier| (|lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1")
             ) (|messages|) (|stepNumber| . 1)))
         |ptree| ((|integer| (|posn| #0# . 0)) . "1"))
      9> (|ncAlist|
          ((|carrier| (|lines| (((0 "1" 1 1 "strings") . 1) . "1"))
                (|messages|) (|stepNumber| . 1))))
      <9 (|ncAlist|
          ((|lines| (((0 "1" 1 1 "strings") . 1) . "1"))
             (|messages|) (|stepNumber| . 1)))
      9> (|ncAlist|
          ((|carrier| (|lines| (((0 "1" 1 1 "strings") . 1) . "1"))
            (|messages|) (|stepNumber| . 1))))
      <9 (|ncAlist|
          ((|lines| (((0 "1" 1 1 "strings") . 1) . "1"))
            (|messages|) (|stepNumber| . 1)))
      9> (|ncTag|
          ((|carrier| (|lines| (((0 "1" 1 1 "strings") . 1) . "1"))
           (|messages|) (|stepNumber| . 1))))
      <9 (|ncTag| |carrier|)
    <8 (|ncPutQ|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
  <7 (|phParse| OK)
  7> (|ncConversationPhase,wrapup|
      ((|carrier|
        (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
         . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
  <7 (|ncConversationPhase,wrapup| NIL)
<6 (|ncConversationPhase| OK)
6> (|ncConversationPhase| |phMacro|
    (((|carrier|
       (|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1)))))
  7> (|phMacro|
      ((|carrier|
        (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
    8> (|ncEltQ|
        ((|carrier|
          (|ptree|
           (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
           (|lines| ((#0# . 1) . "1"))
           (|messages|)
```

```
      (|stepNumber| . 1))) |ptree|)
  9> (|ncAlist|
      ((|carrier|
       (|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
            . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
  <9 (|ncAlist|
      ((|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
           . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1)))
<8 (|ncEltQ|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
8> (|ncPutQ|
    ((|carrier|
      (|ptree| .
         #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
            . "1"))
      (|lines| ((#1# . 1) . "1"))
      (|messages|)
      (|stepNumber| . 1)))
     |ptreePremacro| #0#)
  9> (|ncAlist|
      ((|carrier|
       (|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
           . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
  <9 (|ncAlist|
      ((|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
          . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1)))
  9> (|ncAlist|
      ((|carrier|
       (|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
          . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
  <9 (|ncAlist|
      ((|ptree|
       (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
       (|lines| ((#0# . 1) . "1"))
```

```
      (|messages|)
      (|stepNumber| . 1)))
  9> (|ncTag|
      ((|carrier|
       (|ptree|
        (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
          . "1")
        (|lines| ((#0# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
  <9 (|ncTag| |carrier|)
<8 (|ncPutQ|
     ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
8> (|macroExpanded|
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
  9> (|macExpand|
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
    10> (|pfWhere?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1"))
    11> (|pfAbSynOp?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1") |Where|)
    <11 (|pfAbSynOp?| NIL)
    <10 (|pfWhere?| NIL)
    10> (|pfLambda?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1"))
    11> (|pfAbSynOp?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1") |Lambda|)
    <11 (|pfAbSynOp?| NIL)
    <10 (|pfLambda?| NIL)
    10> (|pfMacro?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1"))
    11> (|pfAbSynOp?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1") |Macro|)
    <11 (|pfAbSynOp?| NIL)
    <10 (|pfMacro?| NIL)
    10> (|pfId?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1"))
    11> (|pfAbSynOp?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1") |id|)
    <11 (|pfAbSynOp?| NIL)
    11> (|pfAbSynOp?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1") |idsy|)
    <11 (|pfAbSynOp?| NIL)
    <10 (|pfId?| NIL)
    10> (|pfApplication?|
```

```
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
             . "1"))
      11> (|pfAbSynOp?|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1") |Application|)
      <11 (|pfAbSynOp?| NIL)
      <10 (|pfApplication?| NIL)
      10> (|pfMapParts| |macExpand|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1"))
      11> (|pfLeaf?|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1"))
      12> (|pfAbSynOp|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1"))
      <12 (|pfAbSynOp| |integer|)
      <11 (|pfLeaf?| (|integer| |Document| |error|))
      <10 (|pfMapParts|
          ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1"))
    <9 (|macExpand|
        ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
  <8 (|macroExpanded|
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
  8> (|ncPutQ|
      ((|carrier|
        (|ptreePremacro| .
         #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
        (|ptree| . #0#)
        (|lines| ((#1# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))) |ptree| #0#)
    9> (|ncAlist|
        ((|carrier|
          (|ptreePremacro| .
            #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
          (|ptree| . #0#)
          (|lines| ((#1# . 1) . "1"))
          (|messages|)
          (|stepNumber| . 1))))
    <9 (|ncAlist|
        ((|ptreePremacro| .
          #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
               . "1"))
        (|ptree| . #0#)
        (|lines| ((#1# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1)))
  <8 (|ncPutQ|
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
<7 (|phMacro| OK)
```

```
  7> (|ncConversationPhase,wrapup|
      ((|carrier|
       (|ptreePremacro| .
         #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
           . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))))
  <7 (|ncConversationPhase,wrapup| NIL)
<6 (|ncConversationPhase| OK)
6> (|ncConversationPhase| |phIntReportMsgs|
    (((|carrier|
       (|ptreePremacro| .
        #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
             . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))) T))
  7> (|phIntReportMsgs|
      ((|carrier|
        (|ptreePremacro| .
         #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
        (|ptree| . #0#)
        (|lines| ((#1# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))) T)
    8> (|ncEltQ|
        ((|carrier|
          (|ptreePremacro| .
            #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                 . "1"))
          (|ptree| . #0#)
          (|lines| ((#1# . 1) . "1"))
          (|messages|)
          (|stepNumber| . 1))) |lines|)
      9> (|ncAlist|
          ((|carrier|
            (|ptreePremacro| .
             #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
               . "1"))
            (|ptree| . #0#)
            (|lines| ((#1# . 1) . "1"))
            (|messages|)
            (|stepNumber| . 1))))
      <9 (|ncAlist|
          ((|ptreePremacro| .
            #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                 . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
```

```
      (|stepNumber| . 1)))
<8 (|ncEltQ| ((((0 "1" 1 1 "strings") . 1) . "1")))
8> (|ncEltQ|
    ((|carrier| (|ptreePremacro| .
     #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) .
          "1"))
     (|ptree| . #0#)
     (|lines| ((#1# . 1) . "1"))
     (|messages|)
     (|stepNumber| . 1)))
     |messages|)
  9> (|ncAlist|
      ((|carrier|
       (|ptreePremacro| .
         #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
             . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))))
  <9 (|ncAlist|
      ((|ptreePremacro| .
       #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
          . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1)))
<8 (|ncEltQ| NIL)
8> (|ncPutQ|
    ((|carrier|
     (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
           . "1"))
     (|ptree| . #0#)
     (|lines| ((#1# . 1) . "1"))
     (|messages|)
     (|stepNumber| . 1))) |ok?| T)
  9> (|ncAlist|
      ((|carrier|
       (|ptreePremacro| .
        #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
           . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))))
  <9 (|ncAlist|
      ((|ptreePremacro| .
        #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
             . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
```

```
          (|stepNumber| . 1)))
      9> (|ncAlist|
          ((|carrier|
           (|ptreePremacro| .
             #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                 . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1))))
      <9 (|ncAlist|
          ((|ptreePremacro| .
            #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1)))
      9> (|ncTag|
          ((|carrier|
           (|ptreePremacro| .
             #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                 . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1))))
      <9 (|ncTag| |carrier|)
    <8 (|ncPutQ| T)
  <7 (|phIntReportMsgs| OK)
  7> (|ncConversationPhase,wrapup|
      ((|carrier|
        (|ok?| . T)
        (|ptreePremacro| .
          #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
        (|ptree| . #0#)
        (|lines| ((#1# . 1) . "1"))
        (|messages|)
        (|stepNumber| . 1))))
  <7 (|ncConversationPhase,wrapup| NIL)
<6 (|ncConversationPhase| OK)
6> (|ncConversationPhase| |phInterpret|
    (((|carrier|
      (|ok?| . T)
      (|ptreePremacro| .
       #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
           . "1"))
      (|ptree| . #0#)
      (|lines| ((#1# . 1) . "1"))
      (|messages|)
      (|stepNumber| . 1)))))
  7> (|phInterpret|
      ((|carrier|
```

```
   (|ok?| . T)
   (|ptreePremacro| .
    #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
          . "1"))
   (|ptree| . #0#)
   (|lines| ((#1# . 1) . "1"))
   (|messages|)
   (|stepNumber| . 1))))
8> (|ncEltQ|
    ((|carrier|
     (|ok?| . T)
     (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
           . "1"))
     (|ptree| . #0#)
     (|lines| ((#1# . 1) . "1"))
     (|messages|)
     (|stepNumber| . 1)))
    |ptree|)
  9> (|ncAlist|
      ((|carrier|
       (|ok?| . T)
       (|ptreePremacro| .
        #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))))
  <9 (|ncAlist|
      ((|ok?| . T)
       (|ptreePremacro| .
        #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1)))
<8 (|ncEltQ|
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
8> (|intInterpretPform|
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
  9> (|pf2Sex|
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
    10> (|pf2Sex1|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
              . "1"))
    11> (|pfNothing?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
           . "1"))
    12> (|pfAbSynOp?|
         ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
            . "1") |nothing|)
    <12 (|pfAbSynOp?| NIL)
```

```
  <11 (|pfNothing?| NIL)
  11> (|pfSymbol?|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
          . "1"))
  12> (|pfAbSynOp?|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
         . "1") |symbol|)
  <12 (|pfAbSynOp?| NIL)
  <11 (|pfSymbol?| NIL)
  11> (|pfLiteral?|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
         . "1"))
  12> (|pfAbSynOp|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
          . "1"))
  <12 (|pfAbSynOp| |integer|)
  <11 (|pfLiteral?|
       (|integer| |symbol| |expression| |one| |zero|
        |char| |string| |float|))
  11> (|pfLiteral2Sex|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
          . "1"))
  12> (|pfLiteralClass|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
         . "1"))
  13> (|pfAbSynOp|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
         . "1"))
  <13 (|pfAbSynOp| |integer|)
  <12 (|pfLiteralClass| |integer|)
  12> (|pfLiteralString|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
         . "1"))
  13> (|tokPart|
       ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
          . "1"))
  <13 (|tokPart| "1")
  <12 (|pfLiteralString| "1")
  <11 (|pfLiteral2Sex| 1)
  <10 (|pf2Sex1| 1)
<9 (|pf2Sex| 1)
9> (|zeroOneTran| 1)
<9 (|zeroOneTran| 1)
9> (|processInteractive| 1
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
  10> (PUT |algebra| |TimeTotal| 0.0)
  <10 (PUT 0.0)
  10> (PUT |algebra| |SpaceTotal| 0)
  <10 (PUT 0)
  10> (PUT |analysis| |TimeTotal| 0.0)
  <10 (PUT 0.0)
  10> (PUT |analysis| |SpaceTotal| 0)
  <10 (PUT 0)
  10> (PUT |coercion| |TimeTotal| 0.0)
```

```
<10 (PUT 0.0)
10> (PUT |coercion| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |compilation| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |compilation| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |debug| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |debug| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |evaluation| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |evaluation| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |gc| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |gc| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |history| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |history| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |instantiation| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |instantiation| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |load| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |load| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |modemaps| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |modemaps| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |optimization| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |optimization| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |querycoerce| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |querycoerce| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |other| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |other| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |diskread| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |diskread| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |print| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |print| |SpaceTotal| 0)
```

```
<10 (PUT 0)
10> (PUT |resolve| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |resolve| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |interpreter| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |interpreter| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (PUT |evaluation| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |evaluation| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (PUT |other| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |other| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (PUT |reclaim| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |reclaim| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (GETL |gc| |TimeTotal|)
<10 (GETL 0.0)
10> (PUT |gc| |TimeTotal| 0.050000000000000003)
<10 (PUT 0.050000000000000003)
10> (PUT |gc| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |gc| |SpaceTotal| 0)
<10 (PUT 0)
10> (|processInteractive1| 1
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
11> (recordFrame |system|)
12> (|diffAlist| NIL NIL)
<12 (|diffAlist| NIL)
<11 (recordFrame NIL)
11> (GETL |other| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |other| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (|interpretTopLevel| 1
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
12> (|interpret| 1
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
13> (|interpret1| 1 NIL
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
14> (|member| 1 (|noBranch| |noMapVal|))
<14 (|member| NIL)
14> (|member| 1 (|nil| |true| |false|))
<14 (|member| NIL)
14> (|member| |--immediateData--| NIL)
```

```
<14 (|member| NIL)
14> (|isDomainValuedVariable| |--immediateData--|)
<14 (|isDomainValuedVariable| NIL)
14> (GETDATABASE |--immediateData--| CONSTRUCTOR)
<14 (GETDATABASE NIL)
14> (GETDATABASE |--immediateData--| ABBREVIATION)
<14 (GETDATABASE NIL)
14> (|member| |--immediateData--|
     (|Record| |Union| |Enumeration|))
<14 (|member| NIL)
14> (|getProplist| |--immediateData--| ((NIL)))
15> (|search| |--immediateData--| ((NIL)))
16> (|searchCurrentEnv| |--immediateData--| (NIL))
<16 (|searchCurrentEnv| NIL)
16> (|searchTailEnv| |--immediateData--| NIL)
<16 (|searchTailEnv| NIL)
<15 (|search| NIL)
15> (|search| |--immediateData--|
     ((((|Category|
         (|modemap| ((((|Category|) (|Category|)) (T *))))
        (|Join|
         (|modemap|
          ((((|Category|)
            (|Category|)
            (|Category|)
            (|Category|))
           (T *))
          ((((|Category|)
            (|Category|)
            (|List| (|Category|))
            (|Category|))
           (T *))))))))
16> (|searchCurrentEnv| |--immediateData--|
      ((((|Category|
         (|modemap| ((((|Category|) (|Category|)) (T *))))
        (|Join|
         (|modemap|
          ((((|Category|)
            (|Category|)
            (|Category|)
            (|Category|))
           (T *))
          ((((|Category|)
            (|Category|)
            (|List| (|Category|))
            (|Category|))
           (T *)))))))
<16 (|searchCurrentEnv| NIL)
16> (|searchTailEnv| |--immediateData--| NIL)
<16 (|searchTailEnv| NIL)
<15 (|search| NIL)
<14 (|getProplist| NIL)
14> (|member| |--immediateData--| NIL)
<14 (|member| NIL)
```

```
14> (|member| |--immediateData--| NIL)
<14 (|member| NIL)
14> (|member| |--immediateData--| NIL)
<14 (|member| NIL)
14> (|member| |--immediateData--| NIL)
<14 (|member| NIL)
14> (|member| |--immediateData--| NIL)
<14 (|member| NIL)
14> (|interpret2|
     (#0=(|PositiveInteger|) . 1) #0#
      ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
         . "1"))
<14 (|interpret2| ((|PositiveInteger|) . 1))
<13 (|interpret1| ((|PositiveInteger|) . 1))
<12 (|interpret| ((|PositiveInteger|) . 1))
<11 (|interpretTopLevel| ((|PositiveInteger|) . 1))
11> (GETL |analysis| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |analysis| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (GETL |other| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |other| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (|recordAndPrint| 1 (|PositiveInteger|))

12> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (|member| (|PositiveInteger|)
      ((|Mode|) (|Domain|) (|SubDomain| (|Domain|))))
<12 (|member| NIL)
12> (|member| (|PositiveInteger|)
      ((|Category|) (|Mode|) (|Domain|)
       (|SubDomain| (|Domain|))))
<12 (|member| NIL)
12> (GETL |print| |TimeTotal|)
<12 (GETL 0.0)
12> (GETL |gc| |TimeTotal|)
<12 (GETL 0.0)
12> (PUT |gc| |TimeTotal| 0.0)
<12 (PUT 0.0)
12> (PUT |print| |TimeTotal| 0.0)
<12 (PUT 0.0)
12> (|isEqualOrSubDomain| (|PositiveInteger|)
                          (|OutputForm|))
<12 (|isEqualOrSubDomain| NIL)
```

```
12> (GETDATABASE |OutputForm| ABBREVIATION)
<12 (GETDATABASE OUTFORM)
12> (HPUT #<hash-table 0000000001ab1e40>
     (|OutputForm|) (1))
<12 (HPUT (1))
12> (HPUT #<hash-table 0000000001ab1ea0>
     (NIL NIL NIL) (1 . T))
<12 (HPUT (1 . T))
12> (HPUT #<hash-table 0000000001ab1ea0>
     (#0=(|OutputForm|) NIL NIL) (1 . #0#))
<12 (HPUT (1 |OutputForm|))
12> (HPUT #<hash-table 0000000001ab1e40>
     (|PositiveInteger|) (1))
<12 (HPUT (1))
12> (|member| (|OutputForm|) ((|Integer|) (|OutputForm|)))
<12 (|member| ((|OutputForm|)))
12> (|member| (|OutputForm|)
      ((|Mode|) (|Domain|) (|SubDomain| (|Domain|))))
<12 (|member| NIL)
12> (GETDATABASE |OutputForm| ABBREVIATION)
<12 (GETDATABASE OUTFORM)
12> (GETDATABASE |OutputForm| COSIG)
<12 (GETDATABASE (NIL))
12> (HPUT #<hash-table 0000000001ab1840>
     (|OutputForm|) (1 . T))
<12 (HPUT (1 . T))
12> (|isPartialMode| (|OutputForm|))
<12 (|isPartialMode| NIL)
12> (|member| |coerce| (= + * -))
<12 (|member| NIL)
12> (|isPartialMode| (|OutputForm|))
<12 (|isPartialMode| NIL)
12> (|member| |PositiveInteger|
     (|List| |Vector| |Stream| |FiniteSet| |Array|))
<12 (|member| NIL)
12> (|member| |PositiveInteger|
     (|Union| |Record| |Mapping| |Enumeration|))
<12 (|member| NIL)
12> (GETDATABASE |PositiveInteger| OPERATIONALIST)
<12 (GETDATABASE
     ((~= ((((|Boolean|) $ $) NIL))
      (|sample| (($) NIL T CONST))
      (|recip| ((((|Union| $ "failed") $) NIL))
      (|one?| ((((|Boolean|) $) NIL))
      (|min| (($ $ $) NIL))
      (|max| (($ $ $) NIL))
      (|latex| ((((|String|) $) NIL))
      (|hash| ((((|SingleInteger|) $) NIL))
      (|gcd| (($ $ $) NIL))
      (|coerce| ((((|OutputForm|) $) NIL))
      (^ (($ $ (|NonNegativeInteger|)) NIL)
         (($ $ (|PositiveInteger|)) NIL))
      (|One| (($) NIL T CONST))
      (>= ((((|Boolean|) $ $) NIL))
```

```
      (> (((|Boolean|) $ $) NIL))
      (= (((|Boolean|) $ $) NIL))
      (<= (((|Boolean|) $ $) NIL))
      (< (((|Boolean|) $ $) NIL))
      (+ (($ $ $) NIL))
      (** (($ $ (|NonNegativeInteger|)) NIL)
          (($ $ (|PositiveInteger|)) NIL))
      (* (($ (|PositiveInteger|) $) NIL) (($ $ $) NIL))))
12> (|constructSubst| (|PositiveInteger|))
<12 (|constructSubst| (($ |PositiveInteger|)))
12> (|isEqualOrSubDomain| #0=(|PositiveInteger|) #0#)
<12 (|isEqualOrSubDomain| T)
12> (|isEqualOrSubDomain| (|OutputForm|) (|OutputForm|))
<12 (|isEqualOrSubDomain| T)
12> (|member| |OutputForm| (|Union| |Record| |Mapping| |Enumeration|))
<12 (|member| NIL)
12> (GETDATABASE |OutputForm| OPERATIONALIST)
<12 (GETDATABASE
     ((~= ((((|Boolean|) $ $) NIL))
      (|zag| (($ $ $) 120))
      (|width| ((((|Integer|) $) 30) (((|Integer|)) 35))
      (|vspace| ((($ (|Integer|)) 47))
      (|vconcat| ((($ $ $) 48) ((($ (|List| $)) 80))
      (|supersub| ((($ $ (|List| $)) 78))
      (|superHeight| ((((|Integer|) $) 33))
      (|super| ((($ $ $) 66))
      (|sum| ((($ $) 135) ((($ $ $) 136) ((($ $ $ $) 137))
      (|subHeight| ((((|Integer|) $) 32))
      (|sub| ((($ $ $) 65))
      (|string| ((($ $) 109))
      (|slash| ((($ $ $) 124))
      (|semicolonSeparate| ((($ (|List| $)) 55))
      (|scripts| ((($ $ (|List| $)) 72))
      (|rspace| ((($ (|Integer|) (|Integer|)) 49))
      (|root| ((($ $) 121) ((($ $ $) 122))
      (|right| ((($ $ (|Integer|)) 42) ((($ $) 45))
      (|rem| ((($ $ $) 93))
      (|rarrow| ((($ $ $) 127))
      (|quote| ((($ $) 110))
      (|quo| ((($ $ $) 94))
      (|prod| ((($ $) 138) ((($ $ $) 139) ((($ $ $ $) 140))
      (|print| ((((|Void|) $) 8))
      (|prime| ((($ $) 113) ((($ $ (|NonNegativeInteger|)) 117))
      (|presuper| ((($ $ $) 68))
      (|presub| ((($ $ $) 67))
      (|prefix| ((($ $ (|List| $)) 104))
      (|postfix| ((($ $ $) 108))
      (|pile| ((($ (|List| $)) 53))
      (|paren| ((($ $) 63) ((($ (|List| $)) 64))
      (|overlabel| ((($ $ $) 118))
      (|overbar| ((($ $) 111))
      (|over| ((($ $ $) 123))
      (|outputForm| ((($ (|Integer|)) 20)
                     ((($ (|Symbol|)) 22)
```

```
                    (($ (|String|)) 29)
                    (($ (|DoubleFloat|)) 24))
          (|or| (($ $ $) 97))
          (|not| (($ $) 98))
          (|messagePrint| ((((|Void|) (|String|)) 14))
          (|message| (($ (|String|)) 13))
          (|matrix| (($ (|List| (|List| $))) 51))
          (|left| (($ $ (|Integer|)) 41) (($ $) 44))
          (|latex| ((((|String|) $) NIL))
          (|label| (($ $ $) 126))
          (|int| (($ $) 141) (($ $ $) 142) (($ $ $ $) 143))
          (|infix?| ((((|Boolean|) $) 102))
          (|infix| (($ $ (|List| $)) 106) (($ $ $ $) 107))
          (|hspace| (($ (|Integer|)) 38))
          (|height| ((((|Integer|) $) 31) ((((|Integer|)) 34))
          (|hconcat| (($ $ $) 39) (($ (|List| $)) 79))
          (|hash| ((((|SingleInteger|) $) NIL))
          (|exquo| (($ $ $) 95))
          (|empty| (($) 12))
          (|elt| (($ $ (|List| $)) 103))
          (|dot| (($ $) 112)
                 (($ $ (|NonNegativeInteger|)) 116))
          (|div| (($ $ $) 92))
          (|differentiate| (($ $ (|NonNegativeInteger|)) 134))
          (|commaSeparate| (($ (|List| $)) 54))
          (|coerce| ((((|OutputForm|) $) 18))
          (|center| (($ $ (|Integer|)) 40) (($ $) 43))
          (|bracket| (($ $) 61) (($ (|List| $)) 62))
          (|brace| (($ $) 59) (($ (|List| $)) 60))
          (|box| (($ $) 119))
          (|blankSeparate| (($ (|List| $)) 58))
          (|binomial| (($ $ $) 101))
          (|assign| (($ $ $) 125))
          (|and| (($ $ $) 96))
          (^= (($ $ $) 81))
          (SEGMENT (($ $ $) 99) (($ $) 100))
          (>= (($ $ $) 85))
          (> (($ $ $) 83))
          (= ((((|Boolean|) $ $) 15) (($ $ $) 16))
          (<= (($ $ $) 84))
          (< (($ $ $) 82))
          (/ (($ $ $) 90))
          (- (($ $ $) 87) (($ $) 88))
          (+ (($ $ $) 86))
          (** (($ $ $) 91))
          (* (($ $ $) 89))))
12> (|constructSubst| (|OutputForm|))
<12 (|constructSubst| (($ |OutputForm|)))
12> (|isEqualOrSubDomain|
      (|PositiveInteger|) (|OutputForm|))
<12 (|isEqualOrSubDomain| NIL)
12> (HPUT #<hash-table 0000000001ab1e70>
     (|coerce| (|OutputForm|) (#0=(|PositiveInteger|))
      (#0#) NIL) (1 ((#0# #1=(|OutputForm|) #0#)
```

```
      (#1# $) (NIL))))
<12 (HPUT (1 ((#0=(|PositiveInteger|)
      #1=(|OutputForm|) #0#) (#1# $) (NIL))))
12> (HPUT #<hash-table 0000000001ab1f00>
     (|coerce| #0=(|PositiveInteger|) (|OutputForm|))
     (1 (#0# #1=(|OutputForm|) #0#) (#1# $) (NIL)))
<12 (HPUT (1 (#0=(|PositiveInteger|)
     #1=(|OutputForm|) #0#) (#1# $) (NIL)))
12> (|evalDomain| (|PositiveInteger|))
13> (GETL |print| |TimeTotal|)
<13 (GETL 0.0)
13> (GETL |gc| |TimeTotal|)
<13 (GETL 0.0)
13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |print| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (|mkEvalable| (|PositiveInteger|))
14> (CANFUNCALL? |PositiveInteger|)
<14 (CANFUNCALL? T)
14> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (GETDATABASE |PositiveInteger| COSIG)
<14 (GETDATABASE (NIL))
<13 (|mkEvalable| (|PositiveInteger|))
13> (GETL |PositiveInteger| LOADED)
<13 (GETL NIL)
13> (|loadLib| |PositiveInteger|)
14> (GETL |instantiation| |TimeTotal|)
<14 (GETL 0.0)
14> (GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> (PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (PUT |instantiation| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (GETDATABASE |PositiveInteger| OBJECT)
<14 (GETDATABASE
     "/home/daly/noise/mnt/ubuntu/algebra/PI.o")
14> (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/PI.o")
15> (|pathname|
     "/home/daly/noise/mnt/ubuntu/algebra/PI.o")
<15 (|pathname|
      #p"/home/daly/noise/mnt/ubuntu/algebra/PI.o")
<14 (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
14> (|isSystemDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
<14 (|isSystemDirectory| T)
14> (|loadLibNoUpdate| |PositiveInteger| |PositiveInteger|
     "/home/daly/noise/mnt/ubuntu/algebra/PI.o")
15> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
```

```
15> (|getProplist| |NonNegativeInteger|
     ((((|Category| (|modemap|
          (((|Category|) (|Category|)) (T *))))
        (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
           (|Category|))
          (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
           (|Category|)) (T *))))))))
16> (|search| |NonNegativeInteger|
     ((((|Category| (|modemap|
         (((|Category|) (|Category|)) (T *))))
       (|Join| (|modemap|
        (((|Category|) (|Category|) (|Category|)
          (|Category|))
         (T *))
        (((|Category|) (|Category|) (|List| (|Category|))
          (|Category|)) (T *))))))))
17> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Category| (|modemap|
        (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|))
        (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<17 (|searchCurrentEnv| NIL)
17> (|searchTailEnv| |NonNegativeInteger| NIL)
<17 (|searchTailEnv| NIL)
<16 (|search| NIL)
16> (|search| |NonNegativeInteger|
     ((((|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
     (|Join| (|modemap|
      (((|Category|) (|Category|) (|Category|)
        (|Category|))
       (T *))
      (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
17> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Category| (|modemap|
        (((|Category|) (|Category|)) (T *))))
       (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
           (|Category|)) (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
           (|Category|)) (T *)))))))
<17 (|searchCurrentEnv| NIL)
17> (|searchTailEnv| |NonNegativeInteger| NIL)
<17 (|searchTailEnv| NIL)
<16 (|search| NIL)
<15 (|getProplist| NIL)
15> (|addBinding| |NonNegativeInteger|
```

```
     ((|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
     ((((|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *))))))))
16> (|getProplist| |NonNegativeInteger|
     ((((|Category| (|modemap|
      (((|Category|) (|Category|)) (T *))))
     (|Join| (|modemap|
      (((|Category|) (|Category|) (|Category|)
        (|Category|)) (T *))
      (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
17> (|search| |NonNegativeInteger|
     ((((|Category| (|modemap|
      (((|Category|) (|Category|)) (T *))))
     (|Join| (|modemap|
      (((|Category|) (|Category|) (|Category|)
        (|Category|)) (T *))
      (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
18> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Category| (|modemap|
        (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
        (((|Category|) (|Category|) (|Category|)
          (|Category|)) (T *))
        (((|Category|) (|Category|) (|List| (|Category|))
          (|Category|)) (T *)))))))
<18 (|searchCurrentEnv| NIL)
18> (|searchTailEnv| |NonNegativeInteger| NIL)
<18 (|searchTailEnv| NIL)
<17 (|search| NIL)
17> (|search| |NonNegativeInteger|
     ((((|Category| (|modemap|
      (((|Category|) (|Category|)) (T *))))
     (|Join| (|modemap|
      (((|Category|) (|Category|) (|Category|)
        (|Category|)) (T *))
      (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
18> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<18 (|searchCurrentEnv| NIL)
```

```
18> (|searchTailEnv| |NonNegativeInteger| NIL)
<18 (|searchTailEnv| NIL)
<17 (|search| NIL)
<16 (|getProplist| NIL)
16> (|addBindingInteractive| |NonNegativeInteger|
     ((|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
     ((((|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *))))))))
<16 (|addBindingInteractive|
     ((((|NonNegativeInteger|
      (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
        (((|Category|) (|Category|)) (T *))))
       (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
           (|Category|)) (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
           (|Category|)) (T *))))))))
<15 (|addBinding|
     ((((|NonNegativeInteger|
      (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
        (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
        (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
15> (|getProplist| |PositiveInteger|
     ((((|NonNegativeInteger|
       (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
       (|Category| (|modemap|
          (((|Category|) (|Category|)) (T *))))
       (|Join| (|modemap|
          (((|Category|) (|Category|) (|Category|)
            (|Category|)) (T *))
          (((|Category|) (|Category|) (|List| (|Category|))
            (|Category|)) (T *))))))))
16> (|search| |PositiveInteger|
     ((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
     (|Category| (|modemap|
      (((|Category|) (|Category|)) (T *))))
     (|Join| (|modemap|
```

```
      (((|Category|) (|Category|) (|Category|)
        (|Category|)) (T *))
      (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
17> (|searchCurrentEnv| |PositiveInteger|
     (((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<17 (|searchCurrentEnv| NIL)
17> (|searchTailEnv| |PositiveInteger| NIL)
<17 (|searchTailEnv| NIL)
<16 (|search| NIL)
16> (|search| |PositiveInteger|
     (((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
17> (|searchCurrentEnv| |PositiveInteger|
     (((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<17 (|searchCurrentEnv| NIL)
17> (|searchTailEnv| |PositiveInteger| NIL)
<17 (|searchTailEnv| NIL)
<16 (|search| NIL)
<15 (|getProplist| NIL)
15> (|addBinding| |PositiveInteger|
     ((|SuperDomain| |NonNegativeInteger|))
     (((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
```

```
      (((|Category|) (|Category|) (|Category|)
        (|Category|)) (T *))
      (((|Category|) (|Category|) (|List| (|Category|))
        (|Category|)) (T *))))))))
16> (|getProplist| |PositiveInteger|
     ((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *))))))))
17> (|search| |PositiveInteger|
     ((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *))))))))
18> (|searchCurrentEnv| |PositiveInteger|
     (((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<18 (|searchCurrentEnv| NIL)
18> (|searchTailEnv| |PositiveInteger| NIL)
<18 (|searchTailEnv| NIL)
<17 (|search| NIL)
17> (|search| |PositiveInteger|
     ((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       (((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       (((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       (((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *))))))))
18> (|searchCurrentEnv| |PositiveInteger|
     (((|NonNegativeInteger|
```

```
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       ((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       ((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       ((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<18 (|searchCurrentEnv| NIL)
18> (|searchTailEnv| |PositiveInteger| NIL)
<18 (|searchTailEnv| NIL)
<17 (|search| NIL)
<16 (|getProplist| NIL)
16> (|addBindingInteractive| |PositiveInteger|
     ((|SuperDomain| |NonNegativeInteger|))
     ((((|NonNegativeInteger|
      (|SubDomain|
       (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
      (|Category| (|modemap|
       ((|Category|) (|Category|)) (T *))))
      (|Join| (|modemap|
       ((|Category|) (|Category|) (|Category|)
         (|Category|)) (T *))
       ((|Category|) (|Category|) (|List| (|Category|))
         (|Category|)) (T *)))))))
<16 (|addBindingInteractive|
     ((((|PositiveInteger|
      (|SuperDomain| |NonNegativeInteger|))
      (|NonNegativeInteger|
       (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
       (|Category| (|modemap|
        ((|Category|) (|Category|)) (T *))))
       (|Join| (|modemap|
        ((|Category|) (|Category|) (|Category|)
          (|Category|)) (T *))
        ((|Category|) (|Category|) (|List| (|Category|))
          (|Category|)) (T *)))))))
<15 (|addBinding|
     ((((|PositiveInteger|
      (|SuperDomain| |NonNegativeInteger|))
      (|NonNegativeInteger|
       (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
       (|Category| (|modemap|
        ((|Category|) (|Category|)) (T *))))
       (|Join| (|modemap|
        ((|Category|) (|Category|) (|Category|)
          (|Category|)) (T *))
        ((|Category|) (|Category|) (|List| (|Category|))
          (|Category|)) (T *)))))))
15> (|makeByteWordVec2| 1 (0 0 0 0 0 0 0))
<15 (|makeByteWordVec2| #<bit-vector 0000000001ab1db0>)
```

```
15> (|makeByteWordVec2| 12 (2 5 6 0 0 7 2 0 6 0 0 1 0 0 0
      1 1 0 9 0 1 1 0 6 0 1 2 0 0 0 0 1 2 0 0 0 0 1 1 0 11
      0 1 1 0 10 0 1 2 0 0 0 0 1 1 0 12 0 1 2 0 0 0 8 1 2
      0 0 0 5 1 0 0 0 1 2 0 6 0 0 1 2 0 6 0 0 1 2 0 6 0 0
      1 2 0 6 0 0 1 2 0 6 0 0 1 2 0 0 0 0 1 2 0 0 0 8 1 2
      0 0 0 5 1 2 0 0 0 0 1 2 0 0 8 0 1))
<15 (|makeByteWordVec2| #<vector 0000000001ab1d80>)
15> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
15> (GETL |load| |TimeTotal|)
<15 (GETL 0.0)
15> (GETL |gc| |TimeTotal|)
<15 (GETL 0.0)
15> (PUT |gc| |TimeTotal| 0.0)
<15 (PUT 0.0)
15> (PUT |load| |TimeTotal| 0.0)
<15 (PUT 0.0)
<14 (|loadLibNoUpdate| T)
<13 (|loadLib| T)
13> (HPUT #<hash-table 000000000105e810> |PositiveInteger|
      ((NIL 1 . #<vector 0000000001ab1d50>)))
<13 (HPUT ((NIL 1 . #<vector 0000000001ab1d50>)))
13> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<13 (GETDATABASE |domain|)
13> (GETL |PositiveInteger| |infovec|)
<13 (GETL (#<vector 0000000000fa5db0>
           #<vector 0000000000fa5cf0>
     ((|commutative| "*") . 0))
     (#<bit-vector 0000000001ab1db0>
      #<vector 0000000000fa5c90>
      #<vector 0000000001ab1de0>
      . #<vector 0000000001ab1d80>) |lookupComplete|))
13> (HPUT #<hash-table 000000000105e810> |PositiveInteger|
      ((NIL 1 . #<vector 0000000001ab1d50>)))
<13 (HPUT ((NIL 1 . #<vector 0000000001ab1d50>)))
13> (GETL |instantiation| |TimeTotal|)
<13 (GETL 0.0)
13> (GETL |gc| |TimeTotal|)
<13 (GETL 0.0)
13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |instantiation| |TimeTotal| 0.0)
<13 (PUT 0.0)
<12 (|evalDomain| #<vector 0000000001ab1d50>)
12> (|compiledLookup| |coerce| ((|OutputForm|) $)
      #<vector 0000000001ab1d50>)
13> (|NRTevalDomain| #<vector 0000000001ab1d50>)
14> (|evalDomain| #<vector 0000000001ab1d50>)
15> (GETL |print| |TimeTotal|)
<15 (GETL 0.0)
15> (GETL |gc| |TimeTotal|)
<15 (GETL 0.0)
15> (PUT |gc| |TimeTotal| 0.0)
<15 (PUT 0.0)
```

```
15> (PUT |print| |TimeTotal| 0.0)
<15 (PUT 0.0)
15> (|mkEvalable| #<vector 0000000001ab1d50>)
<15 (|mkEvalable| #<vector 0000000001ab1d50>)
15> (GETL |instantiation| |TimeTotal|)
<15 (GETL 0.0)
15> (GETL |gc| |TimeTotal|)
<15 (GETL 0.0)
15> (PUT |gc| |TimeTotal| 0.0)
<15 (PUT 0.0)
15> (PUT |instantiation| |TimeTotal| 0.0)
<15 (PUT 0.0)
<14 (|evalDomain| #<vector 0000000001ab1d50>)
<13 (|NRTevalDomain| #<vector 0000000001ab1d50>)
13> (|basicLookup| |coerce| ((|OutputForm|) $)
     #<vector 0000000001ab1d50> #<vector 0000000001ab1d50>)
14> (|oldCompLookup| |coerce| ((|OutputForm|) $)
     #<vector 0000000001ab1d50> #<vector 0000000001ab1d50>)
15> (|lookupInDomainVector| |coerce| ((|OutputForm|) $)
     #<vector 0000000001ab1d50> #<vector 0000000001ab1d50>)
16> (GETDATABASE |OutputForm| COSIG)
<16 (GETDATABASE (NIL))
16> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<16 (GETDATABASE |domain|)
16> (GETL |NonNegativeInteger| LOADED)
<16 (GETL NIL)
16> (|loadLib| |NonNegativeInteger|)
17> (GETL |print| |TimeTotal|)
<17 (GETL 0.0)
17> (GETL |gc| |TimeTotal|)
<17 (GETL 0.0)
17> (PUT |gc| |TimeTotal| 0.0)
<17 (PUT 0.0)
17> (PUT |print| |TimeTotal| 0.0)
<17 (PUT 0.0)
17> (GETDATABASE |NonNegativeInteger| OBJECT)
<17 (GETDATABASE
     "/home/daly/noise/mnt/ubuntu/algebra/NNI.o")
17> (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/NNI.o")
18> (|pathname|
     "/home/daly/noise/mnt/ubuntu/algebra/NNI.o")
<18 (|pathname|
     #p"/home/daly/noise/mnt/ubuntu/algebra/NNI.o")
<17 (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
17> (|isSystemDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
<17 (|isSystemDirectory| T)
17> (|loadLibNoUpdate| |NonNegativeInteger|
     |NonNegativeInteger|
     "/home/daly/noise/mnt/ubuntu/algebra/NNI.o")
18> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<18 (GETDATABASE |domain|)
```

```
18> (|getProplist| |Integer| ((NIL)))
19> (|search| |Integer| ((NIL)))
20> (|searchCurrentEnv| |Integer| (NIL))
<20 (|searchCurrentEnv| NIL)
20> (|searchTailEnv| |Integer| NIL)
<20 (|searchTailEnv| NIL)
<19 (|search| NIL)
19> (|search| |Integer| ((NIL)))
20> (|searchCurrentEnv| |Integer| (NIL))
<20 (|searchCurrentEnv| NIL)
20> (|searchTailEnv| |Integer| NIL)
<20 (|searchTailEnv| NIL)
<19 (|search| NIL)
<18 (|getProplist| NIL)
18> (|addBinding| |Integer|
     ((|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))) ((NIL)))
19> (|getProplist| |Integer| ((NIL)))
20> (|search| |Integer| ((NIL)))
21> (|searchCurrentEnv| |Integer| (NIL))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |Integer| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
20> (|search| |Integer| ((NIL)))
21> (|searchCurrentEnv| |Integer| (NIL))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |Integer| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
<19 (|getProplist| NIL)
19> (|addBindingInteractive| |Integer|
     ((|SubDomain|
      (|NonNegativeInteger|
       COND
        ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))) ((NIL)))
<19 (|addBindingInteractive|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
<18 (|addBinding|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
```

```
18> (|getProplist| |NonNegativeInteger|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
         COND
          ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T)))))))))
19> (|search| |NonNegativeInteger|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
20> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
<20 (|searchCurrentEnv| NIL)
20> (|searchTailEnv| |NonNegativeInteger| NIL)
<20 (|searchTailEnv| NIL)
<19 (|search| NIL)
19> (|search| |NonNegativeInteger|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
20> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
<20 (|searchCurrentEnv| NIL)
20> (|searchTailEnv| |NonNegativeInteger| NIL)
<20 (|searchTailEnv| NIL)
<19 (|search| NIL)
<18 (|getProplist| NIL)
18> (|addBinding| |NonNegativeInteger|
     ((|SuperDomain| |Integer|))
      ((((|Integer|
       (|SubDomain|
        (|NonNegativeInteger|
         COND
          ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T)))))))))
19> (|getProplist| |NonNegativeInteger|
     ((((|Integer|
```

```
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
20> (|search| |NonNegativeInteger|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
21> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |NonNegativeInteger| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
20> (|search| |NonNegativeInteger|
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
21> (|searchCurrentEnv| |NonNegativeInteger|
     (((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |NonNegativeInteger| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
<19 (|getProplist| NIL)
19> (|addBindingInteractive| |NonNegativeInteger|
     ((|SuperDomain| |Integer|))
     ((((|Integer|
      (|SubDomain|
       (|NonNegativeInteger|
        COND
         ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))))))))
<19 (|addBindingInteractive|
     ((((|NonNegativeInteger| (|SuperDomain| |Integer|))
     (|Integer|
      (|SubDomain|
```

```
       (|NonNegativeInteger|
         COND
          ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T)))))))))
<18 (|addBinding|
     (((((|NonNegativeInteger| (|SuperDomain| |Integer|))
      (|Integer|
       (|SubDomain|
        (|NonNegativeInteger|
         COND
          ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T)))))))))
18> (|makeByteWordVec2| 1 (0 0 0 0 0 0 0 0 0 0 0 0 0))
<18 (|makeByteWordVec2| #<bit-vector 0000000001ab1990>)
18> (|makeByteWordVec2| 18 (2 5 6 0 0 7 2 5 0 0 0 10 2 0 6
       0 0 1 1 0 6 0 1 2 0 0 0 0 8 2 0 11 0 0 12 2 0 0 0 5
       9 0 0 0 1 2 0 0 0 0 1 1 0 11 0 1 1 0 0 0 1 2 0 0 0
       0 1 1 0 6 0 1 2 0 0 0 0 1 2 0 0 0 0 1 1 0 17 0 1 1
       0 16 0 1 2 0 0 0 0 1 2 0 11 0 0 1 2 0 13 0 0 1 1 0
       18 0 1 2 0 0 0 14 1 2 0 0 0 15 1 0 0 0 1 0 0 0 1 2
       0 6 0 0 1 2 0 6 0 0 1 2 0 6 0 0 1 2 0 6 0 0 1 2 0
       6 0 0 1 2 0 0 0 0 1 2 0 0 0 14 1 2 0 0 0 15 1 2 0
       0 0 0 1 2 0 0 14 0 1 2 0 0 15 0 1))
<18 (|makeByteWordVec2| #<vector 0000000001ab1960>)
18> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<18 (GETDATABASE |domain|)
18> (GETL |load| |TimeTotal|)
<18 (GETL 0.0)
18> (GETL |gc| |TimeTotal|)
<18 (GETL 0.0)
18> (PUT |gc| |TimeTotal| 0.0)
<18 (PUT 0.0)
18> (PUT |load| |TimeTotal| 0.0)
<18 (PUT 0.0)
<17 (|loadLibNoUpdate| T)
<16 (|loadLib| T)
16> (HPUT #<hash-table 000000000105e810>
     |NonNegativeInteger|
     ((NIL 1 . #<vector 0000000001ab1930>)))
<16 (HPUT ((NIL 1 . #<vector 0000000001ab1930>)))
16> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<16 (GETDATABASE |domain|)
16> (GETL |NonNegativeInteger| |infovec|)
<16 (GETL (#<vector 0000000001ab1b10>
           #<vector 0000000001ab1ae0>
           ((|commutative| "*") . 0))
           (#<bit-vector 0000000001ab1990>
            #<vector 0000000001ab1ab0>
            #<vector 0000000001ab1a80>
            . #<vector 0000000001ab1960>)
      |lookupComplete|))
16> (HPUT #<hash-table 000000000105e810>
          |NonNegativeInteger|
          ((NIL 1 . #<vector 0000000001ab1930>)))
```

```
<16 (HPUT ((NIL 1 . #<vector 0000000001ab1930>)))
16> (|lookupInDomainVector| |coerce| ((|OutputForm|) $)
     #<vector 0000000001ab1930> #<vector 0000000001ab1d50>)
17> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<17 (GETDATABASE |domain|)
17> (PNAME |NonNegativeInteger|)
<17 (PNAME "NonNegativeInteger")
17> (PNAME |NonNegativeInteger|)
<17 (PNAME "NonNegativeInteger")
17> (GETDATABASE |OutputForm| COSIG)
<17 (GETDATABASE (NIL))
17> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<17 (GETDATABASE |domain|)
17> (GETL |Integer| LOADED)
<17 (GETL NIL)
17> (|loadLib| |Integer|)
18> (GETL |print| |TimeTotal|)
<18 (GETL 0.0)
18> (GETL |gc| |TimeTotal|)
<18 (GETL 0.0)
18> (PUT |gc| |TimeTotal| 0.0)
<18 (PUT 0.0)
18> (PUT |print| |TimeTotal| 0.0)
<18 (PUT 0.0)
18> (GETDATABASE |Integer| OBJECT)
<18 (GETDATABASE
     "/home/daly/noise/mnt/ubuntu/algebra/INT.o")
18> (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/INT.o")
19> (|pathname|
     "/home/daly/noise/mnt/ubuntu/algebra/INT.o")
<19 (|pathname|
     #p"/home/daly/noise/mnt/ubuntu/algebra/INT.o")
<18 (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
18> (|isSystemDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
<18 (|isSystemDirectory| T)
18> (|loadLibNoUpdate| |Integer| |Integer|
     "/home/daly/noise/mnt/ubuntu/algebra/INT.o")
19> (GETDATABASE |Integer| CONSTRUCTORKIND)
<19 (GETDATABASE |domain|)
19> (|makeByteWordVec2| 1
     (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
<19 (|makeByteWordVec2| #<bit-vector 0000000001723f60>)
19> (|makeByteWordVec2| 133
     (1 7 6 0 8 3 7 6 0 9 9 10 2 7 6 0 11 12 1 7 6 0 13 0
      14 0 15 2 7 0 9 14 16 1 7 6 0 17 1 7 6 0 18 1 7 6 0
      19 1 35 0 11 36 1 44 0 11 45 1 47 0 11 48 1 50 0 11
      51 1 9 0 11 53 2 93 90 91 92 94 1 97 95 96 98 1 96
      0 0 99 1 96 2 0 100 1 101 95 96 102 1 96 0 2 103 1
      0 104 0 105 2 108 95 106 107 109 2 110 95 95 95 111
      1 101 95 96 112 1 96 21 0 113 1 96 0 0 114 1 116 96
```

```
      115 117 2 0 21 0 0 1 1 0 21 0 25 1 0 87 0 88 1 0 0
      0 89 1 0 21 0 1 2 0 0 0 0 1 2 0 83 0 0 1 3 0 0 0 0
      0 42 1 0 0 0 1 1 0 104 0 1 2 0 21 0 0 1 1 0 11 0 1
      2 0 0 0 0 82 0 0 0 1 1 0 124 0 1 1 0 11 0 1 2 0 0 0
      0 81 2 0 60 58 61 62 1 0 57 58 59 1 0 83 0 85 1 0
      120 0 1 1 0 21 0 1 1 0 121 0 1 1 0 0 0 65 0 0 0 64
      2 0 0 0 0 80 1 0 127 126 1 1 0 21 0 1 3 0 0 0 0 0 1
      2 0 0 0 0 56 1 0 21 0 1 2 0 0 0 0 1 3 0 122 0 123
      122 1 1 0 21 0 26 1 0 21 0 75 1 0 83 0 1 1 0 21 0
      34 2 0 125 126 0 1 3 0 0 0 0 0 43 2 0 0 0 0 77 2 0
      0 0 0 76 1 0 0 0 1 1 0 0 0 40 2 0 131 0 0 1 1 0 0
      126 1 2 0 0 0 0 1 1 0 9 0 55 2 0 0 0 0 1 0 0 0 1 1
      0 0 0 31 1 0 0 0 33 1 0 133 0 1 2 0 118 118 118 119
      2 0 0 0 0 86 1 0 0 126 1 1 0 0 0 1 1 0 104 0 105 3
      0 129 0 0 0 1 2 0 130 0 0 1 2 0 83 0 0 84 2 0 125
      126 0 1 1 0 21 0 1 1 0 73 0 1 2 0 78 0 0 79 1 0 0 0
      1 2 0 0 0 73 1 1 0 0 0 32 1 0 0 0 30 1 0 9 0 54 1 0
      47 0 49 1 0 44 0 46 1 0 50 0 52 1 0 123 0 1 1 0 11
      0 39 1 0 0 11 38 1 0 0 0 1 1 0 0 11 38 1 0 35 0 37
      0 0 73 1 2 0 21 0 0 1 2 0 0 0 0 1 0 0 0 29 2 0 21 0
      0 1 3 0 0 0 0 0 41 1 0 0 0 63 2 0 0 0 73 1 2 0 0 0
      132 1 0 0 0 27 0 0 0 28 3 0 6 7 0 21 24 2 0 9 0 21
      22 2 0 6 7 0 23 1 0 9 0 20 1 0 0 0 1 2 0 0 0 73 1 2
      0 21 0 0 1 2 0 21 0 0 1 2 0 21 0 0 66 2 0 21 0 0 1
      2 0 21 0 0 67 2 0 0 0 0 70 1 0 0 0 68 2 0 0 0 0 69
      2 0 0 0 73 74 2 0 0 0 132 1 2 0 0 0 0 71 2 0 0 11 0
      72 2 0 0 73 0 1 2 0 0 132 0 1))
<19 (|makeByteWordVec2| #<vector 0000000001723f00>)
19> (GETDATABASE |Integer| CONSTRUCTORKIND)
<19 (GETDATABASE |domain|)
19> (GETL |load| |TimeTotal|)
<19 (GETL 0.0)
19> (GETL |gc| |TimeTotal|)
<19 (GETL 0.0)
19> (PUT |gc| |TimeTotal| 0.0)
<19 (PUT 0.0)
19> (PUT |load| |TimeTotal| 0.0)
<19 (PUT 0.0)
<18 (|loadLibNoUpdate| T)
<17 (|loadLib| T)
17> (HPUT #<hash-table 000000000105e810> |Integer|
     ((NIL 1 . #<vector 0000000001723ed0>)))
<17 (HPUT ((NIL 1 . #<vector 0000000001723ed0>)))
17> (GETDATABASE |Integer| CONSTRUCTORKIND)
<17 (GETDATABASE |domain|)
17> (GETL |Integer| |infovec|)
<17 (GETL (#<vector 0000000001ab1780>
           #<vector 0000000001ab1750> ((|infinite| . 0)
           (|noetherian| . 0) (|canonicalsClosed| . 0)
           (|canonical| . 0) (|canonicalUnitNormal| . 0)
           (|multiplicativeValuation| . 0)
           (|noZeroDivisors| . 0)
           ((|commutative| "*") . 0) (|rightUnitary| . 0)
           (|leftUnitary| . 0) (|unitsKnown| . 0))
```

```
                    (#<bit-vector 0000000001723f60>
                     #<vector 0000000001723fc0>
                     #<vector 0000000001723f90>
                     . #<vector 0000000001723f00>)
                  |lookupComplete|))
              17> (HPUT #<hash-table 000000000105e810> |Integer|
                   ((NIL 1 . #<vector 0000000001723ed0>)))
              <17 (HPUT ((NIL 1 . #<vector 0000000001723ed0>)))
              17> (|lookupInDomainVector| |coerce| ((|OutputForm|) $)
                   #<vector 0000000001723ed0> #<vector 0000000001ab1d50>)
              18> (GETDATABASE |Integer| CONSTRUCTORKIND)
              <18 (GETDATABASE |domain|)
              18> (PNAME |Integer|)
              <18 (PNAME "Integer")
              18> (PNAME |Integer|)
              <18 (PNAME "Integer")
              18> (GETDATABASE |OutputForm| COSIG)
              <18 (GETDATABASE (NIL))
              18> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
              <18 (GETDATABASE |domain|)
              <17 (|lookupInDomainVector|
                   (#<compiled-function |INT;coerce;$Of;16|>
                    . #<vector 0000000001723ed0>))
              <16 (|lookupInDomainVector|
                   (#<compiled-function |INT;coerce;$Of;16|>
                    . #<vector 0000000001723ed0>))
              <15 (|lookupInDomainVector|
                   (#<compiled-function |INT;coerce;$Of;16|>
                    . #<vector 0000000001723ed0>))
              <14 (|oldCompLookup|
                   (#<compiled-function |INT;coerce;$Of;16|>
                    . #<vector 0000000001723ed0>))
              <13 (|basicLookup|
                   (#<compiled-function |INT;coerce;$Of;16|>
                    . #<vector 0000000001723ed0>))
              <12 (|compiledLookup|
                   (#<compiled-function |INT;coerce;$Of;16|>
                    . #<vector 0000000001723ed0>))

"TPD:INT:coerce(x):OutputForm"

              12> (GETDATABASE |Integer| CONSTRUCTORKIND)
              <12 (GETDATABASE |domain|)
              12> (GETL |OutputForm| LOADED)
              <12 (GETL NIL)
              12> (|loadLib| |OutputForm|)
              13> (GETL |print| |TimeTotal|)
              <13 (GETL 0.0)
              13> (GETL |gc| |TimeTotal|)
              <13 (GETL 0.0)
              13> (PUT |gc| |TimeTotal| 0.0)
              <13 (PUT 0.0)
              13> (PUT |print| |TimeTotal| 0.0)
              <13 (PUT 0.0)
```

```
13> (GETDATABASE |OutputForm| OBJECT)
<13 (GETDATABASE
     "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
13> (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
14> (|pathname|
     "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
<14 (|pathname|
     #p"/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
<13 (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
13> (|isSystemDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
<13 (|isSystemDirectory| T)
13> (|loadLibNoUpdate| |OutputForm| |OutputForm|
     "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
14> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (|makeByteWordVec2| 1 (0 0 0))
<14 (|makeByteWordVec2| #<bit-vector 0000000001723c90>)
14> (|makeByteWordVec2| 144
     (1 10 9 0 11 0 25 0 26 2 10 0 0 25 27 2 10 0 25 0 28
      2 19 0 0 0 36 2 19 0 0 0 37 2 19 9 0 0 46 1 6 0 0 56
      2 6 0 0 0 57 1 6 9 0 69 1 6 0 0 70 1 6 2 0 71 1 6 73
      0 74 1 19 9 0 75 2 76 0 0 0 77 1 76 0 0 105 1 25 0
      10 114 2 10 0 73 25 115 1 73 9 0 128 2 73 9 0 0 129
      1 131 10 130 132 1 10 0 0 133 2 0 9 0 0 1 2 0 0 0 0
      120 0 0 19 35 1 0 19 0 30 1 0 0 19 47 1 0 0 52 80 2
      0 0 0 0 48 2 0 0 0 52 78 1 0 19 0 33 2 0 0 0 0 66 2
      0 0 0 0 136 3 0 0 0 0 0 137 1 0 0 0 135 1 0 19 0 32
      2 0 0 0 0 65 1 0 0 0 109 2 0 0 0 0 124 1 0 0 52 55
      2 0 0 0 52 72 2 0 0 19 19 49 1 0 0 0 121 2 0 0 0 0
      122 1 0 0 0 45 2 0 0 0 19 42 2 0 0 0 0 93 2 0 0 0 0
      127 1 0 0 0 110 2 0 0 0 0 94 3 0 0 0 0 0 140 1 0 0
      0 138 2 0 0 0 0 139 1 0 7 0 8 2 0 0 0 73 117 1 0 0
      0 113 2 0 0 0 0 68 2 0 0 0 0 67 2 0 0 0 52 104 2 0
      0 0 0 108 1 0 0 52 53 1 0 0 52 64 1 0 0 0 63 2 0 0
      0 0 118 1 0 0 0 111 2 0 0 0 0 123 1 0 0 10 29 1 0 0
      23 24 1 0 0 21 22 1 0 0 19 20 2 0 0 0 0 97 1 0 0 0
      98 1 0 7 10 14 1 0 0 10 13 1 0 0 50 51 1 0 0 0 44 2
      0 0 0 19 41 1 0 10 0 1 2 0 0 0 0 126 3 0 0 0 0 0
      143 2 0 0 0 0 142 1 0 0 0 141 1 0 9 0 102 2 0 0 0
      52 106 3 0 0 0 0 0 107 1 0 0 19 38 0 0 19 34 1 0 19
      0 31 1 0 0 52 79 2 0 0 0 0 39 1 0 144 0 1 2 0 0 0 0
      95 0 0 0 12 2 0 0 0 52 103 2 0 0 0 73 116 1 0 0 0
      112 2 0 0 0 0 92 2 0 0 0 73 134 1 0 0 52 54 1 0 17
      0 18 1 0 0 0 43 2 0 0 0 19 40 1 0 0 0 61 1 0 0 52
      62 1 0 0 52 60 1 0 0 0 59 1 0 0 0 119 1 0 0 52 58 2
      0 0 0 0 101 2 0 0 0 0 125 2 0 0 0 0 96 2 0 0 0 0 81
      1 0 0 0 100 2 0 0 0 0 99 2 0 0 0 0 85 2 0 0 0 0 83
      2 0 0 0 0 16 2 0 9 0 0 15 2 0 0 0 0 84 2 0 0 0 0 82
      2 0 0 0 0 90 1 0 0 0 88 2 0 0 0 0 87 2 0 0 0 0 86 2
      0 0 0 0 91 2 0 0 0 0 89))
<14 (|makeByteWordVec2| #<vector 0000000001723c60>)
```

```
14> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (GETL |load| |TimeTotal|)
<14 (GETL 0.0)
14> (GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> (PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (PUT |load| |TimeTotal| 0.0)
<14 (PUT 0.0)
<13 (|loadLibNoUpdate| T)
<12 (|loadLib| T)
12> (HPUT #<hash-table 000000000105e810> |OutputForm|
     ((NIL 1 . #<vector 0000000001723c30>)))
<12 (HPUT ((NIL 1 . #<vector 0000000001723c30>)))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETL |OutputForm| |infovec|)
<12 (GETL (#<vector 0000000001723d80>
           #<vector 0000000001723d50> NIL
           (#<bit-vector 0000000001723c90>
            #<vector 0000000001723cf0>
            #<vector 0000000001723cc0>
            . #<vector 0000000001723c60>)
       |lookupComplete|))
12> (GETL |List| LOADED)
<12 (GETL NIL)
12> (|loadLib| |List|)
13> (GETL |print| |TimeTotal|)
<13 (GETL 0.0)
13> (GETL |gc| |TimeTotal|)
<13 (GETL 0.0)
13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |print| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (GETDATABASE |List| OBJECT)
<13 (GETDATABASE
     "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
13> (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
14> (|pathname|
     "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
<14 (|pathname|
     #p"/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
<13 (|pathnameDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
13> (|isSystemDirectory|
     "/home/daly/noise/mnt/ubuntu/algebra/")
<13 (|isSystemDirectory| T)
13> (|loadLibNoUpdate| |List| |List|
     "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
14> (GETDATABASE |List| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
```

```
14> (|makeByteWordVec2| 8
     (0 0 0 0 0 0 0 0 0 3 0 0 8 4 0 0 8 1 2 4 5))
<14 (|makeByteWordVec2| #<vector 0000000001723a50>)
14> (|makeByteWordVec2| 51
     (1 13 12 0 14 3 13 12 0 15 15 16 1 0 6 0 17 3 6 12
      13 0 8 18 1 0 0 0 19 1 13 12 0 20 0 21 0 22 2 13 0
      15 21 23 1 13 12 0 24 1 13 12 0 25 1 13 12 0 26 1
      0 15 0 27 2 0 15 0 8 28 2 0 12 13 0 29 3 0 12 13 0
      8 30 2 0 0 0 0 31 1 0 0 0 32 2 0 0 0 0 33 0 0 0 34
      1 0 8 0 35 2 0 8 6 0 36 2 0 0 0 0 37 2 0 6 0 38 39
      2 0 0 6 0 40 2 0 0 0 0 41 1 42 0 15 43 1 44 0 42 45
      1 6 44 0 46 2 47 0 44 0 48 1 44 0 49 50 1 0 44 0 51
      2 1 0 0 0 33 2 1 0 0 0 37 2 1 0 0 0 41 1 0 0 0 19 1
      1 0 0 32 1 0 8 0 9 0 0 0 7 2 1 8 6 0 36 1 0 6 0 17
      1 0 8 0 35 0 0 0 34 2 0 6 0 38 39 1 2 44 0 51 2 0 0
      6 0 10 2 0 0 6 0 40 2 0 0 0 0 31 2 0 0 0 0 11 3 5 12
      13 0 8 30 2 5 12 13 0 29 1 5 15 0 27 2 5 15 0 8 28))
<14 (|makeByteWordVec2| #<vector 0000000001723a20>)
14> (GETDATABASE |List| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (GETL |load| |TimeTotal|)
<14 (GETL 0.0)
14> (GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> (PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (PUT |load| |TimeTotal| 0.0)
<14 (PUT 0.0)
<13 (|loadLibNoUpdate| T)
<12 (|loadLib| T)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |SetCategory| COSIG)
<12 (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
```

```
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |SetCategory| COSIG)
<12 (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |Integer| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |Integer|)
<12 (PNAME "Integer")
12> (PNAME |Integer|)
<12 (PNAME "Integer")
12> (GETDATABASE |OrderedSet| COSIG)
<12 (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (HPUT #<hash-table 000000000105e810> |List|
     ((((|OutputForm|)) 1 . #<vector 00000000017239f0>)))
<12 (HPUT ((((|OutputForm|)) 1
          . #<vector 00000000017239f0>)))
12> (GETDATABASE |List| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETL |List| |infovec|)
<12 (GETL (#<vector 0000000001723b10>
```

```
                    #<vector 0000000001723ae0>
                    ((|shallowlyMutable| . 0)
                     (|finiteAggregate| . 0))
                     (#<vector 0000000001723a50>
                      #<vector 0000000001723ab0>
                      #<vector 0000000001723a80>
                      . #<vector 0000000001723a20>)
                |lookupIncomplete|))
           12> (HPUT #<hash-table 000000000105e810> |OutputForm|
                 ((NIL 1 . #<vector 0000000001723c30>)))
           <12 (HPUT ((NIL 1 . #<vector 0000000001723c30>)))
           12> (GETDATABASE |Integer| COSIG)
           <12 (GETDATABASE (NIL))
           12> (|basicLookup| |outputForm| ($ (|Integer|))
                #<vector 0000000001723c30> #<vector 0000000001723c30>)
           13> (|oldCompLookup| |outputForm| ($ (|Integer|))
                #<vector 0000000001723c30> #<vector 0000000001723c30>)
           14> (|lookupInDomainVector| |outputForm| ($ (|Integer|))
                #<vector 0000000001723c30> #<vector 0000000001723c30>)
           15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
           <15 (GETDATABASE |domain|)
           15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
           <15 (GETDATABASE |domain|)
           15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
           <15 (GETDATABASE |domain|)
           15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
           <15 (GETDATABASE |domain|)
           15> (GETDATABASE |Integer| COSIG)
           <15 (GETDATABASE (NIL))
           <14 (|lookupInDomainVector|
                (#<compiled-function |OUTFORM;outputForm;I$;7|>
                 . #<vector 0000000001723c30>))
           <13 (|oldCompLookup|
                (#<compiled-function |OUTFORM;outputForm;I$;7|>
                 . #<vector 0000000001723c30>))
           <12 (|basicLookup|
                (#<compiled-function |OUTFORM;outputForm;I$;7|>
                 . #<vector 0000000001723c30>))

"TPD:OUTFORM:outputForm n"

           12> (GETL |print| |TimeTotal|)
           <12 (GETL 0.0)
           12> (GETL |gc| |TimeTotal|)
           <12 (GETL 0.0)
           12> (PUT |gc| |TimeTotal| 0.0)
           <12 (PUT 0.0)
           12> (PUT |print| |TimeTotal| 0.0)
           <12 (PUT 0.0)
           12> (|member| 1 ("failed" "nil" "prime" "sqfr" "irred"))
           <12 (|member| NIL)
           12> (|member| EQUATNUM (SLASH OVER))
           <12 (|member| NIL)
           12> (GETL EQUATNUM |Led|)
```

```
                    <12 (GETL (|dummy| |dummy| 10000 0))
                    12> (|member| EQUATNUM (SLASH OVER))
                    <12 (|member| NIL)
                    12> (GETL EQUATNUM |Led|)
                    <12 (GETL (|dummy| |dummy| 10000 0))
                    12> (GETL EQUATNUM INFIXOP)
                    <12 (GETL "  ")
                    12> (GETL EQUATNUM WIDTH)
                    <12 (GETL NIL)
                    12> (GETL EQUATNUM APP)
                    <12 (GETL NIL)
                    12> (|member| EQUATNUM (SLASH OVER))
                    <12 (|member| NIL)
                    12> (GETL EQUATNUM |Led|)
                    <12 (GETL (|dummy| |dummy| 10000 0))
                    12> (|member| EQUATNUM (SLASH OVER))
                    <12 (|member| NIL)
                    12> (GETL EQUATNUM |Led|)
                    <12 (GETL (|dummy| |dummy| 10000 0))
                    12> (GETL EQUATNUM INFIXOP)
                    <12 (GETL "  ")
                    12> (GETL EQUATNUM SUPERSPAN)
                    <12 (GETL NIL)
                    12> (GETL EQUATNUM SUBSPAN)
                    <12 (GETL NIL)
(1)  1

                    12> (|putHist| % |value| ((|PositiveInteger|) . 1) ((NIL)))
                    13> (|recordNewValue| % |value| ((|PositiveInteger|) . 1))
                    14> (GETL |print| |TimeTotal|)
                    <14 (GETL 0.0)
                    14> (GETL |gc| |TimeTotal|)
                    <14 (GETL 0.0)
                    14> (PUT |gc| |TimeTotal| 0.0)
                    <14 (PUT 0.0)
                    14> (PUT |print| |TimeTotal| 0.0)
                    <14 (PUT 0.0)
                    14> (|recordNewValue0| % |value| ((|PositiveInteger|) . 1))
                    <14 (|recordNewValue0|
                         ((% (|value| (|PositiveInteger|) . 1))))
                    14> (GETL |history| |TimeTotal|)
                    <14 (GETL 0.0)
                    14> (GETL |gc| |TimeTotal|)
                    <14 (GETL 0.0)
                    14> (PUT |gc| |TimeTotal| 0.0)
                    <14 (PUT 0.0)
                    14> (PUT |history| |TimeTotal| 0.0)
                    <14 (PUT 0.0)
                    <13 (|recordNewValue| |history|)
                    13> (|search| % ((NIL)))
                    14> (|searchCurrentEnv| % (NIL))
                    <14 (|searchCurrentEnv| NIL)
                    14> (|searchTailEnv| % NIL)
                    <14 (|searchTailEnv| NIL)
```

```
<13 (|search| NIL)
<12 (|putHist| ((((% (|value| (|PositiveInteger|) . 1))))))
12> (|printTypeAndTime| 1 (|PositiveInteger|))
14> (|sayKeyedMsg| "%rjon Type: %1p %rjoff" ((|PositiveInteger|)))
15> (|sayKeyedMsgLocal| "%rjon Type: %1p %rjoff" ((|PositiveInteger|)))
16> (|segmentKeyedMsg| " %rjon Type: %1p %rjoff")
<16 (|segmentKeyedMsg| ("%rjon" "Type:" "%1p" "%rjoff"))
16> (|member| "%rjon" (|%ceon| "%ceon"))
<16 (|member| NIL)
16> (|member| "%rjon" (|%rjon| "%rjon"))
<16 (|member| ("%rjon"))
16> (|member| "Type:"
     (|%ceoff| "%ceoff" |%rjoff| "%rjoff"))
<16 (|member| NIL)
16> (|member| "%1p" (|%ceoff| "%ceoff" |%rjoff| "%rjoff"))
<16 (|member| NIL)
16> (|member| "%rjoff"
     (|%ceoff| "%ceoff" |%rjoff| "%rjoff"))
<16 (|member| ("%rjoff"))
16> (|member| "%rj" (|%ceon| "%ceon"))
<16 (|member| NIL)
16> (|member| "%rj" (|%rjon| "%rjon"))
<16 (|member| NIL)
16> (|member| "Type:" (|%ceon| "%ceon"))
<16 (|member| NIL)
16> (|member| "Type:" (|%rjon| "%rjon"))
<16 (|member| NIL)
16> (|member| "%1p" (|%ceon| "%ceon"))
<16 (|member| NIL)
16> (|member| "%1p" (|%rjon| "%rjon"))
<16 (|member| NIL)
16> (DIGITP #\r)
<16 (DIGITP NIL)
16> (DIGITP #\1)
<16 (DIGITP 1)
16> (GETDATABASE |PositiveInteger| ABBREVIATION)
<16 (GETDATABASE PI)
16> (|member| "Type:" ("%n" |%n|))
<16 (|member| NIL)
16> (|member| "Type:" ("%y" |%y|))
<16 (|member| NIL)
16> (|member| "%rj"
     (" " | | "%" % |%b| |%d| |%l| |%i| |%u| %U |%n| |%x|
      |%ce| |%rj| "%U" "%b" "%d" "%l" "%i" "%u" "%U" "%n"
      "%x" "%ce" "%rj" [ |(| "[" "("))
<16 (|member| ("%rj" [ |(| "[" "("))
16> (|member| |PositiveInteger| ("%n" |%n|))
<16 (|member| NIL)
16> (|member| |PositiveInteger| ("%y" |%y|))
<16 (|member| NIL)
16> (|member| "Type:"
     (" " | | "%" % |%b| |%d| |%l| |%i| |%u| %U |%n| |%x|
      |%ce| |%rj| "%U" "%b" "%d" "%l" "%i" "%u" "%U" "%n"
       "%x" "%ce" "%rj" [ |(| "[" "("))
```

```
                    <16 (|member| NIL)
                    16> (SIZE "Type:")
                    <16 (SIZE 5)
                    16> (|member| |PositiveInteger|
                         (" " | | "%" % |%b| |%d| |%l| |%i| |%u| %U |%n| |%x|
                          |%ce| |%rj| "%U" "%b" "%d" "%l" "%i" "%u" "%U" "%n"
                          "%x" "%ce" "%rj" |.| |,| ! |:| |;| ? ] |)| "." ","
                          "!" ":" ";" "?" "]" ")"))
                    <16 (|member| NIL)
                    16> (|member| "%rj" (|%ce| "%ce" |%rj| "%rj"))
                    <16 (|member| ("%rj"))
                    16> (|sayMSG| (("%rj" "Type:" " " |PositiveInteger|)))
                    17> (SAYBRIGHTLY1 (("%rj" "Type:" " " |PositiveInteger|))
                          #<synonym stream to *TERMINAL-IO*>)
                    18> (BRIGHTPRINT (("%rj" "Type:" " " |PositiveInteger|)))
                    19> (|member| "%rj" ("%p" "%s"))
                    <19 (|member| NIL)
                    19> (|member| "Type:" (|%l| "%l"))
                    <19 (|member| NIL)
                    19> (|member| " " (|%l| "%l"))
                    <19 (|member| NIL)
                    19> (|member| |PositiveInteger| (|%l| "%l"))
                    <19 (|member| NIL)
                    19> (|member| "Type:" ("%b" "%d" |%b| |%d|))
                    <19 (|member| NIL)
                    19> (|member| "Type:" ("%l" |%l|))
                    <19 (|member| NIL)
                    19> (|member| " " ("%b" "%d" |%b| |%d|))
                    <19 (|member| NIL)
                    19> (|member| " " ("%l" |%l|))
                    <19 (|member| NIL)
                    19> (|member| |PositiveInteger| ("%b" "%d" |%b| |%d|))
                    <19 (|member| NIL)
                    19> (|member| |PositiveInteger| ("%l" |%l|))
                    <19 (|member| NIL)
                    19> (PNAME |PositiveInteger|)
                    <19 (PNAME "PositiveInteger")
                    19> (|fillerSpaces| 56 " ")
                    <19 (|fillerSpaces|
                    "                                                        ")
                                                      Type:
                    19> (PNAME |PositiveInteger|)
                    <19 (PNAME "PositiveInteger")
PositiveInteger

                    <18 (BRIGHTPRINT NIL)
                    <17 (SAYBRIGHTLY1 NIL)
                    <16 (|sayMSG| NIL)
                    <15 (|sayKeyedMsgLocal| NIL)
                    <14 (|sayKeyedMsg| NIL)
                    <12 (|printTypeAndTime| NIL)
                    <11 (|recordAndPrint| |done|)
                    11> (recordFrame |normal|)
                    12> (|diffAlist|
```

```
     ((% (|value| (|PositiveInteger|) . 1))) NIL)
<12 (|diffAlist| ((% (|value|))))
<11 (recordFrame ((% (|value|))))
11> (GETL |print| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |print| |TimeTotal| 0.0)
<11 (PUT 0.0)
<10 (|processInteractive1| ((|PositiveInteger|) . 1))
10> (|writeHistModesAndValues|)
11> (|putHist| % |value|
    #0=((|PositiveInteger|) . 1) ((((% (|value| . #0#))))))
12> (|recordNewValue| % |value| ((|PositiveInteger|) . 1))
13> (GETL |other| |TimeTotal|)
<13 (GETL 0.0)
13> (GETL |gc| |TimeTotal|)
<13 (GETL 0.0)
13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |other| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (|recordNewValue0| % |value| ((|PositiveInteger|) . 1))
<13 (|recordNewValue0| (|value| (|PositiveInteger|) . 1))
13> (GETL |history| |TimeTotal|)
<13 (GETL 0.0)
13> (GETL |gc| |TimeTotal|)
<13 (GETL 0.0)
13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |history| |TimeTotal| 0.0)
<13 (PUT 0.0)
<12 (|recordNewValue| |history|)
12> (|search| %
     ((((% (|value| (|PositiveInteger|) . 1))))))
13> (|searchCurrentEnv| %
     (((% (|value| (|PositiveInteger|) . 1)))))
<13 (|searchCurrentEnv|
     ((|value| (|PositiveInteger|) . 1)))
<12 (|search| ((|value| (|PositiveInteger|) . 1)))
<11 (|putHist| ((((% (|value| (|PositiveInteger|) . 1))))))
<10 (|writeHistModesAndValues| NIL)
10> (|updateHist|)
11> (GETL |other| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |other| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (|updateInCoreHist|)
```

```
    <11 (|updateInCoreHist| 1)
    11> (|writeHiFi|)
    <11 (|writeHiFi|
         ((1 (% (|value| (|PositiveInteger|) . 1)))))
    11> (|disableHist|)
    <11 (|disableHist| NIL)
    11> (|updateCurrentInterpreterFrame|)
    12> (|createCurrentInterpreterFrame|)
    <12 (|createCurrentInterpreterFrame|
         (|frame0|
          ((((% (|value| . #0=((|PositiveInteger|) . 1))))))
          2 T
          #1=(NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL
              NIL NIL NIL NIL NIL NIL NIL NIL . #1#)
          20 1 NIL ((1 (% (|value| . #0#))))
          #<vector 0000000000fa5cc0>))
    12> (|updateFromCurrentInterpreterFrame|)
    <12 (|updateFromCurrentInterpreterFrame| NIL)
    <11 (|updateCurrentInterpreterFrame| NIL)
    11> (GETL |history| |TimeTotal|)
    <11 (GETL 0.0)
    11> (GETL |gc| |TimeTotal|)
    <11 (GETL 0.0)
    11> (PUT |gc| |TimeTotal| 0.0)
    <11 (PUT 0.0)
    11> (PUT |history| |TimeTotal| 0.0)
    <11 (PUT 0.0)
    <10 (|updateHist| |history|)
  <9 (|processInteractive| ((|PositiveInteger|) . 1))
<8 (|intInterpretPform| ((|PositiveInteger|) . 1))
8> (|ncPutQ|
    ((|carrier| (|ok?| . T)
      (|ptreePremacro|
        . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
      (|ptree| . #0#)
     (|lines| ((#1# . 1) . "1"))
     (|messages|)
     (|stepNumber| . 1))) |value|
     ((|PositiveInteger|) . 1))
  9> (|ncAlist|
      ((|carrier| (|ok?| . T)
       (|ptreePremacro|
        . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))))
  <9 (|ncAlist| ((|ok?| . T)
      (|ptreePremacro|
       . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
             . "1"))
      (|ptree| . #0#)
```

```
          (|lines| ((#1# . 1) . "1"))
          (|messages|)
          (|stepNumber| . 1)))
      9> (|ncAlist|
          ((|carrier| (|ok?| . T)
           (|ptreePremacro|
            . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                  . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1))))
      <9 (|ncAlist|
          ((|ok?| . T)
           (|ptreePremacro|
            . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                  . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1)))
      9> (|ncTag|
          ((|carrier| (|ok?| . T)
           (|ptreePremacro|
            . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                  . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1))))
      <9 (|ncTag| |carrier|)
    <8 (|ncPutQ| ((|PositiveInteger|) . 1))
  <7 (|phInterpret| ((|PositiveInteger|) . 1))
  7> (|ncConversationPhase,wrapup|
      ((|carrier| (|value| (|PositiveInteger|) . 1) (|ok?| . T)
       (|ptreePremacro|
        . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
              . "1"))
       (|ptree| . #0#)
       (|lines| ((#1# . 1) . "1"))
       (|messages|)
       (|stepNumber| . 1))))
  <7 (|ncConversationPhase,wrapup| NIL)
<6 (|ncConversationPhase| ((|PositiveInteger|) . 1))
6> (|ncEltQ|
    ((|carrier| (|value| (|PositiveInteger|) . 1) (|ok?| . T)
     (|ptreePremacro|
      . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
            . "1"))
     (|ptree| . #0#)
     (|lines| ((#1# . 1) . "1"))
     (|messages|)
     (|stepNumber| . 1)))
     |messages|)
```

```
      7> (|ncAlist|
          ((|carrier| (|value| (|PositiveInteger|) . 1) (|ok?| . T)
           (|ptreePremacro|
            . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                  . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1))))
      <7 (|ncAlist|
          ((|value| (|PositiveInteger|) . 1) (|ok?| . T)
           (|ptreePremacro|
            . #0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
                  . "1"))
           (|ptree| . #0#)
           (|lines| ((#1# . 1) . "1"))
           (|messages|)
           (|stepNumber| . 1)))
    <6 (|ncEltQ| NIL)
  <5 (|intloopSpadProcess,interp| NIL)
<4 (|intloopSpadProcess| 2)

4> (|StreamNull|
    (|nonnullstream| #0=|incAppend1| NIL
     (|nonnullstream| #2=|next1| |ncloopParse|
      (|nonnullstream| #0# NIL
       (|nonnullstream| #2# |lineoftoks| (|nullstream|))))))
  5> (|incAppend1| NIL
      (|nonnullstream| #0=|next1| |ncloopParse|
       (|nonnullstream| |incAppend1| NIL
        (|nonnullstream| #0# |lineoftoks| (|nullstream|)))))
    6> (|StreamNull| NIL)
    <6 (|StreamNull| T)
    6> (|StreamNull|
        (|nonnullstream| #0=|next1| |ncloopParse|
         (|nonnullstream| |incAppend1| NIL
          (|nonnullstream| #0# |lineoftoks| (|nullstream|)))))
      7> (|next1| |ncloopParse|
          (|nonnullstream| |incAppend1| NIL
           (|nonnullstream| |next1| |lineoftoks| (|nullstream|))))
        8> (|StreamNull|
            (|nonnullstream| |incAppend1| NIL
             (|nonnullstream| |next1| |lineoftoks| (|nullstream|))))
          9> (|incAppend1| NIL
              (|nonnullstream| |next1| |lineoftoks| (|nullstream|)))
            10> (|StreamNull| NIL)
            <10 (|StreamNull| T)
            10> (|StreamNull|
                 (|nonnullstream| |next1| |lineoftoks| (|nullstream|)))
            11> (|next1| |lineoftoks| (|nullstream|))
            12> (|StreamNull| (|nullstream|))
            <12 (|StreamNull| T)
            <11 (|next1| (|nullstream|))
            <10 (|StreamNull| T)
```

```
                <9 (|incAppend1| (|nullstream|))
              <8 (|StreamNull| T)
            <7 (|next1| (|nullstream|))
          <6 (|StreamNull| T)
        <5 (|incAppend1| (|nullstream|))
      <4 (|StreamNull| T)
    <3 (|intloopProcess| 2)
```

Chapter 11

Axiom Details

11.1 Variables Used

11.2 Data Structures

11.3 Functions

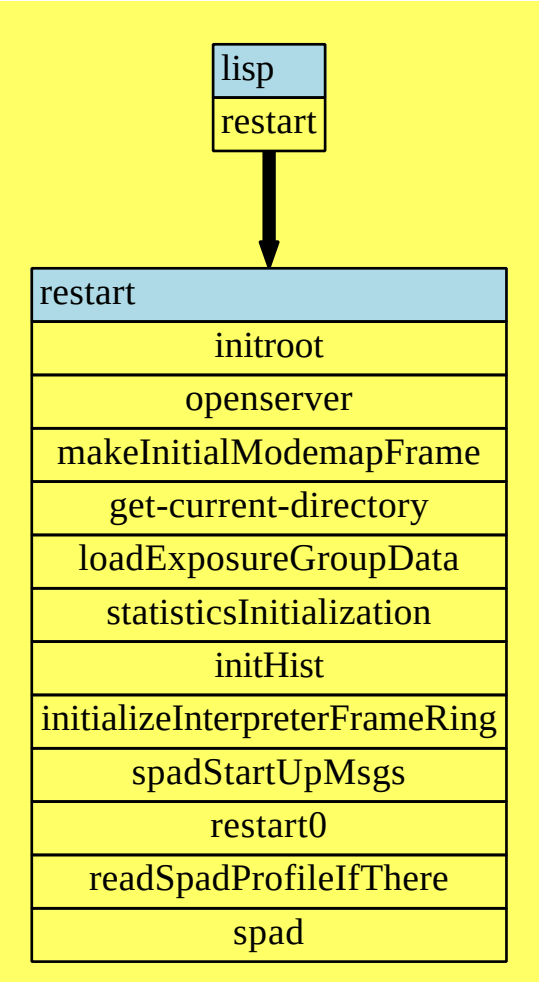
11.3.1 Set the restart hook

When a lisp image containing code is reloaded there is a hook to allow a function to be called. In our case it is the restart function which is the entry to the Axiom interpreter.
set-restart-hook : **Void** → **'restart**
— **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
 )
```

———-

11.3.2 restart function (The restart function)



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

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

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

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

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

The "initroot" (12.3.10 p 295) 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" (33.0.23 p 1049) 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 (3.1 page 15).

The "printLoadMsgs" (26.44.4 p 897) 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" (12.3.24 p 301) variable is set to the current directory. This is used by the `)cd` function and some of the compile routines.

The "statisticsInitialization" (37.0.51 p 1093) function initializes variables used to collect statistics. Currently, only the garbage collector information is initialized.

We test ***ThisIsARunningSystem***(p853). If this variable is true then we are restarting from a previously running system and we do not want to reset all of the user variables.

[init-memory-config p294]
[initroot p295]
[openserver p1049]
[makeInitialModemapFrame p296]
[get-current-directory p296]
[statisticsInitialization p1093]
[initHist p790]
[initializeInterpreterFrameRing p23]
[spadStartUpMsgs p279]
[restart0 p278]
[readSpadProfileIfThere p1021]
[spad p280]
[$openServerIfTrue p177]
[$SpadServerName p179]
[$SpadServer p178]
[$IOindex p34]
[$InteractiveFrame p34]
[$printLoadMsgs p897]
[$current-directory p301]
[$displayStartMsgs p908]
[$currentLine p**??**]

— **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
    :array 800 :string 1024 :cfun 200 :cpages 6000 :rpages 2000 :hole 4000)
#+: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 |$IOindex| 1)
  (setq |$printLoadMsgs| nil)
  (setq $current-directory (get-current-directory))
  (setq *default-pathname-defaults* (pathname $current-directory))
  (|statisticsInitialization|)
  (unless *ThisIsARunningSystem*
   (setq |$InteractiveFrame| (|makeInitialModemapFrame|))
   (|initHist|)
   (|initializeInterpreterFrameRing|))
  (setq *ThisIsARunningSystem* t)
  (when |$displayStartMsgs| (|spadStartUpMsgs|))
  (setq |$currentLine| nil)
  (restart0)
  (|readSpadProfileIfThere|)
  (|spad|))
```

———-

11.3.3 defvar localVars

— **initvars** —

```
(defvar |$localVars| ())        ;checked by isType
```

———-

11.3.4 defun Non-interactive restarts

[interpopen p1064]
[operationopen p1067]
[categoryopen p1066]
[browseopen p1065]

— **defun restart0** —

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

———-

11.3.5 defun The startup banner messages

[fillerSpaces p279]
[specialChar p1043]
[sayKeyedMsg p39]
[sayMSG p40]
[$msgAlist p38]
[$opSysName p??]
[$linelength p936]
[*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| $linelength (|specialChar| '|hbar|)))
   (|sayKeyedMsg|
    (format nil "%ceon AXIOM Computer Algebra System %l Version: %1 %l ~
                 Timestamp: %2 %ceoff")
    (list *build-version* *yearweek*))
   (|sayMSG| bar)
   (say "   Issue )copyright to view copyright notices.")
   (say "   Issue )summary for a summary of useful system commands.")
   (say "   Issue )quit to leave AXIOM and return to shell.")
   (say "   Visit http://axiom-developer.org for more information")
   (|sayMSG| bar)
   (setq |$msgAlist| nil)
   (|sayMSG| '| |))))
```

————-

11.3.6 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) " ")))))
```

————-

11.3.7 defvar $PrintCompilerMessageIfTrue

The `$PrintCompilerMessageIfTrue` variable is set to NIL in spad.

— **initvars** —

```
(defvar |$PrintCompilerMessageIfTrue| nil)
```

————-

11.3.8 Starts the interpreter but do not read in profiles

[setOutputAlgebra p921]
[runspad p280]
[$PrintCompilerMessageIfTrue p280]

— **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|))
```

————-

11.3.9 defvar $quitTag

— **initvars** —

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

————-

11.3.10 defun runspad

[quitTag p280]
[coerceFailure p??]
[top-level p??]
[seq p??]
[exit p??]
[resetStackLimits p281]
[ncTopLevel p285]
[$quitTag p280]

— **defun runspad** —

```
(defun |runspad| ()
```

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

———-

11.3.11 defun Reset the stack limits

[reset-stack-limits p**??**]

— **defun resetStackLimits 0** —

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

———-

Chapter 12

Handling Terminal Input

12.1 Streams

12.1.1 defvar curinstream

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

— **initvars** —

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

———-

12.1.2 defvar curoutstream

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

— **initvars** —

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

———-

12.1.3 defvar errorinstream

— **initvars** —

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

———-

12.1.4 defvar erroroutstream

— **initvars** —

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

————-

12.1.5 defvar *eof*

— **initvars** —

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

————-

12.1.6 defvar *whitespace*

— **initvars** —

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

————-

There are several different environments used in the interpreter:

`$InteractiveFrame` 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.

$e is the name used for `$InteractiveFrame` while interpreting.

`$env` is local environment used by the interpreter. Only temporary information (such as types of local variables is stored in `$env`. It is thrown away after evaluation of each expression.

12.1.7 defvar $InteractiveMode

— **initvars** —

```
(defvar |$InteractiveMode| (list (list nil)) "top level environment")
```

————-

12.1.8 defvar $env

— **initvars** —

```
(defvar |$env| nil "checked in isDomainValuedVariable")
```

————-

12.1.9 defvar $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`.

Prints out the value x which is of type m, and records the changes in environment `$e` into `$InteractiveFrame` Thus `$e` is a copy of the variable `$InitialModemapFrame`.

This variable is used in the undo mechanism.

— **initvars** —

```
(defvar |$e| nil "the environment?")
```

————-

12.1.10 defvar $boot

— **initvars** —

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

————-

12.1.11 $newspad

The `$newspad` is set to T in ncTopLevel.

12.1.12 defvar $newspad

— **initvars** —

```
(defvar $newspad nil)
```

————-

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

Top-level read-parse-eval-print loop for the interpreter. Uses the Bill Burge's parser. [ncIntLoop p286]
[$e p285]
[$spad p280]

[$newspad p285]
[$boot p734]
[$InteractiveMode p284]
[$InteractiveFrame p34]
[*eof* p284]
[in-stream p1021]

— **defun ncTopLevel** —

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

———-

12.1.14 defun ncIntLoop

[intloop p287]
[curinstream p283]
[curoutstream p283]

— **defun ncIntLoop** —

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

———-

12.1.15 defvar $intTopLevel

— **initvars** —

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

———-

12.1.16 defvar $intRestart

— **initvars** —

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

————-

12.1.17 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 p286]
[SpadInterpretStream p289]
[resetStackLimits p281]
[$intTopLevel p286]
[$intRestart p287]

— **defun intloop** —

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

————-

12.1.18 defvar $ncMsgList

— **initvars** —

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

————-

12.1.19 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 command is entered this function is called.

12.1.20 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)))
```

————-

12.1.21 defvar $newcompErrorCount

— **initvars** —

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

————-

12.1.22 defvar $nopos

— **initvars** —

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

————-

[mkprompt p301]
[intloopReadConsole p290]
[intloopInclude p320]

[$systemCommandFunction p??]
[$ncMsgList p287]
[$erMsgToss p??]
[$lastPos p??]
[$inclAssertions p??]
[$okToExecuteMachineCode p??]
[$newcompErrorCount p288]
[$libQuiet p??]
[$fn p??]
[$nopos p288]

— **defun SpadInterpretStream** —

```
(defun |SpadInterpretStream| (str source interactive?)
 (let (|$systemCommandFunction|
       |$ncMsgList| |$erMsgToss| |$lastPos| |$inclAssertions|
       |$okToExecuteMachineCode| |$newcompErrorCount|
       |$libQuiet|)
  (declare (special
            |$systemCommandFunction| |$ncMsgList| |$erMsgToss| |$lastPos|
            |$inclAssertions| |$okToExecuteMachineCode| |$newcompErrorCount|
            |$libQuiet| |$nopos|))
  (setq |$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|)
  (if interactive?
   (progn
     (princ (mkprompt))
     (|intloopReadConsole| "" str))
   (|intloopInclude| source 0))))
```

———-

12.2 The Read-Eval-Print Loop

12.2.1 defun intloopReadConsole

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

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

We read a line from standard input.

- If it is a null line then we exit Axiom.
- If it is a zero length line we prompt and recurse

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

Notice that all but two paths (a null input or a ")fi" or a ")fin") will end up as a recursive call to ourselves.

[top-level p**??**]
[serverReadLine p304]
[leaveScratchpad p836]
[mkprompt p301]
[intloopReadConsole p290]
[intloopPrefix? p295]
[intnplisp p296]
[setCurrentLine p301]
[ncloopCommand p727]
[concat p1107]
[ncloopEscaped p297]
[intloopProcessString p297]
[$dalymode p869]

intloopReadConsole : (String Integer) → Throw
— defun intloopReadConsole —

```
(defun |intloopReadConsole| (prefix stepNumber)
 (declare (special $dalymode))
 (let (newStepNo cmd pfx input)
   ; read the next line
   (setq input (|serverReadLine| *standard-input*))
   ; if we have lost *standard-input* then exit Axiom
   (when (null (stringp input)) (|leaveScratchpad|))
   ; if the input is a zero-length input, recurse
   (when (eql (length input) 0)
     (princ (mkprompt))
     (|intloopReadConsole| "" stepNumber))
   ; if $dalymode is nonnil anything starting with '(' is a lisp expression
   ; evaluate the expression in lisp and recurse
   (when (and $dalymode (|intloopPrefix?| "(" input))
     (|intnplisp| input)
     (princ (mkprompt))
     (|intloopReadConsole| "" stepNumber))
```

```
  ; if the input starts with ")fi" or ")fin" throw into lisp
  (setq pfx (|intloopPrefix?| ")fi" input))
  (when (and pfx (or (string= pfx ")fi") (string= pfx ")fin")))
    (throw '|top_level| nil))
  ; if the input starts with ')' it is a command; execute and recurse
  (when (and (equal prefix "") (setq cmd (|intloopPrefix?| ")" input)))
    (|setCurrentLine| cmd)
    (setq newStepNo (|ncloopCommand| cmd stepNumber))
    (princ (mkprompt))
    (|intloopReadConsole| "" newStepNo))
  ; if the last non-blank character on the line is an underscore
  ; we use the current accumulated input as a prefix and recurse.
  ; this has the effect of concatenating the next line (minus the underscore)
  (setq input (concat prefix input))
  (when (|ncloopEscaped| input)
    (|intloopReadConsole| (subseq input 0 (- (length input) 1)) stepNumber))
  ; if there are no special cases, process the current line and recurse
  (setq newStepNo (|intloopProcessString| input stepNumber))
  (princ (mkprompt))
  (|intloopReadConsole| "" newStepNo)))
```

———-

12.3 Helper Functions

12.3.1 Get the value of an evironment variable

[getenv p??]

— **defun getenviron 0** —

```
(defun getenviron (var)
 "Get the value of an evironment variable"
 #+allegro (sys::getenv (string var))
 #+clisp (ext:getenv (string var))
 #+(or cmu scl)
  (cdr
   (assoc (string var) ext:*environment-list* :test #'equalp :key #'string))
 #+(or kcl akcl gcl) (si::getenv (string var))
 #+lispworks (lw:environment-variable (string var))
 #+lucid (lcl:environment-variable (string var))
 #+mcl (ccl::getenv var)
 #+sbcl (sb-ext:posix-getenv var)
 )
```

———-

12.3.2 defvar $intCoerceFailure

— **initvars** —

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

————-

12.3.3 defvar $intSpadReader

— **initvars** —

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

————-

12.3.4 defun InterpExecuteSpadSystemCommand

[intCoerceFailure p292]
[intSpadReader p292]
[ExecuteInterpSystemCommand p292]
[$intSpadReader p292]
[$intCoerceFailure p292]

— **defun InterpExecuteSpadSystemCommand** —

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

————-

12.3.5 defun ExecuteInterpSystemCommand

[intProcessSynonyms p293]
[substring p293]
[doSystemCommand p699]
[$currentLine p??]

— **defun ExecuteInterpSystemCommand** —

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

```
  (unless (equal string "") (|doSystemCommand| string))))
```

———-

12.3.6 defun substring

— **defun substring 0** —

```
(defun substring (cvec start length)
  (if length
   (subseq (string cvec) start (+ start length))
   (subseq (string cvec) start)))
```

———-

12.3.7 defun Handle Synonyms

[processSynonyms p293]
[line p1027]

— **defun intProcessSynonyms** —

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

———-

12.3.8 defun Synonym File Reader

[strpos p1106]
[substring p293]
[string2id-n p??]
[lassoc p??]
[concat p1107]
[size p1106]
[concat p1107]
[rplacstr p??]
[processSynonyms p293]
[$CommandSynonymAlist p727]
[line p1027]

— **defun processSynonyms** —

```
(defun |processSynonyms| ()
 (let (fill p aline synstr syn to opt fun cl chr)
 (declare (special |$CommandSynonymAlist| line))
```

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

———-

12.3.9 defun init-memory-config

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

[allocate p**??**]
[allocate-contiguous-pages p**??**]
[allocate-relocatable-pages p**??**]
[set-hole-size p**??**]

— defun init-memory-config 0 —

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

```
                          (rpages 1000)
                          (hole 2000) )
  ;; initialize AKCL memory allocation parameters
  #+:AKCL
  (progn
    (system:allocate 'cons cons)
    (system:allocate 'fixnum fixnum)
    (system:allocate 'symbol symbol)
    (system:allocate 'package package)
    (system:allocate 'array array)
    (system:allocate 'string string)
    (system:allocate 'cfun cfun)
    (system:allocate-contiguous-pages cpages)
    (system:allocate-relocatable-pages rpages)
    (system:set-hole-size hole))
  #-:AKCL
  nil)
```

———-

12.3.10 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 p299]
[getenviron p291]
[$spadroot p178]

— **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)"))))
```

———-

12.3.11 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.
intloopPrefix? : **String** → **Union(String,NIL)**

— **defun intloopPrefix? 0** —

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

———-

12.3.12 defun Interpret a line of lisp code

This is used to hande `)lisp` top level commands [nplisp p722]
[$currentLine p??]

— **defun intnplisp** —

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

———-

12.3.13 Get the current directory

— **defun get-current-directory 0** —

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

———-

12.3.14 Prepend the absolute path to a filename

Prefix a filename with the **AXIOM** shell variable.

[$spadroot p178]

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

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

———-

12.3.15 Make the initial modemap frame

[copy p??]
[$InitialModemapFrame p176]

— **defun makeInitialModemapFrame 0** —

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

———-

12.3.16 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)))))
```

———-

12.3.17 defun intloopProcessString

[setCurrentLine p301]
[intloopProcess p321]
[next p298]
[incString p298]

intloopProcessString : **(String,StepNo)** → **StepNo**

— **defun intloopProcessString** —

```
(defun |intloopProcessString| (currentline stepno)
 (|setCurrentLine| currentline)
 (|intloopProcess| stepno t
  (|next| #'|ncloopParse|
   (|next| #'|lineoftoks| (|incString| currentline)))))
```

———-

12.3.18 defun ncloopParse

[ncloopDQlines p327]
[npParse p389]
[dqToList p566]

— **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))))
```

———-

12.3.19 defun next

[Delay p356]
[next1 p298]

next : (Function,Delay) → Delay
— defun next —

```
(defun |next| (function delay)
 (|Delay| #'|next1| (list function delay)))
```

———-

12.3.20 defun next1

[StreamNull p555]
[incAppend p341]
[next p298]

next1 : Delay → ParsePair
— defun next1 —

```
(defun |next1| (&rest delayArg)
 (let (h delay function)
  (setq function (car delayArg))
  (setq delay (cadr delayArg))
  (cond
   ((|StreamNull| delay) |StreamNil|)
   (t
    (setq h (apply function (list delay)))
    (|incAppend| (car h) (|next| function (cdr h)))))))
```

———-

12.3.21 defun incString

The **incString** function gets a string, usually from Axiom's input, and constructs a set of nested function calls to process the input line. [incRenumber p329]
[incLude p332]
[Top p333]

incString : **String** → **Function**
— **defun incString** —

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

———-

12.3.22 Call the garbage collector

Call the garbage collector on various platforms.
— **defun reclaim 0** —

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

———-

12.3.23 defun reroot

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

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

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

For the example call:

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

the variables are set as:

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

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

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

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

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

|$msgDatabaseName| = nil

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

[make-absolute-filename p296]
[$spadroot p178]
[$directory-list p176]
[$relative-directory-list p177]
[$library-directory-list p176]
[$relative-library-directory-list p178]
[$current-directory p301]

— **defun reroot** —

```
(defun reroot (dir)
  (declare (special $spadroot $directory-list $relative-directory-list
     $library-directory-list $relative-library-directory-list
     $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 $current-directory $spadroot))
```

————-

12.3.24 defvar $current-directory

— **initvars** —

```
(defvar |$currentLine| "" "A list of the input line history")
```

————-

12.3.25 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

[$currentLine p**??**]

setCurrentLine : **String** → **List(String)**

— **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|)
```

————-

12.3.26 Show the Axiom prompt

[concat p1107]
[substring p293]
[currenttime p**??**]
[$inputPromptType p906]
[$IOindex p34]
[$interpreterFrameName p34]

mkprompt : **Void** → **String**

— **defun mkprompt** —

```
(defun mkprompt ()
 "Show the Axiom prompt"
```

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

———-

12.3.27 defvar $frameAlist

— **initvars** —

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

———-

12.3.28 defvar $frameNumber

— **initvars** —

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

———-

12.3.29 defvar $currentFrameNum

— **initvars** —

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

———-

12.3.30 defvar $EndServerSession

— **initvars** —

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

———-

12.3.31 defvar $NeedToSignalSessionManager

— **initvars** —

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

————-

12.3.32 defvar $sockBufferLength

— **initvars** —

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

————-

12.3.33 READ-LINE in an Axiom server system

[coerceFailure p??]
[top-level p??]
[spad-reader p??]
[read-line p??]
[addNewInterpreterFrame p29]
[sockSendInt p??]
[sockSendString p??]
[mkprompt p301]
[sockGetInt p??]
[lassoc p??]
[changeToNamedInterpreterFrame p28]
[sockGetString p??]
[unescapeStringsInForm p319]
[protectedEVAL p305]
[executeQuietCommand p306]
[parseAndInterpret p306]
[serverReadLine is-console (vol9)]
[serverSwitch p??]
[$KillLispSystem p??]
[$NonSmanSession p??]
[$SpadCommand p??]
[$QuietSpadCommand p??]
[$MenuServer p??]
[$sockBufferLength p303]
[$LispCommand p??]
[$EndServerSession p302]
[$EndSession p??]
[$SwitchFrames p??]
[$CreateFrameAnswer p??]

[$currentFrameNum p302]
[$frameNumber p302]
[$frameAlist p302]
[$CreateFrame p??]
[$CallInterp p??]
[$EndOfOutput p??]
[$SessionManager p??]
[$NeedToSignalSessionManager p303]
[$EndServerSession p302]
[$SpadServer p178]
[*eof* p284]
[in-stream p1021]

serverReadLine : Stream → String
— defun serverReadLine —

```
(defun |serverReadLine| (stream)
 "used in place of READ-LINE in a Axiom server system."
 (let (in-stream *eof* l framename currentframe form stringbuf line action)
 (declare (special in-stream *eof* |$SpadServer| |$EndServerSession|
     |$NeedToSignalSessionManager| |$SessionManager| |$EndOfOutput|
     |$CallInterp| |$CreateFrame| |$frameAlist| |$frameNumber|
     |$currentFrameNum| |$CreateFrameAnswer| |$SwitchFrames| |$EndSession|
     |$EndServerSession| |$LispCommand| |$sockBufferLength| |$MenuServer|
     |$QuietSpadCommand| |$SpadCommand| |$NonSmanSession| |$KillLispSystem|))
  (force-output)
  (if (or (null |$SpadServer|) (null (is-console stream)))
   (|read-line| stream)
   (progn
    (setq in-stream stream)
    (setq *eof* nil)
    (setq line
     (do ()
         ((null (and (null |$EndServerSession|) (null *eof*))) nil)
         (when |$NeedToSignalSessionManager|
          (|sockSendInt| |$SessionManager| |$EndOfOutput|))
      (setq |$NeedToSignalSessionManager| nil)
      ; see bookvol8 for the constants that serverSwitch returns
      (setq action (|serverSwitch|))
      (cond
       ((= action |$CallInterp|)
        (setq l (|read-line| stream))
        (setq |$NeedToSignalSessionManager| t)
        (return l))
       ((= action |$CreateFrame|)
        (setq framename (gentemp "frame"))
        (|addNewInterpreterFrame| framename)
        (setq |$frameAlist|
         (cons (cons |$frameNumber| framename) |$frameAlist|))
        (setq |$currentFrameNum| |$frameNumber|)
        (|sockSendInt| |$SessionManager| |$CreateFrameAnswer|)
        (|sockSendInt| |$SessionManager| |$frameNumber|)
        (setq |$frameNumber| (1+ |$frameNumber|))
```

```
        (|sockSendString| |$SessionManager| (mkprompt)))
       ((= action |$SwitchFrames|)
        (setq |$currentFrameNum| (|sockGetInt| |$SessionManager|))
        (setq currentframe (lassoc |$currentFrameNum| |$frameAlist|))
        (|changeToNamedInterpreterFrame| currentframe))
       ((= action |$EndSession|)
        (setq |$EndServerSession| t))
       ((= action |$LispCommand|)
        (setq |$NeedToSignalSessionManager| t)
        (setq stringbuf (make-string |$sockBufferLength|))
        (|sockGetString| |$MenuServer| stringbuf |$sockBufferLength|)
        (setq form (|unescapeStringsInForm| (read-from-string stringbuf)))
        (|protectedEVAL| form))
       ((= action |$QuietSpadCommand|)
        (setq |$NeedToSignalSessionManager| t)
        (|executeQuietCommand|))
       ((= action |$SpadCommand|)
        (setq |$NeedToSignalSessionManager| t)
        (setq stringbuf (make-string 512))
        (|sockGetString| |$MenuServer| stringbuf 512)
        (catch '|coerceFailure|
         (catch '|top_level|
          (catch 'spad_reader
           (|parseAndInterpret| stringbuf))))
        (princ (mkprompt))
        (finish-output))
       ((= action |$NonSmanSession|) (setq |$SpadServer| nil))
       ((= action |$KillLispSystem|) (bye))
       (t nil))))
    (cond
     (line line)
     (t '|||||||)))
```

———-

12.3.34 defun protectedEVAL

[resetStackLimits p281]
[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)))
```

———-

12.3.35 defvar $QuietCommand

— **initvars** —

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

———-

12.3.36 defun executeQuietCommand

When `$QuiteCommand` is true Spad will not produce any output from a top level command

[spad-reader p??]
[coerceFailure p??]
[toplevel p??]
[spadreader p??]
[make-string p??]
[sockGetString p??]
[parseAndInterpret p306]
[$MenuServer p??]
[$QuietCommand p306]

— **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))))))
```

———-

12.3.37 defun parseAndInterpret

[$InteractiveMode p284]
[$boot p734]
[$spad p280]
[$e p285]
[$InteractiveFrame p34]

— **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)))
```

———-

12.3.38 defun parseFromString

[next p298]
[ncloopParse p297]
[lineoftoks p363]
[incString p298]
[StreamNull p555]
[pf2Sex p531]
[macroExpanded p463]

— **defun parseFromString** —

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

———-

12.3.39 defvar $interpOnly

— **initvars** —

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

———-

12.3.40 defvar $minivectorNames

— **initvars** —

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

———-

12.3.41 defvar $domPvar

— **initvars** —

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

———-

12.3.42 defvar $compilingMap

`$compilingMap`: true when compiling a map, used to detect where to THROW when interpret-only is invoked

— **initvars** —

```
(defvar |$compilingMap| ())
```

———-

12.3.43 defvar $instantRecord

— **initvars** —

```
(setq |$instantRecord| (make-hash-table :test #'eq))
```

———-

12.3.44 defun processInteractive

Parser Output --> Interpreter

Top-level dispatcher for the interpreter. It sets local variables and then calls processInteractive1 to do most of the work. This function receives the output from the parser.

[initializeTimedNames p**??**]
[qcar p**??**]
[processInteractive1 p311]
[reportInstantiations p903]
[clrhash p**??**]
[writeHistModesAndValues p812]
[updateHist p799]
[$op p**??**]
[$Coerce p**??**]
[$compErrorMessageStack p**??**]
[$freeVars p**??**]
[$mapList p**??**]
[$compilingMap p308]
[$compilingLoop p**??**]
[$interpOnly p307]
[$whereCacheList p**??**]

[$timeGlobalName p??]
[$StreamFrame p??]
[$declaredMode p??]
[$localVars p278]
[$analyzingMapList p??]
[$lastLineInSEQ p??]
[$instantCoerceCount p??]
[$instantCanCoerceCount p??]
[$instantMmCondCount p??]
[$fortVar p??]
[$minivector p??]
[$minivectorCode p??]
[$minivectorNames p307]
[$domPvar p308]
[$inRetract p??]
[$instantRecord p308]
[$reportInstantiations p903]
[$ProcessInteractiveValue p310]
[$defaultFortVar p??]
[$interpreterTimedNames p??]
[$interpreterTimedClasses p??]

— **defun processInteractive** —

```
(defun |processInteractive| (form posnForm)
 (let (|$op| |$Coerce| |$compErrorMessageStack| |$freeVars|
             |$mapList| |$compilingMap| |$compilingLoop|
             |$interpOnly| |$whereCacheList| |$timeGlobalName|
             |$StreamFrame| |$declaredMode| |$localVars|
             |$analyzingMapList| |$lastLineInSEQ|
             |$instantCoerceCount| |$instantCanCoerceCount|
             |$instantMmCondCount| |$fortVar| |$minivector|
             |$minivectorCode| |$minivectorNames| |$domPvar|
             |$inRetract| object)
 (declare (special |$op| |$Coerce| |$compErrorMessageStack|
                   |$freeVars| |$mapList| |$compilingMap|
                   |$compilingLoop| |$interpOnly| |$whereCacheList|
                   |$timeGlobalName| |$StreamFrame| |$declaredMode|
                   |$localVars| |$analyzingMapList| |$lastLineInSEQ|
                   |$instantCoerceCount| |$instantCanCoerceCount|
                   |$instantMmCondCount| |$fortVar| |$minivector|
                   |$minivectorCode| |$minivectorNames| |$domPvar|
                   |$inRetract| |$instantRecord| |$reportInstantiations|
                   |$ProcessInteractiveValue| |$defaultFortVar|
                   |$interpreterTimedNames| |$interpreterTimedClasses|))
  (|initializeTimedNames| |$interpreterTimedNames| |$interpreterTimedClasses|)
  (if (consp form)                        ; compute name of operator
   (setq |$op| (qcar form))
   (setq |$op| form))
  (setq |$Coerce| nil)
  (setq |$compErrorMessageStack| nil)
  (setq |$freeVars| nil)
  (setq |$mapList| nil)                   ; list of maps being type analyzed
```

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

———-

12.3.45 defvar $ProcessInteractiveValue

— **initvars** —

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

———-

12.3.46 defvar $HTCompanionWindowID

— **initvars** —

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

———-

12.3.47 defun processInteractive1

This calls the analysis and output printing routines [recordFrame p1001]
[startTimingProcess p??]
[interpretTopLevel p311]
[stopTimingProcess p??]
[recordAndPrint p315]
[objValUnwrap p462]
[objMode p462]
[$e p285]
[$ProcessInteractiveValue p310]
[$InteractiveFrame p34]

— **defun processInteractive1** —

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

———-

12.3.48 defun interpretTopLevel

[interpreter p??]
[interpret p312]
[stopTimingProcess p??]
[peekTimedName p??]
[interpretTopLevel p311]
[$timedNameStack p??]

— **defun interpretTopLevel** —

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

```
  (|interpretTopLevel| x posnForm)
  c)))
```

———-

12.3.49 defvar $genValue

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

— **initvars** —

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

———-

12.3.50 defun Type analyzes and evaluates expression x, returns object

[interpret1 p312]
[$env p284]
[$eval p??]
[$genValue p312]

— **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)))
```

———-

12.3.51 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**??**]
[mkObj p460]
[getValue p**??**]
[interpret2 p313]
[keyedSystemError p**??**]
[$genValue p312]
[$eval p**??**]

— **defun interpret1** —

```
(defun |interpret1| (x rootMode posnForm)
 (let (node modeSet newRootMode argVal val)
 (declare (special |$genValue| |$eval|))
  (setq node (|mkAtreeWithSrcPos| x posnForm))
  (when rootMode (|putTarget| node rootMode))
  (setq modeSet (|bottomUp| node))
  (if (null |$eval|)
   modeSet
   (progn
    (if (null rootMode)
     (setq newRootMode (car modeSet))
     (setq newRootMode rootMode))
    (setq argVal (|getArgValue| node newRootMode))
    (cond
     ((and argVal (null |$genValue|))
       (mkObj argVal newRootMode))
     ((and argVal (setq val (|getValue| node)))
       (|interpret2| val newRootMode posnForm))
     (t
       (|keyedSystemError|
         "Interpreter code generation failed for expression %1s"
         (list x))))))))
```

———-

12.3.52 defvar $ThrowAwayMode

— **initvars** —

```
(defvar |$ThrowAwayMode| '|$ThrowAwayMode| "interp constant")
```

———-

12.3.53 defun interpret2

This is the late interpretCoerce. I removed the call to coerceInteractive, so it only does the JENKS cases.

[objVal p462]
[objMode p462]
[member p1108]

[mkObj p460]
[systemErrorHere p??]
[coerceInteractive p658]
[throwKeyedMsgCannotCoerceWithValue p??]
[$EmptyMode p629]
[$ThrowAwayMode p313]

— **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)))
          (mkObj x m1))
        ((equal m1 |$EmptyMode|)
          (mkObj x m))
        (t
          (|systemErrorHere| "interpret2"))))
     (m1
      (if (setq ans (|coerceInteractive| object m1))
       ans
       (|throwKeyedMsgCannotCoerceWithValue| x m m1)))
     (t object))))))
```

———-

12.3.54 defvar $runTestFlag

This is referenced by maPrin to stash output by recordAndPrint to not print type/time

— **initvars** —

```
(defvar |$runTestFlag| nil)
```

———-

12.3.55 defvar $mkTestFlag

This referenced by READLN to stash input by maPrin to stash output by recordAndPrint to write i/o onto `$testStream`

— **initvars** —

```
(defvar |$mkTestFlag| nil)
```

———-

12.3.56 defun Result Output Printing

Prints out the value x which is of type m, and records the changes in environment `$e` into `$InteractiveFrame` `$printAnyIfTrue` is documented in setvart.boot. It is controlled with the `)se me any` command.

[output p**??**]
[putHist p800]
[mkObjWrap p461]
[printTypeAndTime p317]
[printStorage p316]
[printStatisticsSummary p316]
[mkCompanionPage p**??**]
[recordAndPrintTest p**??**]
[$outputMode p**??**]
[$mkTestOutputType p**??**]
[$runTestFlag p314]
[$e p285]
[$mkTestFlag p314]
[$HTCompanionWindowID p310]
[$QuietCommand p306]
[$printStatisticsSummaryIfTrue p909]
[$printTypeIfTrue p911]
[$printStorageIfTrue p**??**]
[$printTimeIfTrue p911]
[$Void p634]
[$algebraOutputStream p920]
[$collectOutput p**??**]
[$EmptyMode p629]
[$printVoidIfTrue p912]
[$outputMode p**??**]
[$printAnyIfTrue p897]

— **defun recordAndPrint** —

```
(defun |recordAndPrint| (x md)
 (let (|$outputMode| xp mdp mode)
 (declare (special |$outputMode| |$mkTestOutputType| |$runTestFlag| |$e|
                   |$mkTestFlag| |$HTCompanionWindowID| |$QuietCommand|
                   |$printStatisticsSummaryIfTrue| |$printTypeIfTrue|
                   |$printStorageIfTrue| |$printTimeIfTrue| |$Void|
                   |$algebraOutputStream| |$collectOutput| |$EmptyMode|
                   |$printVoidIfTrue| |$outputMode| |$printAnyIfTrue|))
  (cond
   ((and (equal md '(|Any|)) |$printAnyIfTrue|)
    (setq mdp (car x))
    (setq xp (cdr x)))
```

```
   (t
    (setq mdp md)
    (setq xp x)))
  (setq |$outputMode| md)
  (if (equal md |$EmptyMode|)
   (setq mode (|quadSch|))
   (setq mode md))
  (when (or (not (equal md |$Void|)) |$printVoidIfTrue|)
    (unless |$collectOutput| (terpri |$algebraOutputStream|))
    (unless |$QuietCommand| (|output| xp mdp)))
  (|putHist| '% '|value| (mkObjWrap x md) |$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|))))
```

———-

12.3.57 defun printStatisticsSummary

[sayKeyedMsg p39]
[statisticsSummary p**??**]
[$collectOutput p**??**]

— **defun printStatisticsSummary** —

```
(defun |printStatisticsSummary| ()
 (declare (special |$collectOutput|))
 (unless |$collectOutput|
  (|sayKeyedMsg| "%rjon Summary: %1 %rjoff" (list (|statisticsSummary|)))))
```

———-

12.3.58 defun printStorage

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

— **defun printStorage** —

```
(defun |printStorage| ()
 (declare (special |$interpreterTimedClasses| |$collectOutput|
```

```
                  |$interpreterTimedNames|))
  (unless |$collectOutput|
   (|sayKeyedMsg| "%rjon Storage: %1 %rjoff"
    (list
     (|makeLongSpaceString|
       |$interpreterTimedNames|
       |$interpreterTimedClasses|)))))
```

———-

12.3.59 defun printTypeAndTime

[retract p1137]
[qcar p??]
[retract p1137]
[mkObjWrap p461]
[objMode p462]
[sameUnionBranch p318]
[makeLongTimeString p??]
[msgText p319]
[sayKeyedMsg p39]
[justifyMyType p319]
[$collectOutput p??]
[$printTypeIfTrue p911]
[$printTimeIfTrue p911]
[$outputLines p??]
[$interpreterTimedNames p??]
[$interpreterTimedClasses p??]

— **defun printTypeAndTime** —

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

```
   ((and |$printTimeIfTrue| |$printTypeIfTrue|)
    (if |$collectOutput|
     (push (|msgText| "%rjon Type: %1p %rjoff" (list m)) |$outputLines|)
     (|sayKeyedMsg| "%rjon Type: %1p %l Time: %2 %rjoff"
       (list m timeString ))))
   (|$printTimeIfTrue|
    (unless |$collectOutput|
      (|sayKeyedMsg| "%rjon Time: %1 %rjoff" (list timeString))))
   (|$printTypeIfTrue|
    (if |$collectOutput|
     (push (|justifyMyType|
            (|msgText| "%rjon Type: %1p %rjoff" (list m))) |$outputLines|)
     (|sayKeyedMsg| "%rjon Type: %1p %rjoff" (list m)))))))
```

———-

12.3.60 defun printAsTeX

[$texOutputStream p??]

— **defun printAsTeX 0** —

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

———-

12.3.61 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)))))
```

———-

12.3.62 defun msgText

[segmentKeyedMsg p40]
[substituteSegmentedMsg p??]
[flowSegmentedMsg p??]
[$linelength p936]
[$margin p935]

— **defun msgText** —

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

———-

12.3.63 defun Right-justify the Type output

[fillerSpaces p279]
[$linelength p936]

— **defun justifyMyType** —

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

———-

12.3.64 defun Destructively fix quotes in strings

[unescapeStringsInForm p319]
[$funnyBacks p1379]
[$funnyQuote p1379]

— **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))))
```

———-

12.3.65 Include a file into the stream

[intloopInclude0 p320]

— **defun intloopInclude** —

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

———-

12.3.66 defun intloopInclude0

[incStream p329]
[intloopProcess p321]
[next p298]
[intloopEchoParse p325]
[insertpile p557]
[lineoftoks p363]
[$lines p??]

— **defun intloopInclude0** —

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

———-

12.3.67 defun intloopProcess

An example call looks like:

```
3> (|intloopProcess| 1 T
    (|nonnullstream| #0=|next1| |ncloopParse|
      (|nonnullstream| #0# |lineoftoks|
        (|nonnullstream| |incZip1| |incRenumberLine|
         (|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
         (|nonnullstream| |incIgen1| 0)))))
```

which was constructed **intloopProcessString**(p297). This call says we are processing the first input, in this case "1". It is interactive. The third argument, the delay, contains the information to drive the rest of the process. [StreamNull p555]
[pfAbSynOp? p624]
[setCurrentLine p301]
[tokPart p625]
[intloopProcess p321]
[intloopSpadProcess p321]
[$systemCommandFunction p??]
[$systemCommandFunction p??]

intloopProcess : (StepNo,Boolean,Delay) → StepNo
— defun intloopProcess —

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

———-

12.3.68 defun intloopSpadProcess

[flung p??]
[SpadCompileItem p??]
[intCoerceFailure p292]
[intSpadReader p292]
[ncPutQ p628]
[CatchAsCan p??]
[Catch p??]
[intloopSpadProcess,interp p322]

[$currentCarrier p??]
[$ncMsgList p287]
[$intCoerceFailure p292]
[$intSpadReader p292]
[$prevCarrier p??]
[$stepNo p??]
[$NeedToSignalSessionManager p303]
[flung p??]

— **defun intloopSpadProcess** —

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

———-

12.3.69 defun intloopSpadProcess,interp

[ncConversationPhase p324]
[ncEltQ p628]
[ncError p325]

— **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|)))
```

———-

12.3.70 defun phParse

TPDHERE: The pform function has a leading percent sign

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

[ncPutQ p628]

— **defun phParse** —

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

———-

12.3.71 defun phIntReportMsgs

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

[ncEltQ p628]
[ncPutQ p628]
[processMsgList p587]
[$erMsgToss p??]

— **defun phIntReportMsgs** —

```
(defun |phIntReportMsgs| (carrier interactive?)
 (declare (ignore interactive?))
 (let (nerr msgs lines)
 (declare (special |$erMsgToss|))
  (cond
   (|$erMsgToss| 'ok)
   (t
     (setq lines (|ncEltQ| carrier '|lines|))
     (setq msgs (|ncEltQ| carrier '|messages|))
     (setq nerr (length msgs))
     (|ncPutQ| carrier '|ok?| (eql nerr 0))
     (cond
      ((eql nerr 0) 'ok)
      (t
       (|processMsgList| msgs lines)
       (|sayKeyedMsg| "%1 error(s) parsing " (list nerr))
       'ok))))))
```

———-

12.3.72 defun phInterpret

[ncEltQ p628]
[intInterpretPform p324]
[ncPutQ p628]

— **defun phInterpret** —

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

———-

12.3.73 defun intInterpretPform

[processInteractive p308]
[zeroOneTran p324]
[pf2Sex p531]

— **defun intInterpretPform** —

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

———-

12.3.74 defun zeroOneTran

[nsubst p??]

— **defun zeroOneTran 0** —

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

———-

12.3.75 defun ncConversationPhase

[ncConversationPhase,wrapup p325]
[$ncMsgList p287]

— **defun ncConversationPhase** —

```
(defun |ncConversationPhase| (fn args)
 (let (|$ncMsgList| carrier)
 (declare (special |$ncMsgList|))
  (setq carrier (car args))
```

```
  (setq |$ncMsgList| nil)
  (unwind-protect
   (apply fn args)
   (|ncConversationPhase,wrapup| carrier))))
```

———-

12.3.76 defun ncConversationPhase,wrapup

[$ncMsgList p287]

— **defun ncConversationPhase,wrapup** —

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

———-

12.3.77 defun ncError

[SpadCompileItem p??]

— **defun ncError 0** —

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

———-

12.3.78 defun intloopEchoParse

[ncloopDQlines p327]
[setCurrentLine p301]
[mkLineList p326]
[ncloopPrintLines p326]
[npParse p389]
[dqToList p566]
[$EchoLines p??]
[$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))))
```

———-

12.3.79 defun ncloopPrintLines

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

— **defun ncloopPrintLines 0** —

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

———-

12.3.80 defun mkLineList

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

— **defun mkLineList** —

```
(defun |mkLineList| (lines)
 (let (l)
  (setq l
   ((lambda (Var2 Var1 line)
     (loop
```

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

———-

12.3.81 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))
```

———-

12.3.82 defun ncloopDQlines

[StreamNull p555]
[poGlobalLinePosn p328]
[tokPosn p625]
[streamChop p328]

— **defun ncloopDQlines** —

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

———-

12.3.83 defun poGlobalLinePosn

[lnGlobalNum p568]
[poGetLineObject p581]
[ncBug p587]

— **defun poGlobalLinePosn** —

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

———-

12.3.84 defun streamChop

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

[StreamNull p555]
[streamChop p328]
[ncloopPrefix? p728]

— **defun streamChop** —

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

———-

12.3.85 defun ncloopInclude0

[incStream p329]
[ncloopProcess p??]
[next p298]
[ncloopEchoParse p??]
[insertpile p557]
[lineoftoks p363]
[$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|))))))
```

———-

12.3.86 defun incStream

[incRenumber p329]
[incLude p332]
[incRgen p356]
[Top p333]

— **defun incStream** —

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

———-

12.3.87 defun incRenumber

[incZip p330]
[incIgen p330]

incRenumber : **Delay** → **Delay**

— **defun incRenumber** —

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

———-

12.3.88 defun incZip

Axiom "zips" a function together with two delays into a delay.

[Delay p356]
[incZip1 p330]

incZip : **(Function,Delay,Delay) → Delay**
— **defun incZip** —

```
(defun |incZip| (function delay1 delay2)
  (|Delay| #'|incZip1| (list function delay1 delay2)))
```

———-

12.3.89 defun incZip1

[StreamNull p555]
[incZip p330]

incZip1 : **Delay → ParsePair**
— **defun incZip1** —

```
(defun |incZip1| (&rest delayArg)
 (let (function delay1 delay2)
  (setq function (car delayArg))
  (setq delay1 (cadr delayArg))
  (setq delay2 (caddr delayArg))
  (cond
   ((|StreamNull| delay1) |StreamNil|)
   ((|StreamNull| delay2) |StreamNil|)
   (t
     (cons
      (funcall function (car delay1) (car delay2))
      (|incZip| function (cdr delay1) (cdr delay2)))))))
```

———-

12.3.90 defun incIgen

[Delay p356]
[incIgen1 p331]

incIgen : **Integer → Delay**
— **defun incIgen** —

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

———-

12.3.91 defun incIgen1

[incIgen p330]

— **defun incIgen1** —

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

————-

12.3.92 defun incRenumberLine

[incRenumberItem p331]
[incHandleMessage p331]

— **defun incRenumberLine** —

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

————-

12.3.93 defun incRenumberItem

[lnSetGlobalNum p568]

— **defun incRenumberItem** —

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

————-

12.3.94 defun incHandleMessage

[ncSoftError p573]
[ncBug p587]

— **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)))))
```

———-

12.3.95 defun incLude

This function takes

1. **eb** – in Integer
2. **ss** – a list of strings
3. **ln** – an Integer
4. **ufos** – a list of strings
5. **states** – a list of integers

and constructs a call to **Delay**(p356).

[Delay p356]
[incLude1 p336]

incLude : **(Int,List(String),Int,List(String),List(Int))** → **Delay**

— **defun incLude** —

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

———-

12.3.96 defmacro Rest

— **defmacro Rest** —

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

———-

12.3.97 defvar Top

— **initvars** —

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

———-

12.3.98 defvar IfSkipToEnd

— **initvars** —

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

———-

12.3.99 defvar IfKeepPart

— **initvars** —

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

———-

12.3.100 defvar IfSkipPart

— **initvars** —

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

———-

12.3.101 defvar ElseifSkipToEnd

— **initvars** —

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

———-

12.3.102 defvar ElseifKeepPart

— **initvars** —

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

———-

12.3.103 defvar ElseifSkipPart

— **initvars** —

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

———-

12.3.104 defvar ElseSkipToEnd

— **initvars** —

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

———-

12.3.105 defvar ElseKeepPart

— **initvars** —

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

———-

12.3.106 defun Top?

[quotient p**??**]

— **defun Top? 0** —

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

———-

12.3.107 defun If?

[quotient p**??**]

— **defun If?** —

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

———-

12.3.108 defun Elseif?

[quotient p??]

— **defun Elseif?** —

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

———-

12.3.109 defun Else?

[quotient p??]

— **defun Else?** —

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

———-

12.3.110 defun SkipEnd?

[remainder p??]

— **defun SkipEnd?** —

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

———-

12.3.111 defun KeepPart?

[remainder p??]

— **defun KeepPart?** —

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

```
 (eql (remainder |st| 10) 1))
```

———-

12.3.112 defun SkipPart?

[remainder p??]

— **defun SkipPart?** —

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

———-

12.3.113 defun Skipping?

[KeepPart? p335]

— **defun Skipping?** —

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

———-

12.3.114 defun incLude1

[StreamNull p555]
[Top? p334]
[xlPrematureEOF p340]
[Skipping? p336]
[xlSkip p343]
[Rest p332]
[xlOK p341]
[xlOK1 p341]
[concat p1107]
[incCommandTail p354]
[xlSay p344]
[xlNoSuchFile p344]
[xlCannotRead p345]
[incActive? p356]
[xlFileCycle p346]
[incLude p332]
[incFileInput p355]
[incAppend p341]
[inclFname p355]

[xlConActive p347]
[xlConStill p347]
[incConsoleInput p355]
[incNConsoles p356]
[xlConsole p348]
[xlSkippingFin p348]
[xlPrematureFin p349]
[assertCond p350]
[ifCond p343]
[If? p334]
[Elseif? p335]
[xlIfSyntax p350]
[SkipEnd? p335]
[KeepPart? p335]
[SkipPart? p336]
[xlIfBug p351]
[xlCmdBug p351]
[expand-tabs p??]
[incClassify p353]

— **defun incLude1** —

```
(defun |incLude1| (&rest z)
 (let (pred s1 n tail head includee fn1 info str state lno states
       ufos ln ss eb)
  (setq eb (car z))
  (setq ss (cadr . (z)))
  (setq ln (caddr . (z)))
  (setq ufos (cadddr . (z)))
  (setq states (car (cddddr . (z))))
  (setq lno (+ ln 1))
  (setq state (elt states 0))
  (cond
   ((|StreamNull| ss)
     (cond
      ((null (|Top?| state))
       (cons (|xlPrematureEOF| eb ")--premature end" lno ufos)
             |StreamNil|))
      (t |StreamNil|)))
   (t
    (progn
     (setq str (expand-tabs (car ss)))
     (setq info (|incClassify| str))
     (cond
      ((null (elt info 0))
       (cond
        ((|Skipping?| state)
         (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
        (t
         (cons (|xlOK| eb str lno (elt ufos 0)) (|Rest|)))))
      ((equal (elt info 2) "other")
       (cond
        ((|Skipping?| state)
```

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

```
      (setq head
       (cons (|xlConActive| eb str lno ufos n) head))
      (setq tail
       (cons (|xlConStill| eb str lno ufos n) tail))))
    (setq head (cons (|xlConsole| eb str lno ufos) head))
    (cons (|xlOK| eb str lno (elt ufos 0))
          (|incAppend| head tail))))))
((equal (elt info 2) "fin")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
  ((null (|Top?| state))
   (cons (|xlPrematureFin| eb str lno ufos) |StreamNil|))
  (t
   (cons (|xlOK| eb str lno (elt ufos 0)) |StreamNil|))))
((equal (elt info 2) "assert")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
  (t
   (progn
    (|assertCond| str info)
    (cons (|xlOK| eb str lno (elt ufos 0))
          (|incAppend| includee (|Rest|)))))))
((equal (elt info 2) "if")
 (progn
  (setq s1
   (cond
    ((|Skipping?| state) |IfSkipToEnd|)
    (t
     (cond
      ((|ifCond| str info) |IfKeepPart|)
      (t |IfSkipPart|)))))
  (cons (|xlOK| eb str lno (elt ufos 0))
        (|incLude| eb (cdr ss) lno ufos (cons s1 states)))))
((equal (elt info 2) "elseif")
 (cond
  ((and (null (|If?| state)) (null (|Elseif?| state)))
   (cons (|xlIfSyntax| eb str lno ufos info states)
         |StreamNil|))
  (t
   (cond
    ((or (|SkipEnd?| state)
         (|KeepPart?| state)
         (|SkipPart?| state))
     (setq s1
      (cond
       ((|SkipPart?| state)
        (setq pred (|ifCond| str info))
        (cond
         (pred |ElseifKeepPart|)
         (t |ElseifSkipPart|)))
       (t |ElseifSkipToEnd|)))
     (cons (|xlOK| eb str lno (elt ufos 0))
```

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

———-

12.3.115 defun xlPrematureEOF

[xlMsg p340]
[inclmsgPrematureEOF p342]

— **defun xlPrematureEOF** —

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

———-

12.3.116 defun xlMsg

[incLine p342]

— **defun xlMsg** —

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

```
  (list (incLine extrablanks string globalnum localnum fileobj) mess)))
```

———-

12.3.117 defun xlOK

[xlOK1 p341]

— **defun xlOK** —

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

———-

12.3.118 defun xlOK1

[incLine1 p342]

— **defun xlOK1** —

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

———-

12.3.119 defun incAppend

[Delay p356]
[incAppend1 p341]

— **defun incAppend** —

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

———-

12.3.120 defun incAppend1

[StreamNull p555]
[incAppend p341]

— **defun incAppend1** —

```
(defun |incAppend1| (&rest z)
 (let (y x)
  (setq x (car z))
```

```
  (setq y (cadr z))
  (cond
   ((|StreamNull| x)
    (cond ((|StreamNull| y) |StreamNil|) (t y)))
   (t
    (cons (car x) (|incAppend| (cdr x) y))))))
```

———-

12.3.121 defun incLine

[incLine1 p342]

— **defun incLine** —

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

———-

12.3.122 defun incLine1

[lnCreate p567]

— **defun incLine1** —

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

———-

12.3.123 defun inclmsgPrematureEOF

[theorigin p343]

— **defun inclmsgPrematureEOF 0** —

```
(defun |inclmsgPrematureEOF| (ufo)
 (list
  (format nil
   "File %1f ended where at least one )endif was still needed.
    An appropriate number of )endif lines has been assumed.")
  (list (|theorigin| ufo))))
```

———-

12.3.124 defun theorigin

— defun theorigin 0 —

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

————-

12.3.125 defun porigin

[pfname p345]

— defun porigin —

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

————-

12.3.126 defun ifCond

[MakeSymbol p??]
[incCommandTail p354]
[$inclAssertions p??]

— defun ifCond —

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

————-

12.3.127 defun xlSkip

[incLine p342]
[concat p1107]

— defun xlSkip —

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

———-

12.3.128 defun xlSay

[xlMsg p340]
[inclmsgSay p344]

— **defun xlSay** —

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

———-

12.3.129 defun inclmsgSay

[theid p344]

— **defun inclmsgSay** —

```
(defun |inclmsgSay| (str)
 (list "%1f" (list (|theid| str))))
```

———-

12.3.130 defun theid

— **defun theid 0** —

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

———-

12.3.131 defun xlNoSuchFile

[xlMsg p340]
[inclmsgNoSuchFile p345]

— **defun xlNoSuchFile** —

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

———-

12.3.132 defun inclmsgNoSuchFile

[thefname p345]

— **defun inclmsgNoSuchFile** —

```
(defun |inclmsgNoSuchFile| (fn)
 (list "The )include file %1f does not exist." (list (|thefname| fn))))
```

————-

12.3.133 defun thefname

[pfname p345]

— **defun thefname 0** —

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

————-

12.3.134 defun pfname

[PathnameString p??]

— **defun pfname** —

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

————-

12.3.135 defun xlCannotRead

[xlMsg p340]
[inclmsgCannotRead p345]

— **defun xlCannotRead** —

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

————-

12.3.136 defun inclmsgCannotRead

[thefname p345]

— **defun inclmsgCannotRead** —

```
(defun |inclmsgCannotRead| (fn)
```

```
 (list "The )include file %1f exists, but cannot be read."
   (list (|thefname| fn))))
```

———-

12.3.137 defun xlFileCycle

[xlMsg p340]
[inclmsgFileCycle p346]

— **defun xlFileCycle** —

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

———-

12.3.138 defun inclmsgFileCycle

```
;inclmsgFileCycle(ufos,fn) ==
;    flist := [porigin n for n in reverse ufos]
;    f1    := porigin fn
;    cycle := [:[:[n,'"==>"] for n in flist], f1]
;    ['SCI0004, [%id cycle, %id f1] ]
```

[porigin p343]
[theid p344]

— **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
   (format nil
    "There is a cycle in the )include files: %i %l %1f %u %l. ~
     The inner occurrence of %2f has not been included.")
   (list (|theid| cycle) (|theid| f1)))))
```

———-

12.3.139 defun xlConActive

[xlMsg p340]
[inclmsgConActive p347]

— **defun xlConActive** —

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

———-

12.3.140 defun inclmsgConActive

[theid p344]

— **defun inclmsgConActive** —

```
(defun |inclmsgConActive| (n)
 (list
  (format nil
   "%1f other )console commands are currently active. ~
    While this new )console command is reading input the others ~
    will have to wait. !
    Remember, each )console command will need a separate )fin.")
  (list (|theid| n))))
```

———-

12.3.141 defun xlConStill

[xlMsg p340]
[inclmsgConStill p348]

— **defun xlConStill** —

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

———-

12.3.142 defun inclmsgConStill

[theid p344]

— **defun inclmsgConStill** —

```
(defun |inclmsgConStill| (n)
 (list
  (format nil
   "The current )console command has finished reading. ~
    %1f are still active.  Remember, each will need a separate )fin.")
   (list (|theid| n))))
```

———-

12.3.143 defun xlConsole

[xlMsg p340]
[inclmsgConsole p348]

— **defun xlConsole** —

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

———-

12.3.144 defun inclmsgConsole

— **defun inclmsgConsole 0** —

```
(defun |inclmsgConsole| ()
 (list "Including source lines from console.  Type )fin when done." nil))
```

———-

12.3.145 defun xlSkippingFin

[xlMsg p340]
[inclmsgFinSkipped p349]

— **defun xlSkippingFin** —

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

———-

12.3.146 defun inclmsgFinSkipped

— **defun inclmsgFinSkipped 0** —

```
(defun |inclmsgFinSkipped| ()
 (list
  (format nil
   "A )fin command was skipped ~
   (along with everything else) in a false branch of an )if...)endif.")
  nil))
```

———-

12.3.147 defun xlPrematureFin

[xlMsg p340]
[inclmsgPrematureFin p349]

— **defun xlPrematureFin** —

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

———-

12.3.148 defun inclmsgPrematureFin

[theorigin p343]

— **defun inclmsgPrematureFin** —

```
(defun |inclmsgPrematureFin| (ufo)
 (list
  (format nil
   "A )fin command has been given in %1f where at least one )endif ~
    was still needed. ~
    An appropriate number of )endif lines have been assumed.")
   (list (|theorigin| ufo))))
```

———-

12.3.149 defun assertCond

[MakeSymbol p??]
[incCommandTail p354]
[$inclAssertions p??]
[*whitespace* p284]

— **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|)))))
```

———-

12.3.150 defun xlIfSyntax

[Top? p334]
[Else? p335]
[xlMsg p340]
[inclmsgIfSyntax p350]

— **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|))))
```

———-

12.3.151 defun inclmsgIfSyntax

[concat p1107]
[theid p344]
[theorigin p343]

— **defun inclmsgIfSyntax** —

```
(defun |inclmsgIfSyntax| (ufo found context)
 (setq found (concat ")" found))
```

```
 (list
  (format nil
   "Incorrect )if...)endif syntax.  A %1f was found %2f. ~
    The processing of the source from %3f has been abandoned.")
  (list (|theid| found) (|theid| context) (|theorigin| ufo))))
```

———-

12.3.152 defun xlIfBug

[xlMsg p340]
[inclmsgIfBug p351]

— **defun xlIfBug** —

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

———-

12.3.153 defun inclmsgIfBug

— **defun inclmsgIfBug 0** —

```
(defun |inclmsgIfBug| ()
  (list "Unexpected state in )if...)endif." nil))
```

———-

12.3.154 defun xlCmdBug

[xlMsg p340]
[inclmsgCmdBug p351]

— **defun xlCmdBug** —

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

———-

12.3.155 defun inclmsgCmdBug

— **defun inclmsgCmdBug 0** —

```
(defun |inclmsgCmdBug| ()
 (list "Unexpected command in source inclusion." nil))
```

———-

12.3.156 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")))
```

———-

12.3.157 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)
```

———-

12.3.158 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? p353]

[incCommands p352]

— **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)))))))
```

———-

12.3.159 defun incCommand?

incCommand? : **String** → **Boolean**

— **defun incCommand? 0** —

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

———-

12.3.160 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))))
```

———-

12.3.161 defun incCommandTail

[incDrop p355]

— **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)))
```

———-

12.3.162 defun incDrop

[substring p293]

— **defun incDrop 0** —

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

————-

12.3.163 defun inclFname

[incFileName p827]
[incCommandTail p354]

— **defun inclFname** —

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

————-

12.3.164 defun incFileInput

[incRgen p356]
[make-instream p1045]

— **defun incFileInput** —

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

————-

12.3.165 defun incConsoleInput

[incRgen p356]
[make-instream p1045]

— **defun incConsoleInput** —

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

————-

12.3.166 defun incNConsoles

[incNConsoles p356]

— **defun incNConsoles** —

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

————-

12.3.167 defun incActive?

— **defun incActive? 0** —

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

————-

12.3.168 defun incRgen

Note that incRgen1 recursively calls this function.

[Delay p356]
[incRgen1 p357]

— **defun incRgen** —

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

————-

12.3.169 defun Delay

Delay prepends a label **nonnullstream**, returning a list of the label, the given function name in **function** and **arguments**. That is, given

```
(|Delay| |incLude1|  (0 ("1") 0 ("strings") (1)))
```

construct

```
(|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))
```

Note that **nonnullstream** is NOT a function so the inputs have been changed from a function call to a simple list.

Delay : **(Function,List(Any))** → **Delay**

— **defun Delay 0** —

```
(defun |Delay| (function arguments)
 (cons '|nonnullstream| (cons function arguments)))
```

———-

12.3.170 defvar StreamNil

— **initvars** —

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

———-

— **postvars** —

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

———-

12.3.171 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 p356]
[StreamNil p357]

— **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 13

The Token Scanner

13.0.172 defvar scanKeyWords

— **postvars** —

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

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

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

———-

13.0.173 defvar infgeneric

— **postvars** —

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

```
      (list 'ge '>=)
      (list 'notequal '~=)
      (list 'by '|by|)
      (list 'arrow '->)
      (list 'larrow '<-)
      (list 'bar '|\||)
      (list 'seg '|..|))
     nil))))
```

———-

13.0.174 defun lineoftoks

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

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

[nextline p363]
[scanIgnoreLine p364]
[incPrefix? p354]
[substring p293]
[dqUnit p565]
[constoken p364]
[$floatok p??]
[$f p??]
[$sz p??]
[$linepos p??]
[$r p??]

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

— **defun lineoftoks** —

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

———-

13.0.175 defun nextline

[npNull p555]
[strposl p1107]
[$sz p??]
[$n p??]
[$linepos p??]
[$ln p??]
[$r p??]
[$f p??]

— **defun nextline** —

```
(defun |nextline| (s)
```

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

———-

13.0.176 defun scanIgnoreLine

[incPrefix? p354]

— **defun scanIgnoreLine** —

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

———-

13.0.177 defun constoken

[ncPutQ p628]

— **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))
```

———-

13.0.178 defun scanToken

[startsComment? p366]
[scanComment p366]
[startsNegComment? p367]
[scanNegComment p367]
[lfid p366]
[punctuation? p368]
[scanPunct p368]
[startsId? p1105]
[scanWord p375]
[scanSpace p378]
[scanString p379]
[scanNumber p380]
[scanEscape p383]
[scanError p383]
[dqUnit p565]
[constoken p364]
[lnExtraBlanks p567]
[$linepos p??]
[$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 (char-code (char |$ln| |$n|)))
  (setq linepos |$linepos|)
  (setq n |$n|)
  (setq ch (elt |$ln| |$n|))
  (setq b
   (cond
    ((|startsComment?|) (|scanComment|) nil)
    ((|startsNegComment?|) (|scanNegComment|) nil)
    ((= c (char-code #\?))
      (setq |$n| (+ |$n| 1))
      (|lfid| "?"))
    ((|punctuation?| c) (|scanPunct|))
    ((|startsId?| ch) (|scanWord| nil))
    ((= c (char-code #\space)) (|scanSpace|) nil)
    ((= c (char-code #\")) (|scanString|))
    ((digitp ch) (|scanNumber|))
    ((= c (char-code #\_)) (|scanEscape|))
    (t (|scanError|))))
  (cond
    ((null b) nil)
    (t
     (|dqUnit|
      (|constoken| ln linepos b (+ n (|lnExtraBlanks| linepos))))))))
```

———-

13.0.179 defun lfid

To pair badge and badgee

— **defun lfid 0** —

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

———-

13.0.180 defun Is it a ++ comment?

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

— **defun startsComment? 0** —

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

———-

13.0.181 defun scanComment

[lfcomment p367]
[substring p293]
[$ln p??]
[$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))))
```

———-

13.0.182 defun lfcomment

— defun lfcomment 0 —

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

———-

13.0.183 defun Is it a – comment?

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

— defun startsNegComment? —

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

———-

13.0.184 defun scanNegComment

[lfnegcomment p368]
[substring p293]
[$ln p??]
[$sz p??]
[$n p??]

— defun scanNegComment —

```
(defun |scanNegComment| ()
 (let (n)
```

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

———-

13.0.185 defun lfnegcomment

— **defun lfnegcomment 0** —

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

———-

13.0.186 defun punctuation?

— **defun punctuation?** —

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

———-

13.0.187 defun scanPunct

[subMatch p369]
[scanError p383]
[scanKeyTr p370]
[$n p??]
[$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)))))
```

———-

13.0.188 defun subMatch

[substringMatch p369]

— **defun subMatch** —

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

————-

13.0.189 defun substringMatch

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

[size p1106]

— **defun substringMatch** —

```
(defun |substringMatch| (l dict i)
 (let (equl ls s s1 done ll u h)
  (setq h (char-code (char l i)))
  (setq u (elt dict h))
  (setq ll (size l))
  (setq s1 "")
  ((lambda (Var4 j)
    (loop
     (cond
      ((or (> j Var4) done) (return nil))
      (t
       (setq s (elt u j))
       (setq ls (size s))
       (setq done
        (cond
```

```
        ((< ll (+ ls i)) nil)
        (t
         (setq equl t)
         ((lambda (Var5 k)
          (loop
           (cond
            ((or (> k Var5) (not equl)) (return nil))
            (t
             (setq equl (= (char-code (char s k))
                           (char-code (char l (+ k i)))))))
             (setq k (+ k 1))))
          (- ls 1) 1)
         (cond (equl (setq s1 s) t) (t nil)))))))
    (setq j (+ j 1))))
  (- (size u) 1) 0)
 s1))
```

———-

13.0.190 defun scanKeyTr

[keyword p370]
[scanPossFloat p371]
[lfkey p371]
[scanCloser? p375]
[$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))))
```

———-

13.0.191 defun keyword

[hget p1105]

— **defun keyword 0** —

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

———-

13.0.192 defun keyword?

[hget p1105]

— **defun keyword? 0** —

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

————-

13.0.193 defun scanPossFloat

[lfkey p371]
[spleI p372]
[scanExponent p375]
[$ln p??]
[$sz p??]
[$n p??]

— **defun scanPossFloat** —

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

————-

13.0.194 defun digit?

[digitp p1106]

— **defun digit?** —

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

————-

13.0.195 defun lfkey

[keyword p370]

— **defun lfkey** —

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

———-

13.0.196 defun spleI

[spleI1 p372]

— **defun spleI** —

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

———-

13.0.197 defun spleI1

[substring p293]
[scanEsc p373]
[spleI1 p372]
[concat p1107]
[$ln p??]
[$sz p??]
[$n p??]

— **defun spleI1** —

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

———-

13.0.198 defun scanEsc

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

[nextline p363]
[scanEsc p373]
[strposl p1107]
[startsNegComment? p367]
[startsComment? p366]
[$ln p??]
[$r p??]
[$sz p??]
[$n p??]

— **defun scanEsc** —

```
(defun |scanEsc| ()
 (let (n1)
 (declare (special |$ln| |$r| |$sz| |$n|))
  (cond
```

```
   ((not (< |$n| |$sz|))
    (cond
     ((|nextline| |$r|)
      ((lambda ()
       (loop
        (cond
         (|$n| (return nil))
         (t (|nextline| |$r|))))))
      (|scanEsc|)
      nil)
     (t nil)))
   (t
    (setq n1 (strposl " " |$ln| |$n| t))
    (cond
     ((null n1)
      (cond
       ((|nextline| |$r|)
        ((lambda ()
          (loop
           (cond
            (|$n| (return nil))
            (t (|nextline| |$r|))))))
        (|scanEsc|)
        nil)
       (t nil)))
     ((equal |$n| n1) t)
     ((= (char-code (char |$ln| n1)) (char-code #\_))
      (setq |$n| (+ n1 1))
      (|scanEsc|)
      nil)
     (t (setq |$n| n1)
      (cond
       ((or (|startsNegComment?|) (|startsComment?|))
        (progn
         (|nextline| |$r|)
         (|scanEsc|)
         nil))
       (t nil))))))))
```

———-

13.0.199 defvar scanCloser

— **postvars** —

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

———-

13.0.200 defun scanCloser?

[keyword p370]
[scanCloser p374]

— **defun scanCloser? 0** —

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

———-

13.0.201 defun scanWord

[scanW p377]
[lfid p366]
[keyword? p371]
[lfkey p371]
[$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)))))
```

———-

13.0.202 defun scanExponent

[lffloat p376]
[digit? p371]
[spleI p372]
[concat p1107]
[$ln p??]
[$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 (char-code (char |$ln| |$n|)))
    (cond
     ((or (= c (char-code #\E)) (= c (char-code #\e)))
      (setq |$n| (+ |$n| 1))
      (cond
       ((not (< |$n| |$sz|))
        (setq |$n| n)
        (|lffloat| a w "0"))
       ((digitp (elt |$ln| |$n|))
        (setq e (|spleI| #'digitp))
        (|lffloat| a w e))
       (t
        (setq c1 (char-code (char |$ln| |$n|)))
        (cond
         ((or (= c1 (char-code #\+)) (= c1 (char-code #\-)))
           (setq |$n| (+ |$n| 1))
           (cond
            ((not (< |$n| |$sz|))
             (setq |$n| n)
             (|lffloat| a w "0"))
            ((digitp (elt |$ln| |$n|))
             (setq e (|spleI| #'digitp))
             (|lffloat| a w
              (cond
               ((= c1 (char-code #\-))
                (concat "-" e))
               (t e))))
            (t
             (setq |$n| n)
             (|lffloat| a w "0")))))))
     (t (|lffloat| a w "0")))))))
```

———-

13.0.203 defun lffloat

[concat p1107]

— **defun lffloat 0** —

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

———-

13.0.204 defmacro idChar?

— defmacro idChar? 0 —

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

————-

13.0.205 defun scanW

[posend p378]
[substring p293]
[scanEsc p373]
[scanW p377]
[idChar? p377]
[concat p1107]
[$ln p??]
[$sz p??]
[$n p??]

— defun scanW —

```
(defun |scanW| (b)
 (let (bb a str endid l n1)
 (declare (special |$ln| |$sz| |$n|))
  (setq n1 |$n|)
  (setq |$n| (+ |$n| 1))
  (setq l |$sz|)
  (setq endid (|posend| |$ln| |$n|))
  (cond
   ((or (equal endid l)
        (not (= (char-code (char |$ln| endid)) (char-code #\_))))
    (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)))))))
```

————-

13.0.206 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)
```

———-

13.0.207 defun scanSpace

[strposl p1107]
[lfspaces p378]
[$floatok p??]
[$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))))
```

———-

13.0.208 defun lfspaces

— **defun lfspaces 0** —

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

———-

13.0.209 defun scanString

[lfstring p379]
[scanS p379]
[$floatok p??]
[$n p??]

— defun scanString —

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

———-

13.0.210 defun lfstring

— defun lfstring 0 —

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

———-

13.0.211 defun scanS

[ncSoftError p573]
[lnExtraBlanks p567]
[strpos p1106]
[substring p293]
[scanEsc p373]
[concat p1107]
[scanTransform p380]
[scanS p379]
[$ln p??]
[$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|) |$n|))
     "Quote added at end of line." nil) "")
  (t
   (setq n |$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|))
       "Quote added at end of line." 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)))))))
```

———-

13.0.212 defun scanTransform

— **defun scanTransform** —

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

———-

13.0.213 defun scanNumber

[spleI p372]
[lfinteger p382]
[spleI1 p372]
[scanExponent p375]
[scanCheckRadix p382]
[lfrinteger p382]
[concat p1107]
[$floatok p??]

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

— **defun scanNumber** —

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

————-

13.0.214 defun rdigit?

[strpos p1106]

— **defun rdigit? 0** —

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

———-

13.0.215 defun lfinteger

— **defun lfinteger 0** —

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

———-

13.0.216 defun lfrinteger

[concat p1107]

— **defun lfrinteger 0** —

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

———-

13.0.217 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))
          "The character %1 is greater than the radix."
          (list (elt w i)))))))
    (setq i (+ i 1))))
   (- ns 1) 0)))
```

———-

13.0.218 defun scanEscape

[scanEsc p373]
[scanWord p375]
[$n p??]

— **defun scanEscape** —

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

———-

13.0.219 defun scanError

[ncSoftError p573]
[lnExtraBlanks p567]
[lferror p384]
[$ln p??]
[$linepos p??]
[$n p??]

— **defun scanError** —

```
(defun |scanError| ()
 (let (n)
 (declare (special |$ln| |$linepos| |$n|))
  (setq n |$n|)
  (setq |$n| (+ |$n| 1))
  (|ncSoftError|
   (cons |$linepos| (+ (|lnExtraBlanks| |$linepos|) |$n|))
    "The character %1 is not an AXIOM character."
    (list (elt |$ln| n)))
  (|lferror| (elt |$ln| n))))
```

———-

13.0.220 defun lferror

— defun lferror 0 —

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

————-

13.0.221 defvar scanKeyTable

— postvars —

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

————-

13.0.222 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))
```

————-

13.0.223 defvar scanDict

— postvars —

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

———-

13.0.224 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 p1105]

— defun scanDictCons —

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

———-

13.0.225 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
```

— **defun scanInsert** —

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

———-

13.0.226 defvar scanPun

— postvars —

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

———-

13.0.227 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 p1105]

— defun scanPunCons —

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

———-

Chapter 14

Input Stream Parser

14.0.228 defun Input Stream Parser

[trappoint p??]
[npFirstTok p391]
[npItem p390]
[ncSoftError p573]
[tokPosn p625]
[pfWrong p530]
[pfDocument p486]
[pfListOf p485]
[$ttok p??]
[$stok p??]
[$stack p??]
[$inputStream p??]

— **defun npParse** —

```
(defun |npParse| (stream)
 (let (|$ttok| |$stok| |$stack| |$inputStream| found)
 (declare (special |$ttok| |$stack| |$inputStream| |$stok|))
  (setq |$inputStream| stream)
  (setq |$stack| nil)
  (setq |$stok| nil)
  (setq |$ttok| nil)
  (|npFirstTok|)
  (setq found (catch 'trappoint (|npItem|)))
  (cond
   ((eq found 'trapped)
    (|ncSoftError| (|tokPosn| |$stok|) "syntax error at top level" nil)
    (|pfWrong| (|pfDocument| "top level syntax error") (|pfListOf| nil)))
   ((null (null |$inputStream|))
    (|ncSoftError| (|tokPosn| |$stok|) " Improper syntax." nil)
    (|pfWrong|
     (|pfDocument| (list "input stream not exhausted"))
     (|pfListOf| nil)))
   ((null |$stack|)
```

```
    (|ncSoftError| (|tokPosn| |$stok|)
      "System error while parsing, stack is empty." nil)
    (|pfWrong| (|pfDocument| (list "stack empty")) (|pfListOf| nil)))
   (t (car |$stack|))))))
```

———-

14.0.229 defun npItem

[npQualDef p392]
[npEqKey p392]
[npItem1 p390]
[npPop1 p391]
[pfEnSequence p501]
[npPush p391]
[pfNovalue p515]

— **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|)))))))
```

———-

14.0.230 defun npItem1

[npQualDef p392]
[npEqKey p392]
[npItem1 p390]
[npPop1 p391]

— **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))))
```

———-

14.0.231 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 p623]
[tokPosn p625]
[tokPart p625]
[$ttok p??]
[$stok p??]
[$inputStream p??]

— **defun npFirstTok** —

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

———-

14.0.232 defun Push one item onto $stack

[$stack p??]

— **defun npPush 0** —

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

———-

14.0.233 defun Pop one item off $stack

[$stack p??]

— **defun npPop1 0** —

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

```
 (pop |$stack|))
```

———-

14.0.234 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))
```

———-

14.0.235 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))
```

———-

14.0.236 defun npQualDef

[npComma p393]
[npPush p391]
[npPop1 p391]

— **defun npQualDef** —

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

———-

14.0.237 defun Advance over a keyword

Test for the keyword, if found advance the token stream

[npNext p393]
[$ttok p??]
[$stok p??]

— **defun npEqKey** —

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

————-

14.0.238 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 p391]
[$inputStream p??]

— **defun npNext** —

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

————-

14.0.239 defun npComma

[npTuple p393]
[npQualifiedDefinition p394]

— **defun npComma** —

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

————-

14.0.240 defun npTuple

[npListofFun p459]
[npCommaBackSet p394]
[pfTupleListOf p526]

— **defun npTuple** —

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

———-

14.0.241 defun npCommaBackSet

[npEqKey p392]

— **defun npCommaBackSet** —

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

———-

14.0.242 defun npQualifiedDefinition

[npQualified p394]
[npDefinitionOrStatement p395]

— **defun npQualifiedDefinition** —

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

———-

14.0.243 defun npQualified

[npEqKey p392]
[npDefinition p412]
[npTrap p452]
[npPush p391]
[pfWhere p528]
[npPop1 p391]
[npLetQualified p412]

— **defun npQualified** —

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

———-

14.0.244 defun npDefinitionOrStatement

[npBackTrack p395]
[npGives p395]
[npDef p430]

— **defun npDefinitionOrStatement** —

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

———-

14.0.245 defun npBackTrack

[npState p452]
[npEqPeek p399]
[npRestore p398]
[npTrap p452]

— **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)))))
```

———-

14.0.246 defun npGives

[npBackTrack p395]
[npExit p455]
[npLambda p396]

— **defun npGives** —

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

———-

14.0.247 defun npLambda

[npVariable p453]
[npLambda p396]
[npTrap p452]
[npPush p391]
[pfLam p509]
[npPop2 p392]
[npPop1 p391]
[npEqKey p392]
[npDefinitionOrStatement p395]
[npType p396]
[pfReturnTyped p520]

— **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|))))))
```

———-

14.0.248 defun npType

[npMatch p397]
[npPop1 p391]
[npWith p397]
[npPush p391]

— **defun npType** —

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

———-

14.0.249 defun npMatch

[npLeftAssoc p447]
[npSuch p397]

— **defun npMatch** —

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

————-

14.0.250 defun npSuch

[npLeftAssoc p447]
[npLogical p439]

— **defun npSuch** —

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

————-

14.0.251 defun npWith

[npEqKey p392]
[npState p452]
[npCategoryL p399]
[npTrap p452]
[npEqPeek p399]
[npRestore p398]
[npVariable p453]
[npCompMissing p398]
[npPush p391]
[pfWith p530]
[npPop2 p392]
[npPop1 p391]
[pfNothing p486]

— **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)))))))
```

———-

14.0.252 defun npCompMissing

[npEqKey p392]
[npMissing p398]

— **defun npCompMissing** —

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

———-

14.0.253 defun npMissing

[trappoint p??]
[ncSoftError p573]
[tokPosn p625]
[pname p1106]
[$stok p??]

— **defun npMissing** —

```
(defun |npMissing| (s)
 (declare (special |$stok|))
  (|ncSoftError| (|tokPosn| |$stok|) "Possibly missing a %1" (list (pname s)))
  (throw 'trappoint 'trapped))
```

———-

14.0.254 defun npRestore

[npFirstTok p391]
[$stack p??]
[$inputStream p??]

— **defun npRestore** —

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

```
  (|npFirstTok|)
  (setq |$stack| (cdr x))
  t)
```

———-

14.0.255 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|)))
```

———-

14.0.256 defun npCategoryL

[npCategory p399]
[npPush p391]
[pfUnSequence p527]
[npPop1 p391]

— **defun npCategoryL** —

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

———-

14.0.257 defun npCategory

[npPP p450]
[npSCategory p400]

— **defun npCategory** —

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

———-

14.0.258 defun npSCategory

[npWConditional p437]
[npCategoryL p399]
[npPush p391]
[npPop1 p391]
[npDefaultValue p437]
[npState p452]
[npPrimary p403]
[npEqPeek p399]
[npRestore p398]
[npSignature p400]
[npApplication p407]
[pfAttribute p493]
[npTrap p452]

— **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))))))
```

———-

14.0.259 defun npSignature

[npSigItemlist p401]
[npPush p391]
[pfWDec p527]
[pfNothing p486]
[npPop1 p391]

— **defun npSignature** —

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

———-

14.0.260 defun npSigItemlist

[npListing p401]
[npSigItem p402]
[npPush p391]
[pfListOf p485]
[pfAppend p494]
[pfParts p489]
[npPop1 p391]

— **defun npSigItemlist** —

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

———-

14.0.261 defun npListing

[npList p401]
[pfListOf p485]

— **defun npListing** —

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

———-

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

[npEqKey p392]
[npTrap p452]
[npPush p391]
[npPop3 p392]
[npPop2 p392]
[npPop1 p391]
[$stack p??]

— **defun npList** —

```
(defun |npList| (f str1 fn)
 (let (a)
 (declare (special |$stack|))
  (cond
   ((apply f nil)
```

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

———-

14.0.263 defun npSigItem

[npTypeVariable p402]
[npSigDecl p403]
[npTrap p452]

— **defun npSigItem** —

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

———-

14.0.264 defun npTypeVariable

[npParenthesized p454]
[npTypeVariablelist p403]
[npSignatureDefinee p403]
[npPush p391]
[pfListOf p485]
[npPop1 p391]

— **defun npTypeVariable** —

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

———-

14.0.265 defun npSignatureDefinee

[npName p445]
[npInfixOperator p406]
[npPrefixColon p407]

— **defun npSignatureDefinee** —

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

———-

14.0.266 defun npTypeVariablelist

[npListing p401]
[npSignatureDefinee p403]

— **defun npTypeVariablelist** —

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

———-

14.0.267 defun npSigDecl

[npEqKey p392]
[npType p396]
[npTrap p452]
[npPush p391]
[pfSpread p480]
[pfParts p489]
[npPop2 p392]
[npPop1 p391]

— **defun npSigDecl** —

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

———-

14.0.268 defun npPrimary

[npPrimary1 p409]
[npPrimary2 p404]

— **defun npPrimary** —

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

————-

14.0.269 defun npPrimary2

[npEncAp p425]
[npAtom2 p405]
[npAdd p405]
[pfNothing p486]
[npWith p397]

— **defun npPrimary2** —

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

————-

14.0.270 defun npADD

TPDHERE: Note that there is also an npAdd function

[npType p396]
[npPop1 p391]
[npAdd p405]
[npPush p391]

— **defun npADD** —

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

————-

14.0.271 defun npAdd

TPDHERE: Note that there is also an npADD function

[npEqKey p392]
[npState p452]
[npDefinitionOrStatement p395]
[npTrap p452]
[npEqPeek p399]
[npRestore p398]
[npVariable p453]
[npCompMissing p398]
[npDefinitionOrStatement p395]
[npPush p391]
[pfAdd p492]
[npPop2 p392]
[npPop1 p391]
[pfNothing p486]

— **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)))))))))
```

————-

14.0.272 defun npAtom2

[npInfixOperator p406]
[npAmpersand p444]
[npPrefixColon p407]
[npFromdom p444]

— **defun npAtom2** —

```
(defun |npAtom2| ()
```

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

———-

14.0.273 defun npInfixOperator

[npInfixOp p406]
[npState p452]
[npEqKey p392]
[npInfixOp p406]
[npPush p391]
[pfSymb p491]
[npPop1 p391]
[tokPosn p625]
[npRestore p398]
[tokConstruct p623]
[tokPart p625]
[$stok p??]

— **defun npInfixOperator** —

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

———-

14.0.274 defun npInfixOp

[npPushId p449]
[$ttok p??]
[$stok p??]

— **defun npInfixOp** —

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

————-

14.0.275 defun npPrefixColon

[npEqPeek p399]
[npPush p391]
[tokConstruct p623]
[tokPosn p625]
[npNext p393]
[$stok p??]

— **defun npPrefixColon** —

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

————-

14.0.276 defun npApplication

[npDotted p408]
[npPrimary p403]
[npApplication2 p409]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]

— **defun npApplication** —

```
(defun |npApplication| ()
 (and
  (|npDotted| #'|npPrimary|)
  (or
   (and
    (|npApplication2|)
```

```
    (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))))
    t)))
```

———-

14.0.277 defun npDotted

— **defun npDotted** —

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

———-

14.0.278 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)
```

———-

14.0.279 defun npSelector

[npEqKey p392]
[npPrimary p403]
[npTrap p452]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]

— **defun npSelector** —

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

———-

14.0.280 defun npApplication2

[npDotted p408]
[npPrimary1 p409]
[npApplication2 p409]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]

— **defun npApplication2** —

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

———-

14.0.281 defun npPrimary1

[npEncAp p425]
[npAtom1 p426]
[npLet p411]
[npFix p411]
[npMacro p410]
[npBPileDefinition p430]
[npDefn p429]
[npRule p436]

— **defun npPrimary1** —

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

———-

14.0.282 defun npMacro

[npPP p450]
[npMdef p410]

— defun npMacro —

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

————-

14.0.283 defun npMdef

TPDHERE: Beware that this function occurs with uppercase also

[npQuiver p439]
[pfCheckMacroOut p481]
[npPop1 p391]
[npDefTail p436]
[npTrap p452]
[npPop1 p391]
[npPush p391]
[pfMacro p513]
[pfPushMacroBody p484]

— defun npMdef —

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

————-

14.0.284 defun npMDEF

TPDHERE: Beware that this function occurs with lowercase also

[npBackTrack p395]
[npStatement p415]
[npMDEFinition p411]

— **defun npMDEF** —

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

————-

14.0.285 defun npMDEFinition

[npPP p450]
[npMdef p410]

— **defun npMDEFinition** —

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

————-

14.0.286 defun npFix

[npEqKey p392]
[npDef p430]
[npPush p391]
[pfFix p503]
[npPop1 p391]

— **defun npFix** —

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

————-

14.0.287 defun npLet

[npLetQualified p412]
[npDefinitionOrStatement p395]

— **defun npLet** —

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

————-

14.0.288 defun npLetQualified

[npEqKey p392]
[npDefinition p412]
[npTrap p452]
[npCompMissing p398]
[npPush p391]
[pfWhere p528]
[npPop2 p392]
[npPop1 p391]

— **defun npLetQualified** —

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

———-

14.0.289 defun npDefinition

[npPP p450]
[npDefinitionItem p412]
[npPush p391]
[pfSequenceToList p480]
[npPop1 p391]

— **defun npDefinition** —

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

———-

14.0.290 defun npDefinitionItem

[npTyping p413]
[npImport p424]
[npState p452]
[npStatement p415]
[npEqPeek p399]
[npRestore p398]
[npDef p430]
[npMacro p410]

[npDefn p429]
[npTrap p452]

— **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|)))))))
```

———-

14.0.291 defun npTyping

[npEqKey p392]
[npDefaultItemlist p413]
[npTrap p452]
[npPush p391]
[pfTyping p526]
[npPop1 p391]

— **defun npTyping** —

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

———-

14.0.292 defun npDefaultItemlist

[npPC p??]
[npSDefaultItem p414]
[npPush p391]
[pfUnSequence p527]
[npPop1 p391]

— **defun npDefaultItemlist** —

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

————-

14.0.293 defun npSDefaultItem

[npListing p401]
[npDefaultItem p414]
[npPush p391]
[pfAppend p494]
[pfParts p489]
[npPop1 p391]

— **defun npSDefaultItem** —

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

————-

14.0.294 defun npDefaultItem

[npTypeVariable p402]
[npDefaultDecl p414]
[npTrap p452]

— **defun npDefaultItem** —

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

————-

14.0.295 defun npDefaultDecl

[npEqKey p392]
[npType p396]
[npTrap p452]
[npPush p391]
[pfSpread p480]
[pfParts p489]

[npPop2 p392]
[npPop1 p391]

— **defun npDefaultDecl** —

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

———-

14.0.296 defun npStatement

[npExpress p423]
[npLoop p419]
[npIterate p418]
[npReturn p422]
[npBreak p419]
[npFree p418]
[npImport p424]
[npInline p418]
[npLocal p417]
[npExport p415]
[npTyping p413]
[npVoid p422]

— **defun npStatement** —

```
(defun |npStatement| ()
 (or
  (|npExpress|)
  (|npLoop|)
  (|npIterate|)
  (|npReturn|)
  (|npBreak|)
  (|npFree|)
  (|npImport|)
  (|npInline|)
  (|npLocal|)
  (|npExport|)
  (|npTyping|)
  (|npVoid|)))
```

———-

14.0.297 defun npExport

[npEqKey p392]
[npLocalItemlist p416]

[npTrap p452]
[npPush p391]
[pfExport p502]
[npPop1 p391]

— **defun npExport** —

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

————-

14.0.298 defun npLocalItemlist

[npPC p??]
[npSLocalItem p416]
[npPush p391]
[pfUnSequence p527]
[npPop1 p391]

— **defun npLocalItemlist** —

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

————-

14.0.299 defun npSLocalItem

[npListing p401]
[npLocalItem p417]
[npPush p391]
[pfAppend p494]
[pfParts p489]
[npPop1 p391]

— **defun npSLocalItem** —

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

————-

14.0.300 defun npLocalItem

[npTypeVariable p402]
[npLocalDecl p417]

— **defun npLocalItem** —

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

————-

14.0.301 defun npLocalDecl

[npEqKey p392]
[npType p396]
[npTrap p452]
[npPush p391]
[pfSpread p480]
[pfParts p489]
[npPop2 p392]
[npPop1 p391]
[pfNothing p486]

— **defun npLocalDecl** —

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

————-

14.0.302 defun npLocal

[npEqKey p392]
[npLocalItemlist p416]
[npTrap p452]
[npPush p391]
[pfLocal p511]
[npPop1 p391]

— **defun npLocal** —

```
(defun |npLocal| ()
 (and
```

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

———-

14.0.303 defun npFree

[npEqKey p392]
[npLocalItemlist p416]
[npTrap p452]
[npPush p391]
[pfFree p503]
[npPop1 p391]

— **defun npFree** —

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

———-

14.0.304 defun npInline

[npAndOr p425]
[npQualTypelist p424]
[pfInline p509]

— **defun npInline** —

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

———-

14.0.305 defun npIterate

[npEqKey p392]
[npPush p391]
[pfIterate p508]
[pfNothing p486]

— **defun npIterate** —

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

———-

14.0.306 defun npBreak

[npEqKey p392]
[npPush p391]
[pfBreak p497]
[pfNothing p486]

— **defun npBreak** —

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

———-

14.0.307 defun npLoop

[npIterators p419]
[npCompMissing p398]
[npAssign p456]
[npTrap p452]
[npPush p391]
[pfLp p513]
[npPop2 p392]
[npPop1 p391]
[npEqKey p392]
[pfLoop1 p512]

— **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|))))))
```

———-

14.0.308 defun npIterators

[npForIn p421]
[npZeroOrMore p421]
[npIterator p420]

[npPush p391]
[npPop2 p392]
[npPop1 p391]
[npWhile p421]
[npIterators p419]

— **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|)))))))
```

———-

14.0.309 defun npIterator

[npForIn p421]
[npSuchThat p420]
[npWhile p421]

— **defun npIterator** —

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

———-

14.0.310 defun npSuchThat

[npAndOr p425]
[npLogical p439]
[pfSuchthat p522]

— **defun npSuchThat** —

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

———-

14.0.311 defun Apply argument 0 or more times

[npPush p391]
[npPop2 p392]
[npPop1 p391]
[$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)))))
```

———-

14.0.312 defun npWhile

[npAndOr p425]
[npLogical p439]
[pfWhile p529]

— **defun npWhile** —

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

———-

14.0.313 defun npForIn

[npEqKey p392]
[npVariable p453]
[npTrap p452]
[npCompMissing p398]
[npBy p441]
[npPush p391]
[pfForin p504]
[npPop2 p392]
[npPop1 p391]

— **defun npForIn** —

```
(defun |npForIn| ()
```

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

———-

14.0.314 defun npReturn

[npEqKey p392]
[npExpress p423]
[npPush p391]
[pfNothing p486]
[npEqKey p392]
[npName p445]
[npTrap p452]
[pfReturn p519]
[npPop2 p392]
[npPop1 p391]
[pfReturnNoName p520]

— **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|))))))
```

———-

14.0.315 defun npVoid

[npAndOr p425]
[npStatement p415]
[pfNovalue p515]

— **defun npVoid** —

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

———-

14.0.316 defun npExpress

[npExpress1 p423]
[npIterators p419]
[npPush p391]
[pfCollect p499]
[npPop2 p392]
[pfListOf p485]
[npPop1 p391]

— **defun npExpress** —

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

———-

14.0.317 defun npExpress1

[npConditionalStatement p423]
[npADD p404]

— **defun npExpress1** —

```
(defun |npExpress1| ()
 (or (|npConditionalStatement|) (|npADD|)))
```

———-

14.0.318 defun npConditionalStatement

[npConditional p437]
[npQualifiedDefinition p394]

— **defun npConditionalStatement** —

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

———-

14.0.319 defun npImport

[npAndOr p425]
[npQualTypelist p424]
[pfImport p508]

— **defun npImport** —

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

———-

14.0.320 defun npQualTypelist

[npPC p**??**]
[npSQualTypelist p424]
[npPush p391]
[pfUnSequence p527]
[npPop1 p391]

— **defun npQualTypelist** —

```
(defun |npQualTypelist| ()
 (and
  (|npPC| #'|npSQualTypelist|)
  (|npPush| (|pfUnSequence| (|npPop1|)))))
```

———-

14.0.321 defun npSQualTypelist

[npListing p401]
[npQualType p425]
[npPush p391]
[pfParts p489]
[npPop1 p391]

— **defun npSQualTypelist** —

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

———-

14.0.322 defun npQualType

[npType p396]
[npPush p391]
[pfQualType p518]
[npPop1 p391]
[pfNothing p486]

— **defun npQualType** —

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

————-

14.0.323 defun npAndOr

[npEqKey p392]
[npTrap p452]
[npPush p391]
[npPop1 p391]

— **defun npAndOr** —

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

————-

14.0.324 defun npEncAp

[npAnyNo p408]
[npEncl p426]
[npFromdom p444]

— **defun npEncAp** —

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

————-

14.0.325 defun npEncl

[npBDefinition p428]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]

— **defun npEncl** —

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

———-

14.0.326 defun npAtom1

[npPDefinition p426]
[npName p445]
[npConstTok p427]
[npDollar p427]
[npBDefinition p428]
[npFromdom p444]

— **defun npAtom1** —

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

———-

14.0.327 defun npPDefinition

[npParenthesized p454]
[npDefinitionlist p435]
[npPush p391]
[pfEnSequence p501]
[npPop1 p391]

— **defun npPDefinition** —

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

———-

14.0.328 defun npDollar

[npEqPeek p399]
[npPush p391]
[tokConstruct p623]
[tokPosn p625]
[npNext p393]
[$stok p**??**]

— **defun npDollar** —

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

———-

14.0.329 defun npConstTok

[tokType p625]
[npPush p391]
[npNext p393]
[npEqPeek p399]
[npState p452]
[npPrimary1 p409]
[pfSymb p491]
[npPop1 p391]
[tokPosn p625]
[npRestore p398]
[$stok p**??**]

— **defun npConstTok** —

```
(defun |npConstTok| ()
 (let (b a)
 (declare (special |$stok|))
  (cond
   ((member (|tokType| |$stok|) '(|integer| |string| |char| |float| |command|))
    (|npPush| |$stok|)
    (|npNext|))
   (((|npEqPeek| '|'|)
    (setq a |$stok|)
    (setq b (|npState|))
    (|npNext|)
    (cond
```

```
      ((and (|npPrimary1|)
            (|npPush| (|pfSymb| (|npPop1|) (|tokPosn| a))))
       t)
      (t (|npRestore| b) nil)))
   (t nil))))
```

———-

14.0.330 defun npBDefinition

[npPDefinition p426]
[npBracketed p428]
[npDefinitionlist p435]

— **defun npBDefinition** —

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

———-

14.0.331 defun npBracketed

[npParened p428]
[npBracked p429]
[npBraced p429]
[npAngleBared p429]

— **defun npBracketed** —

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

———-

14.0.332 defun npParened

[npEnclosed p451]
[pfParen p517]

— **defun npParened** —

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

```
    (|npEnclosed| '|(\|| '|\|)| #'|pfParen| f)))
```

———-

14.0.333 defun npBracked

[npEnclosed p451]
[pfBracket p496]
[pfBracketBar p496]

— **defun npBracked** —

```
(defun |npBracked| (f)
 (or (|npEnclosed| '[ '] #'|pfBracket| f)
     (|npEnclosed| '|[\|| '|\|]| #'|pfBracketBar| f)))
```

———-

14.0.334 defun npBraced

[npEnclosed p451]
[pfBrace p496]
[pfBraceBar p496]

— **defun npBraced** —

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

———-

14.0.335 defun npAngleBared

[npEnclosed p451]
[pfHide p506]

— **defun npAngleBared** —

```
(defun |npAngleBared| (f)
  (|npEnclosed| '|<\|| '|\|>| #'|pfHide| f))
```

———-

14.0.336 defun npDefn

[npEqKey p392]
[npPP p450]
[npDef p430]

— **defun npDefn** —

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

————-

14.0.337 defun npDef

[npMatch p397]
[pfCheckItOut p480]
[npPop1 p391]
[npDefTail p436]
[npTrap p452]
[npPop1 p391]
[npPush p391]
[pfDefinition p499]
[pfPushBody p489]

— **defun npDef** —

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

————-

14.0.338 defun npBPileDefinition

[npPileBracketed p431]
[npPileDefinitionlist p431]
[npPush p391]
[pfSequence p521]
[pfListOf p485]
[npPop1 p391]

— **defun npBPileDefinition** —

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

———-

14.0.339 defun npPileBracketed

[npEqKey p392]
[npPush p391]
[pfNothing p486]
[npMissing p398]
[pfPile p489]
[npPop1 p391]

— **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)))
```

———-

14.0.340 defun npPileDefinitionlist

[npListAndRecover p432]
[npDefinitionlist p435]
[npPush p391]
[pfAppend p494]
[npPop1 p391]

— **defun npPileDefinitionlist** —

```
(defun |npPileDefinitionlist| ()
 (and
  (|npListAndRecover| #'|npDefinitionlist|)
  (|npPush| (|pfAppend| (|npPop1|)))))
```

———-

14.0.341 defun npListAndRecover

[trappoint p??]
[npRecoverTrap p433]
[syGeneralErrorHere p434]
[npEqKey p392]
[npEqPeek p399]
[npNext p393]
[npPop1 p391]
[npPush p391]
[$inputStream p??]
[$stack p??]

— **defun npListAndRecover** —

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

———-

14.0.342 defun npRecoverTrap

[npFirstTok p391]
[tokPosn p625]
[npMoveTo p433]
[syIgnoredFromTo p434]
[npPush p391]
[pfWrong p530]
[pfDocument p486]
[pfListOf p485]
[$stok p??]

— **defun npRecoverTrap** —

```
(defun |npRecoverTrap| ()
 (let (pos2 pos1)
 (declare (special |$stok|))
  (|npFirstTok|)
  (setq pos1 (|tokPosn| |$stok|))
  (|npMoveTo| 0)
  (setq pos2 (|tokPosn| |$stok|))
  (|syIgnoredFromTo| pos1 pos2)
  (|npPush|
   (list (|pfWrong| (|pfDocument| (list "pile syntax error"))
                    (|pfListOf| nil))))))
```

————-

14.0.343 defun npMoveTo

[npEqPeek p399]
[npNext p393]
[npMoveTo p433]
[npEqKey p392]
[$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|))))
```

———-

14.0.344 defun syIgnoredFromTo

[pfGlobalLinePosn p477]
[ncSoftError p573]
[FromTo p598]
[From p597]
[To p598]

— **defun syIgnoredFromTo** —

```
(defun |syIgnoredFromTo| (pos1 pos2)
 (cond
  ((equal (|pfGlobalLinePosn| pos1) (|pfGlobalLinePosn| pos2))
   (|ncSoftError| (|FromTo| pos1 pos2) "Ignored." nil))
  (t
   (|ncSoftError| (|From| pos1) "Ignored from here" nil)
   (|ncSoftError| (|To| pos2) "to here." nil))))
```

———-

14.0.345 defun syGeneralErrorHere

[sySpecificErrorHere p434]

— **defun syGeneralErrorHere** —

```
(defun |syGeneralErrorHere| ()
 (|sySpecificErrorHere| " Improper syntax." nil))
```

———-

14.0.346 defun sySpecificErrorHere

[sySpecificErrorAtToken p435]
[$stok p??]

— **defun sySpecificErrorHere** —

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

———-

14.0.347 defun sySpecificErrorAtToken

[ncSoftError p573]
[tokPosn p625]

— **defun sySpecificErrorAtToken** —

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

———-

14.0.348 defun npDefinitionlist

[npSemiListing p435]
[npQualDef p392]

— **defun npDefinitionlist** —

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

———-

14.0.349 defun npSemiListing

[npListofFun p459]
[npSemiBackSet p435]
[pfAppend p494]

— **defun npSemiListing** —

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

———-

14.0.350 defun npSemiBackSet

[npEqKey p392]

— **defun npSemiBackSet** —

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

———-

14.0.351 defun npRule

[npEqKey p392]
[npPP p450]
[npSingleRule p436]

— **defun npRule** —

```
(defun |npRule| ()
 (and
  (|npEqKey| 'rule)
  (|npPP| #'|npSingleRule|)))
```

————-

14.0.352 defun npSingleRule

[npQuiver p439]
[npDefTail p436]
[npTrap p452]
[npPush p391]
[pfRule p520]
[npPop2 p392]
[npPop1 p391]

— **defun npSingleRule** —

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

————-

14.0.353 defun npDefTail

[npEqKey p392]
[npDefinitionOrStatement p395]

— **defun npDefTail** —

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

————-

14.0.354 defun npDefaultValue

[npEqKey p392]
[npDefinitionOrStatement p395]
[npTrap p452]
[npPush p391]
[pfAdd p492]
[pfNothing p486]
[npPop1 p391]

— **defun npDefaultValue** —

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

———-

14.0.355 defun npWConditional

[npConditional p437]
[npPush p391]
[pfTweakIf p524]
[npPop1 p391]

— **defun npWConditional** —

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

———-

14.0.356 defun npConditional

[npEqKey p392]
[npLogical p439]
[npTrap p452]
[npMissing p398]
[npElse p438]

— **defun npConditional** —

```
(defun |npConditional| (f)
 (cond
  ((and (|npEqKey| 'IF)
        (or (|npLogical|) (|npTrap|))
        (or (|npEqKey| 'backset) t))
   (cond
    ((|npEqKey| 'settab)
```

```
    (cond
     ((|npEqKey| 'then)
      (and (or (apply f nil) (|npTrap|))
           (|npElse| f)
           (|npEqKey| 'backtab)))
     (t (|npMissing| '|then|))))
   ((|npEqKey| 'then)
    (and (or (apply f nil) (|npTrap|)) (|npElse| f)))
   (t (|npMissing| '|then|))))
  (t nil)))
```

———-

14.0.357 defun npElse

[npState p452]
[npBacksetElse p438]
[npTrap p452]
[npPush p391]
[pfIf p506]
[npPop3 p392]
[npPop2 p392]
[npPop1 p391]
[npRestore p398]
[pfIfThenOnly p507]

— **defun npElse** —

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

———-

14.0.358 defun npBacksetElse

TPDHERE: Well this makes no sense.

[npEqKey p392]

— **defun npBacksetElse** —

```
(defun |npBacksetElse| ()
 (if (|npEqKey| 'backset)
```

```
  (|npEqKey| 'else)
  (|npEqKey| 'else)))
```

———-

14.0.359 defun npLogical

[npLeftAssoc p447]
[npDisjand p439]

— **defun npLogical** —

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

———-

14.0.360 defun npDisjand

[npLeftAssoc p447]
[npDiscrim p439]

— **defun npDisjand** —

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

———-

14.0.361 defun npDiscrim

[npLeftAssoc p447]
[npQuiver p439]

— **defun npDiscrim** —

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

———-

14.0.362 defun npQuiver

[npRightAssoc p446]
[npRelation p440]

— **defun npQuiver** —

```
(defun |npQuiver| ()
```

```
 (|npRightAssoc| '(arrow larrow) #'|npRelation|))
```

———-

14.0.363 defun npRelation

[npLeftAssoc p447]
[npSynthetic p440]

— **defun npRelation** —

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

———-

14.0.364 defun npSynthetic

[npBy p441]
[npAmpersandFrom p443]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]
[pfInfApplication p508]

— **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)))
```

———-

14.0.365 defun npBy

[npLeftAssoc p447]
[npInterval p441]

— **defun npBy** —

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

———-

14.0.366 defun

[npArith p442]
[npSegment p441]
[npEqPeek p399]
[npPush p391]
[pfApplication p493]
[npPop1 p391]
[pfInfApplication p508]
[npPop2 p392]

— **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)))
```

———-

14.0.367 defun npSegment

[npEqPeek p399]
[npPushId p449]
[npFromdom p444]

— **defun npSegment** —

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

———-

14.0.368 defun npArith

[npLeftAssoc p447]
[npSum p442]

— **defun npArith** —

```
(defun |npArith| ()
 (|npLeftAssoc| '(mod) #'|npSum|))
```

———-

14.0.369 defun npSum

[npLeftAssoc p447]
[npTerm p442]

— **defun npSum** —

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

———-

14.0.370 defun npTerm

[npInfGeneric p448]
[npRemainder p443]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]

— **defun npTerm** —

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

———-

14.0.371 defun npRemainder

[npLeftAssoc p447]
[npProduct p443]

— **defun npRemainder** —

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

————-

14.0.372 defun npProduct

[npLeftAssoc p447]
[npPower p443]

— **defun npProduct** —

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

————-

14.0.373 defun npPower

[npRightAssoc p446]
[npColon p457]

— **defun npPower** —

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

————-

14.0.374 defun npAmpersandFrom

[npAmpersand p444]
[npFromdom p444]

— **defun npAmpersandFrom** —

```
(defun |npAmpersandFrom| ()
 (and (|npAmpersand|) (|npFromdom|)))
```

————-

14.0.375 defun npFromdom

[npEqKey p392]
[npApplication p407]
[npTrap p452]
[npFromdom1 p444]
[npPop1 p391]
[npPush p391]
[pfFromDom p505]

— **defun npFromdom** —

```
(defun |npFromdom| ()
 (or
  (and
   (|npEqKey| '$)
   (or (|npApplication|) (|npTrap|))
   (|npFromdom1| (|npPop1|))
   (|npPush| (|pfFromDom| (|npPop1|) (|npPop1|))))
  t))
```

————-

14.0.376 defun npFromdom1

[npEqKey p392]
[npApplication p407]
[npTrap p452]
[npFromdom1 p444]
[npPop1 p391]
[npPush p391]
[pfFromDom p505]

— **defun npFromdom1** —

```
(defun |npFromdom1| (c)
 (or
  (and
   (|npEqKey| '$)
   (or (|npApplication|) (|npTrap|))
   (|npFromdom1| (|npPop1|))
   (|npPush| (|pfFromDom| (|npPop1|) c)))
  (|npPush| c)))
```

————-

14.0.377 defun npAmpersand

[npEqKey p392]
[npName p445]
[npTrap p452]

— **defun npAmpersand** —

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

———-

14.0.378 defun npName

[npId p445]
[npSymbolVariable p446]

— **defun npName** —

```
(defun |npName| ()
 (or (|npId|) (|npSymbolVariable|)))
```

———-

14.0.379 defvar $npTokToNames

— **initvars** —

```
(defvar |$npTokToNames| (list '~ '|#| '[] '{} '|[\|\|]| '|{\|\|}|))
```

———-

14.0.380 defun npId

[npPush p391]
[npNext p393]
[tokConstruct p623]
[tokPosn p625]
[$npTokToNames p445]
[$ttok p??]
[$stok p??]

— **defun npId** —

```
(defun |npId| ()
 (declare (special |$npTokToNames| |$ttok| |$stok|))
  (cond
   ((eq (caar |$stok|) '|id|)
     (|npPush| |$stok|)
     (|npNext|))
   ((and (eq (caar |$stok|) '|key|) (member |$ttok| |$npTokToNames|))
     (|npPush| (|tokConstruct| '|id| |$ttok| (|tokPosn| |$stok|)))
```

```
    (|npNext|))
  (t nil)))
```

———-

14.0.381 defun npSymbolVariable

[npState p452]
[npEqKey p392]
[npId p445]
[npPop1 p391]
[npPush p391]
[tokConstruct p623]
[tokPart p625]
[tokPosn p625]
[npRestore p398]

— **defun npSymbolVariable** —

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

———-

14.0.382 defun npRightAssoc

[npState p452]
[npInfGeneric p448]
[npRightAssoc p446]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]
[pfInfApplication p508]
[npRestore p398]

— **defun npRightAssoc** —

```
(defun |npRightAssoc| (o p)
 (let (a)
  (setq a (|npState|))
  (cond
   ((apply p nil)
      ((lambda ()
```

```
      (loop
       (cond
        ((not
          (and
           (|npInfGeneric| o)
           (or
            (|npRightAssoc| o p)
            (progn (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))) nil))))
         (return nil))
        (t
         (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|))))))
      t)
   (t
    (|npRestore| a)
    nil))))
```

———-

14.0.383 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 p448]
[npPush p391]
[pfApplication p493]
[npPop2 p392]
[npPop1 p391]
[pfInfApplication p508]

— **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))
```

———-

14.0.384 defun npInfGeneric

[npDDInfKey p448]
[npEqKey p392]

— **defun npInfGeneric** —

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

———-

14.0.385 defun npDDInfKey

[npInfKey p449]
[npState p452]
[npEqKey p392]
[npPush p391]
[pfSymb p491]
[npPop1 p391]
[tokPosn p625]
[npRestore p398]
[tokConstruct p623]
[tokPart p625]
[$stok p??]

— **defun npDDInfKey** —

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

```
((and (|npEqKey| 'backquote) (|npInfKey| s))
  (setq a (|npPop1|))
   (|npPush| (|tokConstruct| '|idsy| (|tokPart| a) (|tokPosn| a))))
(t
 (|npRestore| a)
 nil))))))))
```

———-

14.0.386 defun npInfKey

[npPushId p449]
[$stok p??]
[$ttok p??]

— **defun npInfKey** —

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

———-

14.0.387 defun npPushId

[tokConstruct p623]
[tokPosn p625]
[npNext p393]
[$stack p??]
[$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|)))
```

———-

14.0.388 defvar npPParg

— **initvars** —

```
(defvar *npPParg* nil "rewrite npPP without flets, using global scoping")
```

———-

14.0.389 defun npPP

This was rewritten by NAG to remove flet.

[npParened p428]
[npPPf p451]
[npPileBracketed p431]
[npPPg p450]
[npPush p391]
[pfEnSequence p501]
[npPop1 p391]
[npPParg p449]

— **defun npPP** —

```
(defun |npPP| (f)
 (declare (special *npPParg*))
  (setq *npPParg* f)
  (or
   (|npParened| #'npPPf)
   (and  (|npPileBracketed| #'npPPg) (|npPush| (|pfEnSequence| (|npPop1|))))
   (funcall f)))
```

———-

14.0.390 defun npPPff

[npPop1 p391]
[npPush p391]
[$npPParg p449]

— **defun npPPff** —

```
(defun npPPff ()
 (and (funcall *npPParg*) (|npPush| (list (|npPop1|)))))
```

———-

14.0.391 defun npPPg

[npListAndRecover p432]
[npPPf p451]
[npPush p391]
[pfAppend p494]
[npPop1 p391]

— **defun npPPg** —

```
(defun npPPg ()
 (and (|npListAndRecover| #'npPPf))
 (|npPush| (|pfAppend| (|npPop1|))))
```

———-

14.0.392 defun npPPf

[npSemiListing p435]
[npPPff p450]

— **defun npPPf** —

```
(defun npPPf ()
 (|npSemiListing| #'npPPff))
```

———-

14.0.393 defun npEnclosed

[npEqKey p392]
[npPush p391]
[pfTuple p526]
[pfListOf p485]
[npMissingMate p455]
[pfEnSequence p501]
[npPop1 p391]
[$stok p??]

— **defun npEnclosed** —

```
(defun |npEnclosed| (open close fn f)
 (let (a)
 (declare (special |$stok|))
  (setq a |$stok|)
  (when (|npEqKey| open)
   (cond
    ((|npEqKey| close)
     (|npPush| (funcall fn a (|pfTuple| (|pfListOf| NIL)))))
    ((and (apply f nil)
          (or (|npEqKey| close)
              (|npMissingMate| close a)))
     (|npPush| (funcall fn a (|pfEnSequence| (|npPop1|)))))
    ('t nil)))))
```

———-

14.0.394 defun npState

[$stack p??]
[$inputStream p??]

— **defun npState** —

```
(defun |npState| ()
 (declare (special |$stack| |$inputStream|))
  (cons |$inputStream| |$stack|))
```

————-

14.0.395 defun npTrap

[trappoint p??]
[tokPosn p625]
[ncSoftError p573]
[$stok p??]

— **defun npTrap** —

```
(defun |npTrap| ()
 (declare (special |$stok|))
  (|ncSoftError| (|tokPosn| |$stok|) " Improper syntax." nil)
  (throw 'trappoint 'trapped))
```

————-

14.0.396 defun npTrapForm

[trappoint p??]
[pfSourceStok p484]
[syGeneralErrorHere p434]
[ncSoftError p573]
[tokPosn p625]

— **defun npTrapForm** —

```
(defun |npTrapForm| (x)
 (let (a)
  (setq a (|pfSourceStok| x))
  (cond
   ((eq a '|NoToken|)
     (|syGeneralErrorHere|)
     (throw 'trappoint 'trapped))
   (t
    (|ncSoftError| (|tokPosn| a) " Improper syntax." nil)
    (throw 'trappoint 'trapped)))))
```

————-

14.0.397 defun npVariable

[npParenthesized p454]
[npVariablelist p453]
[npVariableName p453]
[npPush p391]
[pfListOf p485]
[npPop1 p391]

— **defun npVariable** —

```
(defun |npVariable| ()
 (or
  (|npParenthesized| #'|npVariablelist|)
  (and (|npVariableName|) (|npPush| (|pfListOf| (list (|npPop1|)))))))
```

———-

14.0.398 defun npVariablelist

[npListing p401]
[npVariableName p453]

— **defun npVariablelist** —

```
(defun |npVariablelist| ()
 (|npListing| #'|npVariableName|))
```

———-

14.0.399 defun npVariableName

[npName p445]
[npDecl p454]
[npPush p391]
[pfTyped p525]
[npPop1 p391]
[pfNothing p486]

— **defun npVariableName** —

```
(defun |npVariableName| ()
 (and
  (|npName|)
  (or (|npDecl|) (|npPush| (|pfTyped| (|npPop1|) (|pfNothing|))))))
```

———-

14.0.400 defun npDecl

[npEqKey p392]
[npType p396]
[npTrap p452]
[npPush p391]
[pfTyped p525]
[npPop2 p392]
[npPop1 p391]

— **defun npDecl** —

```
(defun |npDecl| ()
 (and
  (|npEqKey| 'colon)
  (or (|npType|) (|npTrap|))
  (|npPush| (|pfTyped| (|npPop2|) (|npPop1|)))))
```

————-

14.0.401 defun npParenthesized

[npParenthesize p454]

— **defun npParenthesized** —

```
(defun |npParenthesized| (f)
 (or (|npParenthesize| '|(| '|)| f) (|npParenthesize| '|(\|| '|\|)| f)))
```

————-

14.0.402 defun npParenthesize

[npEqKey p392]
[npMissingMate p455]
[npPush p391]
[$stok p??]

— **defun npParenthesize** —

```
(defun |npParenthesize| (open close f)
 (let (a)
 (declare (special |$stok|))
  (setq a |$stok|)
  (cond
   ((|npEqKey| open)
    (cond
     ((and (apply f nil)
           (or (|npEqKey| close)
               (|npMissingMate| close a)))
      t)
```

```
    ((|npEqKey| close) (|npPush| nil))
    (t (|npMissingMate| close a))))
  (t nil))))
```

———-

14.0.403 defun npMissingMate

[ncSoftError p573]
[tokPosn p625]
[npMissing p398]

— **defun npMissingMate** —

```
(defun |npMissingMate| (close open)
 (|ncSoftError| (|tokPosn| open) "Missing mate." nil)
 (|npMissing| close))
```

———-

14.0.404 defun npExit

[npBackTrack p395]
[npAssign p456]
[npPileExit p455]

— **defun npExit** —

```
(defun |npExit| ()
 (|npBackTrack| #'|npAssign| 'exit #'|npPileExit|))
```

———-

14.0.405 defun npPileExit

[npAssign p456]
[npEqKey p392]
[npStatement p415]
[npPush p391]
[pfExit p501]
[npPop2 p392]
[npPop1 p391]

— **defun npPileExit** —

```
(defun |npPileExit| ()
 (and
  (|npAssign|)
  (or (|npEqKey| 'exit) (|npTrap|))
  (or (|npStatement|) (|npTrap|))
```

```
  (|npPush| (|pfExit| (|npPop2|) (|npPop1|)))))
```

———-

14.0.406 defun npAssign

[npBackTrack p395]
[npMDEF p410]
[npAssignment p456]

— **defun npAssign** —

```
(defun |npAssign| ()
 (|npBackTrack| #'|npMDEF| 'becomes #'|npAssignment|))
```

———-

14.0.407 defun npAssignment

[npAssignVariable p456]
[npEqKey p392]
[npTrap p452]
[npGives p395]
[npPush p391]
[pfAssign p494]
[npPop2 p392]
[npPop1 p391]

— **defun npAssignment** —

```
(defun |npAssignment| ()
 (and
  (|npAssignVariable|)
  (or (|npEqKey| 'becomes) (|npTrap|))
  (or (|npGives|) (|npTrap|))
  (|npPush| (|pfAssign| (|npPop2|) (|npPop1|)))))
```

———-

14.0.408 defun npAssignVariable

[npColon p457]
[npPush p391]
[pfListOf p485]
[npPop1 p391]

— **defun npAssignVariable** —

```
(defun |npAssignVariable| ()
 (and (|npColon|) (|npPush| (|pfListOf| (list (|npPop1|))))))
```

———-

14.0.409 defun npColon

[npTypified p458]
[npAnyNo p408]
[npTagged p457]

— **defun npColon** —

```
(defun |npColon| ()
 (and (|npTypified|) (|npAnyNo| #'|npTagged|)))
```

———-

14.0.410 defun npTagged

[npTypedForm1 p457]
[pfTagged p523]

— **defun npTagged** —

```
(defun |npTagged| ()
 (|npTypedForm1| 'colon #'|pfTagged|))
```

———-

14.0.411 defun npTypedForm1

[npEqKey p392]
[npType p396]
[npTrap p452]
[npPush p391]
[npPop2 p392]
[npPop1 p391]

— **defun npTypedForm1** —

```
(defun |npTypedForm1| (sy fn)
 (and
  (|npEqKey| sy)
  (or (|npType|) (|npTrap|))
  (|npPush| (funcall fn (|npPop2|) (|npPop1|)))))
```

———-

14.0.412 defun npTypified

[npApplication p407]
[npAnyNo p408]
[npTypeStyle p458]

— **defun npTypified** —

```
(defun |npTypified| ()
 (and (|npApplication|) (|npAnyNo| #'|npTypeStyle|)))
```

———-

14.0.413 defun npTypeStyle

[npCoerceTo p459]
[npRestrict p459]
[npPretend p458]
[npColonQuery p458]

— **defun npTypeStyle** —

```
(defun |npTypeStyle| ()
 (or (|npCoerceTo|) (|npRestrict|) (|npPretend|) (|npColonQuery|)))
```

———-

14.0.414 defun npPretend

[npTypedForm p459]
[pfPretend p517]

— **defun npPretend** —

```
(defun |npPretend| ()
 (|npTypedForm| 'pretend #'|pfPretend|))
```

———-

14.0.415 defun npColonQuery

[npTypedForm p459]
[pfRetractTo p519]

— **defun npColonQuery** —

```
(defun |npColonQuery| ()
 (|npTypedForm| 'atat #'|pfRetractTo|))
```

———-

14.0.416 defun npCoerceTo

[npTypedForm p459]
[pfCoerceto p497]

— **defun npCoerceTo** —

```
(defun |npCoerceTo| ()
 (|npTypedForm| 'coerce #'|pfCoerceto|))
```

———-

14.0.417 defun npTypedForm

[npEqKey p392]
[npApplication p407]
[npTrap p452]
[npPush p391]
[npPop2 p392]
[npPop1 p391]

— **defun npTypedForm** —

```
(defun |npTypedForm| (sy fn)
 (and
  (|npEqKey| sy)
  (or (|npApplication|) (|npTrap|))
  (|npPush| (funcall fn (|npPop2|) (|npPop1|)))))
```

———-

14.0.418 defun npRestrict

[npTypedForm p459]
[pfRestrict p518]

— **defun npRestrict** —

```
(defun |npRestrict| ()
 (|npTypedForm| 'at #'|pfRestrict|))
```

———-

14.0.419 defun npListofFun

[npTrap p452]
[npPush p391]
[npPop3 p392]
[npPop2 p392]
[npPop1 p391]

[$stack p**??**]

— **defun npListofFun** —

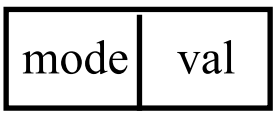
```
(defun |npListofFun| (f h g)
 (let (a)
 (declare (special |$stack|))
  (cond
   ((apply f nil)
    (cond
     ((and (apply h nil) (or (apply f nil) (|npTrap|)))
       (setq a |$stack|)
       (setq |$stack| nil)
       (do ()
           ((not (and (apply h nil)
                 (or (apply f nil) (|npTrap|))))))
       (setq |$stack| (cons (nreverse |$stack|) a))
       (|npPush| (funcall g (cons (|npPop3|) (cons (|npPop2|) (|npPop1|))))))
     (t t)))
   (t nil))))
```

———-

14.1 Functions on interpreter objects

Interpreter objects used to be called triples because they had the structure [value, type, environment]. For many years, the environment was not used, so finally in January, 1990, the structure of objects was changed to be (type . value). This was chosen because it was the structure of objects of type Any. Sometimes the values are wrapped (see the function isWrapped to see what this means physically). Wrapped values are not actual values belonging to their types. An unwrapped value must be evaluated to get an actual value. A wrapped value must be unwrapped before being passed to a library function. Typically, an unwrapped value in the interpreter consists of LISP code, e.g., parts of a function that is being constructed. – RSS 1/14/90

These are the new structure functions.

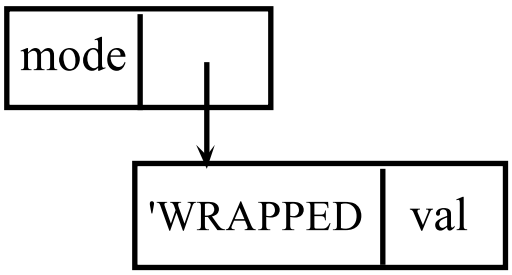


Object representation

14.1.1 defmacro mkObj

— **defmacro mkObj** —

```
(defmacro mkObj (val mode)
 `(cons ,mode ,val))
```



Object representation

14.1.2 defmacro mkObjWrap

[wrap p1104]

— **defmacro mkObjWrap** —

```
(defmacro mkObjWrap (val mode)
 `(cons ,mode (|wrap| ,val)))
```

14.1.3 defmacro mkObjCode

— **defmacro mkObjCode** —

```
(defmacro mkObjCode (val mode)
 `(cons 'cons (cons (mkq ,mode) (cons ,val nil))))
```

14.1.4 defmacro objSetVal

— **defmacro objSetVal** —

```
(defmacro |objSetVal| (obj val)
 `(rplacd ,obj ,val))
```

14.1.5 defmacro objSetMode

— **defmacro objSetMode** —

```
(defmacro |objSetMode| (obj mode)
```

```
 `(rplaca ,obj ,mode))
```

———-

14.1.6 defmacro objVal

— **defmacro objVal** —

```
(defmacro |objVal| (obj)
 `(cdr ,obj))
```

———-

14.1.7 defmacro objValUnwrap

— **defmacro objValUnwrap** —

```
(defmacro |objValUnwrap| (obj)
 `(|unwrap| (cdr ,obj)))
```

———-

14.1.8 defmacro objMode

— **defmacro objMode** —

```
(defmacro |objMode| (obj)
 `(car ,obj))
```

———-

14.1.9 defun objEnv

— **defun objEnv 0** —

```
(defun |objEnv| (obj)
 (declare (special $NE) (ignore obj))
 $NE)
```

———-

14.1.10 defmacro objCodeVal

— **defmacro objCodeVal** —

```
(defmacro |objCodeVal| (obj)
 `(caddr ,obj))
```

———-

14.1.11 defmacro objCodeMode

— **defmacro objCodeMode** —

```
(defmacro |objCodeMode| (obj)
 `(cadr ,obj))
```

———-

14.2 Macro handling

14.2.1 defun phMacro

TPDHERE: The pform function has a leading percent sign

`carrier[ptree,...] -> carrier[ptree, ptreePremacro,...]`

[ncEltQ p628]
[ncPutQ p628]
[macroExpanded p463]
[pform p??]

— **defun phMacro** —

```
(defun |phMacro| (carrier)
 (let (ptree)
  (setq ptree (|ncEltQ| carrier '|ptree|))
  (|ncPutQ| carrier '|ptreePremacro| ptree)
  (setq ptree (|macroExpanded| ptree))
  (|ncPutQ| carrier '|ptree| ptree)
  'ok))
```

———-

14.2.2 defun macroExpanded

$macActive is a list of the bodies being expanded. $posActive is a list of the parse forms where the bodies came from. [macExpand p464]
[$posActive p??]
[$macActive p??]

— **defun macroExpanded** —

```
(defun |macroExpanded| (pf)
 (let (|$posActive| |$macActive|)
```

```
 (declare (special |$posActive| |$macActive|))
  (setq |$macActive| nil)
  (setq |$posActive| nil)
  (|macExpand| pf)))
```

———-

14.2.3 defun macExpand

[pfWhere? p528]
[macWhere p469]
[pfLambda? p510]
[macLambda p469]
[pfMacro? p513]
[macMacro p470]
[pfId? p487]
[macId p468]
[pfApplication? p494]
[macApplication p464]
[pfMapParts p478]
[macExpand p464]

— **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))))
```

———-

14.2.4 defun macApplication

[pfMapParts p478]
[macExpand p464]
[pfApplicationOp p493]
[pfMLambda? p514]
[pf0ApplicationArgs p479]
[mac0MLambdaApply p465]
[$pfMacros p352]

— **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|)))))
```

———-

14.2.5 defun mac0MLambdaApply

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

[pf0MLambdaArgs p514]
[pfMLambdaBody p515]
[pfSourcePosition p479]
[ncHardError p573]
[pfId? p487]
[pform p??]
[mac0Define p471]
[mac0ExpandBody p466]
[$pfMacros p352]
[$posActive p??]
[$macActive p??]

— **defun mac0MLambdaApply** —

```
(defun |mac0MLambdaApply| (mlambda args opf |$pfMacros|)
 (declare (special |$pfMacros|))
 (let (pos body params)
 (declare (special |$posActive| |$macActive|))
  (setq params (|pf0MLambdaArgs| mlambda))
  (setq body (|pfMLambdaBody| mlambda))
  (cond
   ((not (eql (length args) (length params)))
    (setq pos (|pfSourcePosition| opf))
    (|ncHardError| pos "Expected %1 arguments, but received %2."
      (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 "Macro parameter %1f is not an id."
```

```
          (list (|%pform| p))))
        (t
         (|mac0Define| (|pfIdSymbol| p) '|mparam| a)))))
     (setq parms (cdr parms))
     (setq arrgs (cdr arrgs))))
   params nil args nil)
  (|mac0ExpandBody| body opf |$macActive| |$posActive|)))))
```

———-

14.2.6 defun mac0ExpandBody

[pfSourcePosition p479]
[mac0InfiniteExpansion p466]
[macExpand p464]
[$posActive p??]
[$macActive p??]

— **defun mac0ExpandBody** —

```
(defun |mac0ExpandBody| (body opf |$macActive| |$posActive|)
 (declare (special |$macActive| |$posActive|))
 (let (posn pf)
  (cond
   ((member body |$macActive|)
    (setq pf (cadr |$posActive|))
    (setq posn (|pfSourcePosition| pf))
    (|mac0InfiniteExpansion| posn body |$macActive|))
   (t
    (setq |$macActive| (cons body |$macActive|))
    (setq |$posActive| (cons opf |$posActive|))
    (|macExpand| body)))))
```

———-

14.2.7 defun mac0InfiniteExpansion

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

[mac0InfiniteExpansion,name p467]
[ncSoftError p573]
[pform p??]

— **defun mac0InfiniteExpansion** —

```
(defun |mac0InfiniteExpansion| (posn body active)
 (let (rnames fname tmp1 blist result)
  (setq blist (cons body active))
  (setq tmp1 (mapcar #'|mac0InfiniteExpansion,name| blist))
  (setq fname (car tmp1)) ;[fname, :rnames] := [name b for b in blist]
  (setq rnames (cdr tmp1))
  (|ncSoftError| posn
```

```
   "noRep Cycle in macro expansion: %1 %1y %2 %1.  Left as: %3f"
   (list
    (dolist (n (reverse rnames) (nreverse result))
      (setq result (append (reverse (list n "==>")) result)))
    fname (|%pform| body)))
  body))
```

———-

14.2.8 defun mac0InfiniteExpansion,name

[mac0GetName p467]
[pname p1106]

— **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))))))
```

———-

14.2.9 defun mac0GetName

Returns [state, body] or NIL. Returns [sy, state] or NIL.

[pfMLambdaBody p515]
[$pfMacros p352]

— **defun mac0GetName** —

```
(defun |mac0GetName| (body)
 (let (bd tmp1 st tmp2 sy name)
 (declare (special |$pfMacros|))
 ; for [sy,st,bd] in $pfMacros while not name repeat
  ((lambda (macros tmplist)
   (loop
    (cond
     ((or (atom macros)
          (progn (setq tmplist (car macros)) nil)
          name)
       (return nil))
     (t
      (and (consp tmplist)
```

```
          (progn
           (setq sy (car tmplist))
           (setq tmp2 (cdr tmplist))
           (and (consp tmp2)
                (progn
                 (setq st (car tmp2))
                 (setq tmp1 (cdr tmp2))
                 (and (consp tmp1)
                      (eq (cdr tmp1) nil)
                      (progn
                       (setq bd (car tmp1))
                       t)))))
          (progn
           (when (eq st '|mlambda|) (setq bd (|pfMLambdaBody| bd)))
           (when (eq bd body) (setq name (list sy st)))))))
         (setq macros (cdr macros))))
        |$pfMacros| nil)
       name))
```

———-

14.2.10 defun macId

[pfIdSymbol p487]
[mac0Get p469]
[pfCopyWithPos p478]
[pfSourcePosition p479]
[mac0ExpandBody p466]
[$posActive p??]
[$macActive p??]

— **defun macId** —

```
(defun |macId| (pf)
 (let (body state got sy)
 (declare (special |$posActive| |$macActive|))
  (setq sy (|pfIdSymbol| pf))
  (cond
   ((null (setq got (|mac0Get| sy))) pf)
   (t
    (setq state (car got))
    (setq body (cadr got))
    (cond
     ((eq state '|mparam|) body)
     ((eq state '|mlambda|) (|pfCopyWithPos| body (|pfSourcePosition| pf)))
     (t
      (|pfCopyWithPos|
       (|mac0ExpandBody| body pf |$macActive| |$posActive|)
       (|pfSourcePosition| pf))))))))
```

———-

14.2.11 defun mac0Get

[ifcdr p??]
[$pfMacros p352]

— **defun mac0Get** —

```
(defun |mac0Get| (sy)
 (declare (special |$pfMacros|))
 (ifcdr (assoc sy |$pfMacros|)))
```

————-

14.2.12 defun macWhere

[macWhere,mac p469]
[$pfMacros p352]

— **defun macWhere** —

```
(defun |macWhere| (pf)
 (declare (special |$pfMacros|))
 (|macWhere,mac| pf |$pfMacros|))
```

————-

14.2.13 defun macWhere,mac

[pfMapParts p478]
[macExpand p464]
[$pfMacros p352]

— **defun macWhere,mac** —

```
(defun |macWhere,mac| (pf |$pfMacros|)
 (declare (special |$pfMacros|))
 (|pfMapParts| #'|macExpand| pf))
```

————-

14.2.14 defun macLambda

[macLambda,mac p470]
[$pfMacros p352]

— **defun macLambda** —

```
(defun |macLambda| (pf)
 (declare (special |$pfMacros|))
 (|macLambda,mac| pf |$pfMacros|))
```

———-

14.2.15 defun macLambda,mac

[pfMapParts p478]
[macExpand p464]
[$pfMacros p352]

— **defun macLambda,mac** —

```
(defun |macLambda,mac| (pf |$pfMacros|)
 (declare (special |$pfMacros|))
 (|pfMapParts| #'|macExpand| pf))
```

———-

14.2.16 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 p513]
[pfMacroRhs p514]
[pfId? p487]
[ncSoftError p573]
[pfSourcePosition p479]
[pfIdSymbol p487]
[mac0Define p471]
[pform p**??**]
[pfMLambda? p514]
[macSubstituteOuter p471]
[pfNothing? p486]
[pfMacro p513]
[pfNothing p486]

— **defun macMacro** —

```
(defun |macMacro| (pf)
 (let (sy rhs lhs)
  (setq lhs (|pfMacroLhs| pf))
  (setq rhs (|pfMacroRhs| pf))
  (cond
   ((null (|pfId?| lhs))
    (|ncSoftError| (|pfSourcePosition| lhs)
     "%1 is improper for macro definition.  Ignored."
     (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|)))))))
```

———-

14.2.17 defun Add a macro to the global pfMacros list

[$pfMacros p352]

— **defun mac0Define 0** —

```
(defun |mac0Define| (sy state body)
 (declare (special |$pfMacros|))
  (setq |$pfMacros| (cons (list sy state body) |$pfMacros|)))
```

———-

14.2.18 defun macSubstituteOuter

[mac0SubstituteOuter p471]
[macLambdaParameterHandling p472]

— **defun macSubstituteOuter** —

```
(defun |macSubstituteOuter| (pform)
 (|mac0SubstituteOuter| (|macLambdaParameterHandling| nil pform) pform))
```

———-

14.2.19 defun mac0SubstituteOuter

[pfId? p487]
[macSubstituteId p473]
[pfLeaf? p488]
[pfLambda? p510]
[macLambdaParameterHandling p472]
[mac0SubstituteOuter p471]
[pfParts p489]

— **defun mac0SubstituteOuter** —

```
(defun |mac0SubstituteOuter| (replist pform)
 (let (tmplist)
  (cond
   ((|pfId?| pform) (|macSubstituteId| replist pform))
   ((|pfLeaf?| pform) pform)
   ((|pfLambda?| pform)
    (setq tmplist (|macLambdaParameterHandling| replist pform))
```

```
    (dolist (p (|pfParts| pform)) (|macOSubstituteOuter| tmplist p))
    pform)
   (t
    (dolist (p (|pfParts| pform)) (|macOSubstituteOuter| replist p))
    pform))))
```

———-

14.2.20 defun macLambdaParameterHandling

[pfLeaf? p488]
[pfLambda? p510]
[pfTypedId p525]
[pf0LambdaArgs p510]
[pfIdSymbol p487]
[pfMLambda? p514]
[pf0MLambdaArgs p514]
[pfLeaf p487]
[pfAbSynOp p624]
[pfLeafPosition p488]
[pfParts p489]
[macLambdaParameterHandling p472]

— **defun macLambdaParameterHandling** —

```
(defun |macLambdaParameterHandling| (replist pform)
 (let (parlist symlist result)
  (cond
   ((|pfLeaf?| pform) nil)
   ((|pfLambda?| pform) ; remove ( identifier . replacement ) from assoclist
    (setq parlist (mapcar #'|pfTypedId| (|pf0LambdaArgs| pform)))
    (setq symlist (mapcar #'|pfIdSymbol| parlist))
    (dolist (par symlist)
     (setq replist
      (let ((pr (assoc par replist :test #'equal)))
       (when pr (remove par replist :test #'equal)))))
    replist)
   ((|pfMLambda?| pform) ;construct assoclist ( identifier . replacement )
    (setq parlist (|pf0MLambdaArgs| pform)) ; extract parameter list
    (dolist (par parlist (nreverse result))
     (push
      (cons (|pfIdSymbol| par)
            (|pfLeaf| (|pfAbSynOp| par) (gensym) (|pfLeafPosition| par)))
       result)))
   (t
    (dolist (p (|pfParts| pform))
     (|macLambdaParameterHandling| replist p))))))
```

———-

14.2.21 defun macSubstituteId

[pfIdSymbol p487]

— **defun macSubstituteId** —

```
(defun |macSubstituteId| (replist pform)
 (let (ex)
  (setq ex (assoc (|pfIdSymbol| pform) replist :test #'eq))
  (cond
    (ex
     (rplaca pform (cadr ex))
     (rplacd pform (cddr ex))
     pform)
    (t pform))))
```

————-

Chapter 15

Pftrees

15.1 Abstract Syntax Trees Overview

Th functions create and examine abstract syntax trees. These are called pforms, for short.

The pform data structure

- Leaves: [hd, tok, pos] where pos is optional
- Trees: [hd, tree, tree, ...]
- hd is either an id or (id . alist)

The leaves are:

char	:=	('char expr position)
Document	:=	('Document expr position)
error	:=	('error expr position)
expression	:=	('expression expr position)
float	:=	('float expr position)
id	:=	('id expr position)
idsy	:=	('idsy expr position)
integer	:=	('integer expr position)
string	:=	('string expr position)
symbol	:=	('symbol expr position)

The special nodes:

ListOf	:=	('listOf items)
Nothing	:=	('nothing)
SemiColon	:=	('SemiColon (Body: Expr))

The expression nodes:

```
Add             :=  ('Add (Base: [Typed], Addin: Expr))
And             :=  ('And left right)
Application     :=  ('Application (Op: Expr, Arg: Expr))
Assign          :=  ('Assign (LhsItems: [AssLhs], Rhs: Expr))
Attribute       :=  ('Attribute (Expr: Primary))
Break           :=  ('Break (From: ? Id))
Coerceto        :=  ('Coerceto (Expr: Expr, Type: Type))
Collect         :=  ('Collect (Body: Expr, Iterators: [Iterator]))
ComDefinition   :=  ('ComDefinition (Doc: Document, Def: Definition))
DeclPart
Definition      :=  ('Definition (LhsItems: [Typed], Rhs: Expr))
DefinitionSequence :=  (Args: [DeclPart])
Do              :=  ('Do (Body: Expr))
Document        :=  ('Document strings)
DWhere          :=  ('DWhere (Context: [DeclPart], Expr: [DeclPart]))
EnSequence      :=
Exit            :=  ('Exit (Cond: ? Expr, Expr: ? Expr))
Export          :=  ('Export (Items: [Typed]))
Forin           :=  ('Forin (Lhs: [AssLhs], Whole: Expr))
Free            :=  ('Free (Items: [Typed]))
Fromdom         :=  ('Fromdom (What: Id, Domain: Type))
Hide            :=  ('hide, arg)
If              :=  ('If (Cond: Expr, Then: Expr, Else: ? Expr))
Import          :=  ('Import (Items: [QualType]))
Inline          :=  ('Inline (Items: [QualType]))
Iterate         :=  ('Iterate (From: ? Id))
Lambda          :=  ('Lambda (Args: [Typed], Rets: ReturnedTyped, Body: Expr))
Literal
Local           :=  ('Local (Items: [Typed]))
Loop            :=  ('Loop (Iterators: [Iterator]))
Macro           :=  ('Macro (Lhs: Id, Rhs: ExprorNot))
MLambda         :=  ('MLambda (Args: [Id], Body: Expr))
Not             :=  ('Not arg)
Novalue         :=  ('Novalue (Expr: Expr))
Or              :=  ('Or left right)
Pretend         :=  ('Pretend (Expr: Expr, Type: Type))
QualType        :=  ('QualType (Type: Type, Qual: ? Type))
Restrict        :=  ('Restrict (Expr: Expr, Type: Type))
Retract         :=  ('RetractTo (Expr: Expr, Type: Type))
Return          :=  ('Return (Expr: ? Expr, From: ? Id))
ReturnTyped     :=  ('returntyuped (type body))
Rule            :=  ('Rule (lhsitems, rhsitems))
Sequence        :=  ('Sequence (Args: [Expr]))
Suchthat        :=  ('Suchthat (Cond: Expr))
Symb            :=  if leaf then symbol else expression
Tagged          :=  ('Tagged (Tag: Expr, Expr: Expr))
TLambda         :=  ('TLambda (Args: [Typed],
                      Rets: ReturnedTyped Type, Body: Expr))
Tuple           :=  ('Tuple (Parts: [Expr]))
Typed           :=  ('Typed (Id: Id, Type: ? Type))
Typing          :=  ('Typing (Items: [Typed]))
Until           :=  ('Until (Cond: Expr)) NOT USED
WDeclare        :=  ('WDeclare (Signature: Typed, Doc: ? Document))
Where           :=  ('Where (Context: [DeclPart], Expr: Expr))
While           :=  ('While (Cond: Expr))
With            :=  ('With (Base: [Typed], Within: [WithPart]))
WIf             :=  ('WIf (Cond: Primary, Then: [WithPart], Else: [WithPart]))
Wrong           :=  ('Wrong (Why: Document, Rubble: [Expr]))
```

Special cases of expression nodes are:

- Application. The Op parameter is one of `and, or, Y, |, {}, [], {||}, [||]`
- DeclPart. The comment is attached to all signatutres in Typing, Import, Definition, Sequence, DWhere, Macro nodes
- EnSequence. This is either a Tuple or Sequence depending on the argument
- Literal. One of integer symbol expression one zero char string float of the form ('expression expr position)

15.2 Structure handlers

15.2.1 defun pfGlobalLinePosn

[poGlobalLinePosn p328]

— **defun pfGlobalLinePosn** —

```
(defun |pfGlobalLinePosn| (posn)
 (|poGlobalLinePosn| posn))
```

————-

15.2.2 defun pfCharPosn

[poCharPosn p594]

— **defun pfCharPosn** —

```
(defun |pfCharPosn| (posn)
 (|poCharPosn| posn))
```

————-

15.2.3 defun pfLinePosn

[poLinePosn p581]

— **defun pfLinePosn** —

```
(defun |pfLinePosn| (posn)
 (|poLinePosn| posn))
```

————-

15.2.4 defun pfFileName

[poFileName p580]

— **defun pfFileName** —

```
(defun |pfFileName| (posn)
 (|poFileName| posn))
```

———-

15.2.5 defun pfCopyWithPos

[pfLeaf? p488]
[pfLeaf p487]
[pfAbSynOp p624]
[tokPart p625]
[pfTree p492]
[pfParts p489]
[pfCopyWithPos p478]

— **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)))))
```

———-

15.2.6 defun pfMapParts

[pfLeaf? p488]
[pfParts p489]
[pfTree p492]
[pfAbSynOp p624]

— **defun pfMapParts** —

```
(defun |pfMapParts| (f pform)
 (let (parts1 parts0)
  (if (|pfLeaf?| pform)
   pform
   (progn
    (setq parts0 (|pfParts| pform))
    (setq parts1 (loop for p in parts0 collect (funcall f p)))
    (if (reduce #'(lambda (u v) (and u v)) (mapcar #'eq parts0 parts1))
     pform
     (|pfTree| (|pfAbSynOp| pform) parts1))))))
```

———-

15.2.7 defun pf0ApplicationArgs

[pf0FlattenSyntacticTuple p479]
[pfApplicationArg p493]

— **defun pf0ApplicationArgs** —

```
(defun |pf0ApplicationArgs| (pform)
  (|pf0FlattenSyntacticTuple| (|pfApplicationArg| pform)))
```

————-

15.2.8 defun pf0FlattenSyntacticTuple

[pfTuple? p526]
[pf0FlattenSyntacticTuple p479]
[pf0TupleParts p527]

— **defun pf0FlattenSyntacticTuple** —

```
(defun |pf0FlattenSyntacticTuple| (pform)
 (if (null (|pfTuple?| pform))
  (list pform)
   ; [:pf0FlattenSyntacticTuple p for p in pf0TupleParts pform]
  ((lambda (arg0 arg1 p)
   (loop
    (cond
     ((or (atom arg1) (progn (setq p (car arg1)) nil))
      (return (nreverse arg0)))
     (t
      (setq arg0 (append (reverse (|pf0FlattenSyntacticTuple| p)) arg0))))
    (setq arg1 (cdr arg1))))
   nil (|pf0TupleParts| pform) nil)))
```

————-

15.2.9 defun pfSourcePosition

[pfLeaf? p488]
[pfLeafPosition p488]
[poNoPosition? p624]
[pfSourcePosition p479]
[pfParts p489]
[$nopos p288]

— **defun pfSourcePosition** —

```
(defun |pfSourcePosition| (form)
 (let (pos)
 (declare (special |$nopos|))
  (cond
   ((|pfLeaf?| form) (|pfLeafPosition| form))
```

```
   (t
    (setq pos |$nopos|)
    ((lambda (theparts p) ; for p in parts while poNoPosition? pos repeat
     (loop
      (cond
       ((or (atom theparts)
            (progn (setq p (car theparts)) nil)
            (not (|poNoPosition?| pos)))
          (return nil))
       (t (setq pos (|pfSourcePosition| p))))
      (setq theparts (cdr theparts))))
     (|pfParts| form) nil)
    pos))))
```

———-

15.2.10 defun Convert a Sequence node to a list

[pfSequence? p522]
[pfSequenceArgs p522]
[pfListOf p485]

— **defun pfSequenceToList** —

```
(defun |pfSequenceToList| (x)
 (if (|pfSequence?| x)
  (|pfSequenceArgs| x)
  (|pfListOf| (list x))))
```

———-

15.2.11 defun pfSpread

[pfTyped p525]

— **defun pfSpread** —

```
(defun |pfSpread| (arg1 arg2)
 (mapcar #'(lambda (i) (|pfTyped| i arg2)) arg1))
```

———-

15.2.12 defun Deconstruct nodes to lists

[pfTagged? p523]
[pfTaggedExpr p523]
[pfNothing p486]
[pfTaggedTag p524]
[pfId? p487]
[pfListOf p485]

[pfTyped p525]
[pfCollect1? p483]
[pfCollectVariable1 p483]
[pfTuple? p526]
[pf0TupleParts p527]
[pfTaggedToTyped p524]
[pfDefinition? p500]
[pfApplication? p494]
[pfFlattenApp p483]
[pfTaggedToTyped1 p485]
[pfTransformArg p484]
[npTrapForm p452]

— **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)))))
```

———-

15.2.13 defun pfCheckMacroOut

[pfId? p487]
[pfApplication? p494]
[pfFlattenApp p483]
[pfCheckId p482]

[pfCheckArg p482]
[npTrapForm p452]

— **defun pfCheckMacroOut** —

```
(defun |pfCheckMacroOut| (form)
 (let (args op ls)
  (cond
   ((|pfId?| form) (list form nil))
   ((|pfApplication?| form)
    (setq ls (|pfFlattenApp| form))
    (setq op (|pfCheckId| (car ls)))
    (setq args (mapcar #'|pfCheckArg| (cdr ls)))
    (list op args))
   (t (|npTrapForm| form)))))
```

————-

15.2.14 defun pfCheckArg

[pfTuple? p526]
[pf0TupleParts p527]
[pfListOf p485]
[pfCheckId p482]

— **defun pfCheckArg** —

```
(defun |pfCheckArg| (args)
 (let (argl)
  (if (|pfTuple?| args)
   (setq argl (|pf0TupleParts| args))
   (setq argl (list args)))
  (|pfListOf| (mapcar #'|pfCheckId| argl))))
```

————-

15.2.15 defun pfCheckId

[pfId? p487]
[npTrapForm p452]

— **defun pfCheckId** —

```
(defun |pfCheckId| (form)
 (if (null (|pfId?| form))
  (|npTrapForm| form)
  form))
```

————-

15.2.16 defun pfFlattenApp

[pfApplication? p494]
[pfCollect1? p483]
[pfFlattenApp p483]
[pfApplicationOp p493]
[pfApplicationArg p493]

— **defun pfFlattenApp** —

```
(defun |pfFlattenApp| (x)
 (cond
  ((|pfApplication?| x)
   (cond
    ((|pfCollect1?| x) (LIST x))
    (t
     (append (|pfFlattenApp| (|pfApplicationOp| x))
             (|pfFlattenApp| (|pfApplicationArg| x))))))
  (t (list x))))
```

————-

15.2.17 defun pfCollect1?

[pfApplication? p494]
[pfApplicationOp p493]
[pfId? p487]
[pfIdSymbol p487]

— **defun pfCollect1?** —

```
(defun |pfCollect1?| (x)
 (let (a)
  (when (|pfApplication?| x)
   (setq a (|pfApplicationOp| x))
   (when (|pfId?| a) (eq (|pfIdSymbol| a) '|\||)))))
```

————-

15.2.18 defun pfCollectVariable1

[pfApplicationArg p493]
[pf0TupleParts p527]
[pfTaggedToTyped p524]
[pfTyped p525]
[pfSuch p485]
[pfTypedId p525]
[pfTypedType p525]

— **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))))
```

———-

15.2.19 defun pfPushMacroBody

[pfMLambda p514]
[pfPushMacroBody p484]

— **defun pfPushMacroBody** —

```
(defun |pfPushMacroBody| (args body)
 (if (null args)
   body
   (|pfMLambda| (car args) (|pfPushMacroBody| (cdr args) body))))
```

———-

15.2.20 defun pfSourceStok

[pfLeaf? p488]
[pfParts p489]
[pfSourceStok p484]
[pfFirst p502]

— **defun pfSourceStok** —

```
(defun |pfSourceStok| (x)
 (cond
  ((|pfLeaf?| x) x)
  ((null (|pfParts| x)) '|NoToken|)
  (t (|pfSourceStok| (|pfFirst| x)))))
```

———-

15.2.21 defun pfTransformArg

[pfTuple? p526]
[pf0TupleParts p527]
[pfListOf p485]
[pfTaggedToTyped1 p485]

— **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)))))
```

———-

15.2.22 defun pfTaggedToTyped1

[pfCollect1? p483]
[pfCollectVariable1 p483]
[pfDefinition? p500]
[pfTyped p525]
[pfNothing p486]
[pfTaggedToTyped p524]

— **defun pfTaggedToTyped1** —

```
(defun |pfTaggedToTyped1| (arg)
 (cond
  ((|pfCollect1?| arg) (|pfCollectVariable1| arg))
  ((|pfDefinition?| arg) (|pfTyped| arg (|pfNothing|)))
  (t (|pfTaggedToTyped| arg))))
```

———-

15.2.23 defun pfSuch

[pfInfApplication p508]
[pfId p486]

— **defun pfSuch** —

```
(defun |pfSuch| (x y)
 (|pfInfApplication| (|pfId| '|\||) x y))
```

———-

15.3 Special Nodes

15.3.1 defun Create a Listof node

[pfTree p492]

— **defun pfListOf** —

```
(defun |pfListOf| (x)
 (|pfTree| '|listOf| x))
```

———-

15.3.2 defun pfNothing

[pfTree p492]

— **defun pfNothing** —

```
(defun |pfNothing| ()
 (|pfTree| '|nothing| nil))
```

———-

15.3.3 defun Is this a Nothing node?

[pfAbSynOp? p624]

— **defun pfNothing?** —

```
(defun |pfNothing?| (form)
 (|pfAbSynOp?| form '|nothing|))
```

———-

15.4 Leaves

15.4.1 defun Create a Document node

[pfLeaf p487]

— **defun pfDocument** —

```
(defun |pfDocument| (strings)
 (|pfLeaf| '|Document| strings))
```

———-

15.4.2 defun Construct an Id node

[pfLeaf p487]

— **defun pfId** —

```
(defun |pfId| (expr)
 (|pfLeaf| '|id| expr))
```

———-

15.4.3 defun Is this an Id node?

[pfAbSynOp? p624]

— **defun pfId?** —

```
(defun |pfId?| (form)
 (or (|pfAbSynOp?| form '|id|) (|pfAbSynOp?| form '|idsy|)))
```

———-

15.4.4 defun Construct an Id leaf node

[pfLeaf p487]

— **defun pfIdPos** —

```
(defun |pfIdPos| (expr pos)
 (|pfLeaf| '|id| expr pos))
```

———-

15.4.5 defun Return the Id part

[tokPart p625]

— **defun pfIdSymbol** —

```
(defun |pfIdSymbol| (form)
 (|tokPart| form))
```

———-

15.4.6 defun Construct a Leaf node

[tokConstruct p623]
[ifcar p**??**]
[pfNoPosition p625]

— **defun pfLeaf** —

```
(defun |pfLeaf| (x y &rest z)
 (|tokConstruct| x y (or (ifcar z) (|pfNoPosition|))))
```

———-

15.4.7 defun Is this a leaf node?

[pfAbSynOp p624]

— **defun pfLeaf?** —

```
(defun |pfLeaf?| (form)
 (member (|pfAbSynOp| form)
   '(|id| |idsy| |symbol| |string| |char| |float| |expression|
     |integer| |Document| |error|)))
```

————-

15.4.8 defun Return the token position of a leaf node

[tokPosn p625]

— **defun pfLeafPosition** —

```
(defun |pfLeafPosition| (form)
 (|tokPosn| form))
```

————-

15.4.9 defun Return the Leaf Token

[tokPart p625]

— **defun pfLeafToken** —

```
(defun |pfLeafToken| (form)
 (|tokPart| form))
```

————-

15.4.10 defun Is this a Literal node?

[pfAbSynOp p624]

— **defun pfLiteral? 0** —

```
(defun |pfLiteral?| (form)
 (member (|pfAbSynOp| form)
  '(|integer| |symbol| |expression| |one| |zero| |char| |string| |float|)))
```

————-

15.4.11 defun Create a LiteralClass node

[pfAbSynOp p624]

— **defun pfLiteralClass** —

```
(defun |pfLiteralClass| (form)
 (|pfAbSynOp| form))
```

————-

15.4.12 defun Return the LiteralString

[tokPart p625]

— **defun pfLiteralString** —

```
(defun |pfLiteralString| (form)
 (|tokPart| form))
```

————-

15.4.13 defun Return the parts of a tree node

— **defun pfParts 0** —

```
(defun |pfParts| (form)
 (cdr form))
```

————-

15.4.14 defun Return the argument unchanged

— **defun pfPile 0** —

```
(defun |pfPile| (part)
 part)
```

————-

15.4.15 defun pfPushBody

[pfLambda p509]
[pfNothing p486]
[pfPushBody p489]

— **defun pfPushBody** —

```
(defun |pfPushBody| (rt args body)
 (cond
  ((null args) body)
  ((null (cdr args)) (|pfLambda| (car args) rt body))
  (t
   (|pfLambda| (car args) (|pfNothing|)
               (|pfPushBody| rt (cdr args) body)))))
```

———-

15.4.16 defun An S-expression which people can read.

[pfSexpr,strip p490]

— **defun pfSexpr** —

```
(defun |pfSexpr| (pform)
 (|pfSexpr,strip| pform))
```

———-

15.4.17 defun Create a human readable S-expression

[pfId? p487]
[pfIdSymbol p487]
[pfLiteral? p488]
[pfLiteralString p489]
[pfLeaf? p488]
[tokPart p625]
[pfApplication? p494]
[pfApplicationArg p493]
[pfTuple? p526]
[pf0TupleParts p527]
[pfApplicationOp p493]
[pfSexpr,strip p490]
[pfAbSynOp p624]
[pfParts p489]

— **defun pfSexpr,strip** —

```
(defun |pfSexpr,strip| (pform)
 (let (args a result)
  (cond
   ((|pfId?| pform)      (|pfIdSymbol| pform))
   ((|pfLiteral?| pform) (|pfLiteralString| pform))
   ((|pfLeaf?| pform)    (|tokPart| pform))
   ((|pfApplication?| pform)
    (setq a (|pfApplicationArg| pform))
    (if (|pfTuple?| a)
     (setq args (|pf0TupleParts| a))
     (setq args (list a)))
```

```
    (dolist (p (cons (|pfApplicationOp| pform) args) (nreverse result))
     (push (|pfSexpr,strip| p) result)))
   (t
    (cons (|pfAbSynOp| pform)
     (dolist (p (|pfParts| pform) (nreverse result))
      (push (|pfSexpr,strip| p) result)))))))
```

———-

15.4.18 defun Construct a Symbol or Expression node

[pfLeaf? p488]
[pfSymbol p491]
[tokPart p625]
[ifcar p??]
[pfExpression p502]
[pfSexpr p490]

— **defun pfSymb** —

```
(defun |pfSymb| (expr &REST optpos)
 (if (|pfLeaf?| expr)
  (|pfSymbol| (|tokPart| expr) (ifcar optpos))
  (|pfExpression| (|pfSexpr| expr) (ifcar optpos))))
```

———-

15.4.19 defun Construct a Symbol leaf node

[pfLeaf p487]
[ifcar p??]

— **defun pfSymbol** —

```
(defun |pfSymbol| (expr &rest optpos)
 (|pfLeaf| '|symbol| expr (ifcar optpos)))
```

———-

15.4.20 defun Is this a Symbol node?

[pfAbSynOp? p624]

— **defun pfSymbol?** —

```
(defun |pfSymbol?| (form)
 (|pfAbSynOp?| form '|symbol|))
```

———-

15.4.21 defun Return the Symbol part

[tokPart p625]

— **defun pfSymbolSymbol** —

```
(defun |pfSymbolSymbol| (form)
 (|tokPart| form))
```

————-

15.5 Trees

15.5.1 defun Construct a tree node

— **defun pfTree 0** —

```
(defun |pfTree| (x y)
 (cons x y))
```

————-

15.5.2 defun Construct an Add node

[pfNothing p486]
[pfTree p492]

— **defun pfAdd** —

```
(defun |pfAdd| (pfbase pfaddin &rest addon)
 (let (lhs)
  (if addon
   (setq lhs addon)
   (setq lhs (|pfNothing|)))
  (|pfTree| '|Add| (list pfbase pfaddin lhs))))
```

————-

15.5.3 defun Construct an And node

[pfTree p492]

— **defun pfAnd** —

```
(defun |pfAnd| (pfleft pfright)
 (|pfTree| '|And| (list pfleft pfright)))
```

————-

15.5.4 defun pfAttribute

[pfTree p492]

— **defun pfAttribute** —

```
(defun |pfAttribute| (pfexpr)
 (|pfTree| '|Attribute| (list pfexpr)))
```

———-

15.5.5 defun Return an Application node

[pfTree p492]

— **defun pfApplication** —

```
(defun |pfApplication| (pfop pfarg)
 (|pfTree| '|Application| (list pfop pfarg)))
```

———-

15.5.6 defun Return the Arg part of an Application node

— **defun pfApplicationArg 0** —

```
(defun |pfApplicationArg| (pf)
 (caddr pf))
```

———-

15.5.7 defun Return the Op part of an Application node

— **defun pfApplicationOp 0** —

```
(defun |pfApplicationOp| (pf)
 (cadr pf))
```

———-

15.5.8 defun Is this an And node?

[pfAbSynOp? p624]

— **defun pfAnd?** —

```
(defun |pfAnd?| (pf)
 (|pfAbSynOp?| pf '|And|))
```

———-

15.5.9 defun Return the Left part of an And node

— **defun pfAndLeft 0** —

```
(defun |pfAndLeft| (pf)
 (cadr pf))
```

———-

15.5.10 defun Return the Right part of an And node

— **defun pfAndRight 0** —

```
(defun |pfAndRight| (pf)
 (caddr pf))
```

———-

15.5.11 defun Flatten a list of lists

— **defun pfAppend 0** —

```
(defun |pfAppend| (list)
 (apply #'append list))
```

———-

15.5.12 defun Is this an Application node?

[pfAbSynOp? p624]

— **defun pfApplication?** —

```
(defun |pfApplication?| (pf)
  (|pfAbSynOp?| pf '|Application|))
```

———-

15.5.13 defun Create an Assign node

[pfTree p492]

— **defun pfAssign** —

```
(defun |pfAssign| (pflhsitems pfrhs)
 (|pfTree| '|Assign| (list pflhsitems pfrhs)))
```

———-

15.5.14 defun Is this an Assign node?

[pfAbSynOp? p624]

— **defun pfAssign?** —

```
(defun |pfAssign?| (pf)
 (|pfAbSynOp?| pf '|Assign|))
```

———-

15.5.15 defun Return the parts of an LhsItem of an Assign node

[pfParts p489]
[pfAssignLhsItems p495]

— **defun pf0AssignLhsItems 0** —

```
(defun |pf0AssignLhsItems| (pf)
 (|pfParts| (|pfAssignLhsItems| pf)))
```

———-

15.5.16 defun Return the LhsItem of an Assign node

— **defun pfAssignLhsItems 0** —

```
(defun |pfAssignLhsItems| (pf)
 (cadr pf))
```

———-

15.5.17 defun Return the RHS of an Assign node

— **defun pfAssignRhs 0** —

```
(defun |pfAssignRhs| (pf)
 (caddr pf))
```

———-

15.5.18 defun Construct an application node for a brace

[pfApplication p493]
[pfIdPos p487]
[tokPosn p625]

— **defun pfBrace** —

```
(defun |pfBrace| (a part)
 (|pfApplication| (|pfIdPos| '{} (|tokPosn| a)) part))
```

———-

15.5.19 defun Construct an Application node for brace-bars

[pfApplication p493]
[pfIdPos p487]
[tokPosn p625]

— **defun pfBraceBar** —

```
(defun |pfBraceBar| (a part)
 (|pfApplication| (|pfIdPos| '|{\|\|}| (|tokPosn| a)) part))
```

———-

15.5.20 defun Construct an Application node for a bracket

[pfApplication p493]
[pfIdPos p487]
[tokPosn p625]

— **defun pfBracket** —

```
(defun |pfBracket| (a part)
 (|pfApplication| (|pfIdPos| '[] (|tokPosn| a)) part))
```

———-

15.5.21 defun Construct an Application node for bracket-bars

[pfApplication p493]
[pfIdPos p487]
[tokPosn p625]

— **defun pfBracketBar** —

```
(defun |pfBracketBar| (a part)
 (|pfApplication| (|pfIdPos| '|[\|\|]| (|tokPosn| a)) part))
```

———-

15.5.22 defun Create a Break node

[pfTree p492]

— defun pfBreak —

```
(defun |pfBreak| (pffrom)
 (|pfTree| '|Break| (list pffrom)))
```

———-

15.5.23 defun Is this a Break node?

[pfAbSynOp? p624]

— defun pfBreak? —

```
(defun |pfBreak?| (pf)
 (|pfAbSynOp?| pf '|Break|))
```

———-

15.5.24 defun Return the From part of a Break node

— defun pfBreakFrom 0 —

```
(defun |pfBreakFrom| (pf)
 (cadr pf))
```

———-

15.5.25 defun Construct a Coerceto node

[pfTree p492]

— defun pfCoerceto —

```
(defun |pfCoerceto| (pfexpr pftype)
 (|pfTree| '|Coerceto| (list pfexpr pftype)))
```

———-

15.5.26 defun Is this a CoerceTo node?

[pfAbSynOp? p624]

— **defun pfCoerceto?** —

```
(defun |pfCoerceto?| (pf)
 (|pfAbSynOp?| pf '|Coerceto|))
```

———-

15.5.27 defun Return the Expression part of a CoerceTo node

— **defun pfCoercetoExpr 0** —

```
(defun |pfCoercetoExpr| (pf)
 (cadr pf))
```

———-

15.5.28 defun Return the Type part of a CoerceTo node

— **defun pfCoercetoType 0** —

```
(defun |pfCoercetoType| (pf)
 (caddr pf))
```

———-

15.5.29 defun Return the Body of a Collect node

— **defun pfCollectBody 0** —

```
(defun |pfCollectBody| (pf)
 (cadr pf))
```

———-

15.5.30 defun Return the Iterators of a Collect node

— **defun pfCollectIterators 0** —

```
(defun |pfCollectIterators| (pf)
 (caddr pf))
```

———-

15.5.31 defun Create a Collect node

[pfTree p492]

— **defun pfCollect** —

```
(defun |pfCollect| (pfbody pfiterators)
 (|pfTree| '|Collect| (list pfbody pfiterators)))
```

———-

15.5.32 defun Is this a Collect node?

[pfAbSynOp? p624]

— **defun pfCollect?** —

```
(defun |pfCollect?| (pf)
 (|pfAbSynOp?| pf '|Collect|))
```

———-

15.5.33 defun pfDefinition

[pfTree p492]

— **defun pfDefinition** —

```
(defun |pfDefinition| (pflhsitems pfrhs)
 (|pfTree| '|Definition| (list pflhsitems pfrhs)))
```

———-

15.5.34 defun Return the Lhs of a Definition node

— **defun pfDefinitionLhsItems 0** —

```
(defun |pfDefinitionLhsItems| (pf)
 (cadr pf))
```

———-

15.5.35 defun Return the Rhs of a Definition node

— **defun pfDefinitionRhs 0** —

```
(defun |pfDefinitionRhs| (pf)
 (caddr pf))
```

———-

15.5.36 defun Is this a Definition node?

[pfAbSynOp? p624]

— **defun pfDefinition?** —

```
(defun |pfDefinition?| (pf)
 (|pfAbSynOp?| pf '|Definition|))
```

———-

15.5.37 defun Return the parts of a Definition node

[pfParts p489]
[pfDefinitionLhsItems p499]

— **defun pf0DefinitionLhsItems** —

```
(defun |pf0DefinitionLhsItems| (pf)
 (|pfParts| (|pfDefinitionLhsItems| pf)))
```

———-

15.5.38 defun Create a Do node

[pfTree p492]

— **defun pfDo** —

```
(defun |pfDo| (pfbody)
 (|pfTree| '|Do| (list pfbody)))
```

———-

15.5.39 defun Is this a Do node?

[pfAbSynOp? p624]

— **defun pfDo?** —

```
(defun |pfDo?| (pf)
 (|pfAbSynOp?| pf '|Do|))
```

———-

15.5.40 defun Return the Body of a Do node

— **defun pfDoBody 0** —

```
(defun |pfDoBody| (pf)
 (cadr pf))
```

————-

15.5.41 defun Construct a Sequence node

[pfTuple p526]
[pfListOf p485]
[pfSequence p521]

— **defun pfEnSequence** —

```
(defun |pfEnSequence| (a)
 (cond
  ((null a) (|pfTuple| (|pfListOf| a)))
  ((null (cdr a)) (car a))
  (t (|pfSequence| (|pfListOf| a)))))
```

————-

15.5.42 defun Construct an Exit node

[pfTree p492]

— **defun pfExit** —

```
(defun |pfExit| (pfcond pfexpr)
 (|pfTree| '|Exit| (list pfcond pfexpr)))
```

————-

15.5.43 defun Is this an Exit node?

[pfAbSynOp? p624]

— **defun pfExit?** —

```
(defun |pfExit?| (pf)
 (|pfAbSynOp?| pf '|Exit|))
```

————-

15.5.44 defun Return the Cond part of an Exit

— **defun pfExitCond 0** —

```
(defun |pfExitCond| (pf)
 (cadr pf))
```

————-

15.5.45 defun Return the Expression part of an Exit

— **defun pfExitExpr 0** —

```
(defun |pfExitExpr| (pf)
 (caddr pf))
```

————-

15.5.46 defun Create an Export node

[pfTree p492]

— **defun pfExport** —

```
(defun |pfExport| (pfitems)
 (|pfTree| '|Export| (list pfitems)))
```

————-

15.5.47 defun Construct an Expression leaf node

[pfLeaf p487]
[ifcar p**??**]

— **defun pfExpression** —

```
(defun |pfExpression| (expr &rest optpos)
 (|pfLeaf| '|expression| expr (ifcar optpos)))
```

————-

15.5.48 defun pfFirst

— **defun pfFirst 0** —

```
(defun |pfFirst| (form)
 (cadr form))
```

———-

15.5.49 defun Create an Application Fix node

[pfApplication p493]
[pfId p486]

— **defun pfFix** —

```
(defun |pfFix| (pf)
 (|pfApplication| (|pfId| 'Y) pf))
```

———-

15.5.50 defun Create a Free node

[pfTree p492]

— **defun pfFree** —

```
(defun |pfFree| (pfitems)
 (|pfTree| '|Free| (list pfitems)))
```

———-

15.5.51 defun Is this a Free node?

[pfAbSynOp? p624]

— **defun pfFree?** —

```
(defun |pfFree?| (pf)
 (|pfAbSynOp?| pf '|Free|))
```

———-

15.5.52 defun Return the parts of the Items of a Free node

[pfParts p489]
[pfFreeItems p504]

— **defun pf0FreeItems** —

```
(defun |pf0FreeItems| (pf)
 (|pfParts| (|pfFreeItems| pf)))
```

———-

15.5.53 defun Return the Items of a Free node

— **defun pfFreeItems 0** —

```
(defun |pfFreeItems| (pf)
 (cadr pf))
```

————-

15.5.54 defun Construct a Forin node

[pfTree p492]

— **defun pfForin** —

```
(defun |pfForin| (pflhs pfwhole)
 (|pfTree| '|Forin| (list pflhs pfwhole)))
```

————-

15.5.55 defun Is this a ForIn node?

[pfAbSynOp? p624]

— **defun pfForin?** —

```
(defun |pfForin?| (pf)
 (|pfAbSynOp?| pf '|Forin|))
```

————-

15.5.56 defun Return all the parts of the LHS of a ForIn node

[pfParts p489]
[pfForinLhs p504]

— **defun pf0ForinLhs** —

```
(defun |pf0ForinLhs| (pf)
 (|pfParts| (|pfForinLhs| pf)))
```

————-

15.5.57 defun Return the LHS part of a ForIn node

— **defun pfForinLhs 0** —

```
(defun |pfForinLhs| (pf)
```

```
 (cadr pf))
```

———-

15.5.58 defun Return the Whole part of a ForIn node

— **defun pfForinWhole 0** —

```
(defun |pfForinWhole| (pf)
 (caddr pf))
```

———-

15.5.59 defun pfFromDom

[pfApplication? p494]
[pfApplication p493]
[pfApplicationOp p493]
[pfApplicationArg p493]
[pfFromdom p505]

— **defun pfFromDom** —

```
(defun |pfFromDom| (dom expr)
 (cond
  ((|pfApplication?| expr)
   (|pfApplication|
    (|pfFromdom| (|pfApplicationOp| expr) dom)
    (|pfApplicationArg| expr)))
  (t (|pfFromdom| expr dom))))
```

———-

15.5.60 defun Construct a Fromdom node

[pfTree p492]

— **defun pfFromdom** —

```
(defun |pfFromdom| (pfwhat pfdomain)
 (|pfTree| '|Fromdom| (list pfwhat pfdomain)))
```

———-

15.5.61 defun Is this a Fromdom mode?

[pfAbSynOp? p624]

— **defun pfFromdom?** —

```
(defun |pfFromdom?| (pf)
 (|pfAbSynOp?| pf '|Fromdom|))
```

———-

15.5.62 defun Return the What part of a Fromdom node

— **defun pfFromdomWhat 0** —

```
(defun |pfFromdomWhat| (pf)
 (cadr pf))
```

———-

15.5.63 defun Return the Domain part of a Fromdom node

— **defun pfFromdomDomain 0** —

```
(defun |pfFromdomDomain| (pf)
 (caddr pf))
```

———-

15.5.64 defun Construct a Hide node

[pfTree p492]

— **defun pfHide** —

```
(defun |pfHide| (a part)
 (declare (ignore a))
 (|pfTree| '|Hide| (list part)))
```

———-

15.5.65 defun pfIf

[pfTree p492]

— **defun pfIf** —

```
(defun |pfIf| (pfcond pfthen pfelse)
 (|pfTree| '|If| (list pfcond pfthen pfelse)))
```

———-

15.5.66 defun Is this an If node?

[pfAbSynOp? p624]

— **defun pfIf?** —

```
(defun |pfIf?| (pf)
  (|pfAbSynOp?| pf '|If|))
```

————-

15.5.67 defun Return the Cond part of an If

— **defun pfIfCond 0** —

```
(defun |pfIfCond| (pf)
 (cadr pf))
```

————-

15.5.68 defun Return the Then part of an If

— **defun pfIfThen 0** —

```
(defun |pfIfThen| (pf)
 (caddr pf))
```

————-

15.5.69 defun pfIfThenOnly

[pfIf p506]
[pfNothing p486]

— **defun pfIfThenOnly** —

```
(defun |pfIfThenOnly| (pred cararg)
 (|pfIf| pred cararg (|pfNothing|)))
```

————-

15.5.70 defun Return the Else part of an If

— **defun pfIfElse 0** —

```
(defun |pfIfElse| (pf)
 (cadddr pf))
```

———-

15.5.71 defun Construct an Import node

[pfTree p492]

— **defun pfImport** —

```
(defun |pfImport| (pfitems)
 (|pfTree| '|Import| (list pfitems)))
```

———-

15.5.72 defun Construct an Iterate node

[pfTree p492]

— **defun pfIterate** —

```
(defun |pfIterate| (pffrom)
 (|pfTree| '|Iterate| (list pffrom)))
```

———-

15.5.73 defun Is this an Iterate node?

[pfAbSynOp? p624]

— **defun pfIterate?** —

```
(defun |pfIterate?| (pf)
 (|pfAbSynOp?| pf '|Iterate|))
```

———-

15.5.74 defun Handle an infix application

[pfListOf p485]
[pfIdSymbol p487]
[pfAnd p492]
[pfOr p516]
[pfApplication p493]
[pfTuple p526]

— **defun pfInfApplication** —

```
(defun |pfInfApplication| (op left right)
 (cond
```

```
  ((eq (|pfIdSymbol| op) '|and|) (|pfAnd| left right))
  ((eq (|pfIdSymbol| op) '|or|) (|pfOr| left right))
  (t (|pfApplication| op (|pfTuple| (|pfListOf| (list left right)))))))
```

———-

15.5.75 defun Create an Inline node

[pfTree p492]

— **defun pfInline** —

```
(defun |pfInline| (pfitems)
 (|pfTree| '|Inline| (list pfitems)))
```

———-

15.5.76 defun pfLam

[pfAbSynOp? p624]
[pfFirst p502]
[pfNothing p486]
[pfSecond p521]
[pfLambda p509]

— **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)))
```

———-

15.5.77 defun pfLambda

[pfTree p492]

— **defun pfLambda** —

```
(defun |pfLambda| (pfargs pfrets pfbody)
 (|pfTree| '|Lambda| (list pfargs pfrets pfbody)))
```

———-

15.5.78 defun Return the Body part of a Lambda node

— defun pfLambdaBody 0 —

```
(defun |pfLambdaBody| (pf)
 (cadddr pf))
```

————-

15.5.79 defun Return the Rets part of a Lambda node

— defun pfLambdaRets 0 —

```
(defun |pfLambdaRets| (pf)
 (caddr pf))
```

————-

15.5.80 defun Is this a Lambda node?

[pfAbSynOp? p624]

— defun pfLambda? —

```
(defun |pfLambda?| (pf)
 (|pfAbSynOp?| pf '|Lambda|))
```

————-

15.5.81 defun Return the Args part of a Lambda node

— defun pfLambdaArgs 0 —

```
(defun |pfLambdaArgs| (pf)
 (cadr pf))
```

————-

15.5.82 defun Return the Args of a Lambda Node

[pfParts p489]
[pfLambdaArgs p510]

— defun pf0LambdaArgs —

```
(defun |pf0LambdaArgs| (pf)
 (|pfParts| (|pfLambdaArgs| pf)))
```

———-

15.5.83 defun Construct a Local node

[pfTree p492]

— **defun pfLocal** —

```
(defun |pfLocal| (pfitems)
 (|pfTree| '|Local| (list pfitems)))
```

———-

15.5.84 defun Is this a Local node?

[pfAbSynOp? p624]

— **defun pfLocal?** —

```
(defun |pfLocal?| (pf)
 (|pfAbSynOp?| pf '|Local|))
```

———-

15.5.85 defun Return the parts of Items of a Local node

[pfParts p489]
[pfLocalItems p511]

— **defun pf0LocalItems** —

```
(defun |pf0LocalItems| (pf)
 (|pfParts| (|pfLocalItems| pf)))
```

———-

15.5.86 defun Return the Items of a Local node

— **defun pfLocalItems 0** —

```
(defun |pfLocalItems| (pf)
 (cadr pf))
```

———-

15.5.87 defun Construct a Loop node

[pfTree p492]

— **defun pfLoop** —

```
(defun |pfLoop| (pfiterators)
 (|pfTree| '|Loop| (list pfiterators)))
```

———-

15.5.88 defun pfLoop1

[pfLoop p512]
[pfListOf p485]
[pfDo p500]

— **defun pfLoop1** —

```
(defun |pfLoop1| (body)
 (|pfLoop| (|pfListOf| (list (|pfDo| body)))))
```

———-

15.5.89 defun Is this a Loop node?

[pfAbSynOp? p624]

— **defun pfLoop?** —

```
(defun |pfLoop?| (pf)
 (|pfAbSynOp?| pf '|Loop|))
```

———-

15.5.90 defun Return the Iterators of a Loop node

— **defun pfLoopIterators 0** —

```
(defun |pfLoopIterators| (pf)
 (cadr pf))
```

———-

15.5.91 defun pf0LoopIterators

[pfParts p489]
[pf0LoopIterators p512]

— **defun pf0LoopIterators** —

```
(defun |pf0LoopIterators| (pf)
 (|pfParts| (|pfLoopIterators| pf)))
```

————-

15.5.92 defun pfLp

[pfLoop p512]
[pfListOf p485]
[pfDo p500]

— **defun pfLp** —

```
(defun |pfLp| (iterators body)
 (|pfLoop| (|pfListOf| (append iterators (list (|pfDo| body))))))
```

————-

15.5.93 defun Create a Macro node

[pfTree p492]

— **defun pfMacro** —

```
(defun |pfMacro| (pflhs pfrhs)
 (|pfTree| '|Macro| (list pflhs pfrhs)))
```

————-

15.5.94 defun Is this a Macro node?

[pfAbSynOp? p624]

— **defun pfMacro?** —

```
(defun |pfMacro?| (pf)
 (|pfAbSynOp?| pf '|Macro|))
```

————-

15.5.95 defun Return the Lhs of a Macro node

— **defun pfMacroLhs 0** —

```
(defun |pfMacroLhs| (pf)
 (cadr pf))
```

———-

15.5.96 defun Return the Rhs of a Macro node

— **defun pfMacroRhs 0** —

```
(defun |pfMacroRhs| (pf)
 (caddr pf))
```

———-

15.5.97 defun Construct an MLambda node

[pfTree p492]

— **defun pfMLambda** —

```
(defun |pfMLambda| (pfargs pfbody)
 (|pfTree| '|MLambda| (list pfargs pfbody)))
```

———-

15.5.98 defun Is this an MLambda node?

[pfAbSynOp? p624]

— **defun pfMLambda?** —

```
(defun |pfMLambda?| (pf)
 (|pfAbSynOp?| pf '|MLambda|))
```

———-

15.5.99 defun Return the Args of an MLambda

— **defun pfMLambdaArgs 0** —

```
(defun |pfMLambdaArgs| (pf)
 (cadr pf))
```

———-

15.5.100 defun Return the parts of an MLambda argument

[pfParts p489]

— **defun pf0MLambdaArgs** —

```
(defun |pf0MLambdaArgs| (pf)
 (|pfParts| (|pfMLambdaArgs| pf)))
```

————-

15.5.101 defun pfMLambdaBody

— **defun pfMLambdaBody 0** —

```
(defun |pfMLambdaBody| (pf)
 (caddr pf))
```

————-

15.5.102 defun Is this a Not node?

[pfAbSynOp? p624]

— **defun pfNot?** —

```
(defun |pfNot?| (pf)
 (|pfAbSynOp?| pf '|Not|))
```

————-

15.5.103 defun Return the Arg part of a Not node

— **defun pfNotArg 0** —

```
(defun |pfNotArg| (pf)
 (cadr pf))
```

————-

15.5.104 defun Construct a NoValue node

[pfTree p492]

— **defun pfNovalue** —

```
(defun |pfNovalue| (pfexpr)
 (|pfTree| '|Novalue| (list pfexpr)))
```

————-

15.5.105 defun Is this a Novalue node?

[pfAbSynOp? p624]

— **defun pfNovalue?** —

```
(defun |pfNovalue?| (pf)
 (|pfAbSynOp?| pf '|Novalue|))
```

————-

15.5.106 defun Return the Expr part of a Novalue node

— **defun pfNovalueExpr 0** —

```
(defun |pfNovalueExpr| (pf)
 (cadr pf))
```

————-

15.5.107 defun Construct an Or node

[pfTree p492]

— **defun pfOr** —

```
(defun |pfOr| (pfleft pfright)
 (|pfTree| '|Or| (list pfleft pfright)))
```

————-

15.5.108 defun Is this an Or node?

[pfAbSynOp? p624]

— **defun pfOr?** —

```
(defun |pfOr?| (pf)
 (|pfAbSynOp?| pf '|Or|))
```

————-

15.5.109 defun Return the Left part of an Or node

— **defun pfOrLeft 0** —

```
(defun |pfOrLeft| (pf)
 (cadr pf))
```

———-

15.5.110 defun Return the Right part of an Or node

— defun pfOrRight 0 —

```
(defun |pfOrRight| (pf)
 (caddr pf))
```

———-

15.5.111 defun Return the part of a parenthesised expression

— defun pfParen —

```
(defun |pfParen| (a part)
 (declare (ignore a))
 part)
```

———-

15.5.112 defun pfPretend

[pfTree p492]

— defun pfPretend —

```
(defun |pfPretend| (pfexpr pftype)
 (|pfTree| '|Pretend| (list pfexpr pftype)))
```

———-

15.5.113 defun Is this a Pretend node?

[pfAbSynOp? p624]

— defun pfPretend? —

```
(defun |pfPretend?| (pf)
 (|pfAbSynOp?| pf '|Pretend|))
```

———-

15.5.114 defun Return the Expression part of a Pretend node

— **defun pfPretendExpr 0** —

```
(defun |pfPretendExpr| (pf)
 (cadr pf))
```

————-

15.5.115 defun Return the Type part of a Pretend node

— **defun pfPretendType 0** —

```
(defun |pfPretendType| (pf)
 (caddr pf))
```

————-

15.5.116 defun Construct a QualType node

[pfTree p492]

— **defun pfQualType** —

```
(defun |pfQualType| (pftype pfqual)
 (|pfTree| '|QualType| (list pftype pfqual)))
```

————-

15.5.117 defun Construct a Restrict node

[pfTree p492]

— **defun pfRestrict** —

```
(defun |pfRestrict| (pfexpr pftype)
 (|pfTree| '|Restrict| (list pfexpr pftype)))
```

————-

15.5.118 defun Is this a Restrict node?

[pfAbSynOp? p624]

— **defun pfRestrict?** —

```
(defun |pfRestrict?| (pf)
 (|pfAbSynOp?| pf '|Restrict|))
```

———-

15.5.119 defun Return the Expr part of a Restrict node

— **defun pfRestrictExpr 0** —

```
(defun |pfRestrictExpr| (pf)
 (cadr pf))
```

———-

15.5.120 defun Return the Type part of a Restrict node

— **defun pfRestrictType 0** —

```
(defun |pfRestrictType| (pf)
 (caddr pf))
```

———-

15.5.121 defun Construct a RetractTo node

[pfTree p492]

— **defun pfRetractTo** —

```
(defun |pfRetractTo| (pfexpr pftype)
 (|pfTree| '|RetractTo| (list pfexpr pftype)))
```

———-

15.5.122 defun Construct a Return node

[pfTree p492]

— **defun pfReturn** —

```
(defun |pfReturn| (pfexpr pffrom)
 (|pfTree| '|Return| (list pfexpr pffrom)))
```

———-

15.5.123 defun Is this a Return node?

[pfAbSynOp? p624]

— **defun pfReturn?** —

```
(defun |pfReturn?| (pf)
 (|pfAbSynOp?| pf '|Return|))
```

————-

15.5.124 defun Return the Expr part of a Return node

— **defun pfReturnExpr 0** —

```
(defun |pfReturnExpr| (pf)
 (cadr pf))
```

————-

15.5.125 defun pfReturnNoName

[pfReturn p519]
[pfNothing p486]

— **defun pfReturnNoName** —

```
(defun |pfReturnNoName| (|value|)
 (|pfReturn| |value| (|pfNothing|)))
```

————-

15.5.126 defun Construct a ReturnTyped node

[pfTree p492]

— **defun pfReturnTyped** —

```
(defun |pfReturnTyped| (type body)
 (|pfTree| '|returntyped| (list type body)))
```

————-

15.5.127 defun Construct a Rule node

[pfTree p492]

— **defun pfRule** —

```
(defun |pfRule| (pflhsitems pfrhs)
 (|pfTree| '|Rule| (list pflhsitems pfrhs)))
```

————-

15.5.128 defun Return the Lhs of a Rule node

— defun pfRuleLhsItems 0 —

```
(defun |pfRuleLhsItems| (pf)
 (cadr pf))
```

————-

15.5.129 defun Return the Rhs of a Rule node

— defun pfRuleRhs 0 —

```
(defun |pfRuleRhs| (pf)
 (caddr pf))
```

————-

15.5.130 defun Is this a Rule node?

[pfAbSynOp? p624]

— defun pfRule? —

```
(defun |pfRule?| (pf)
 (|pfAbSynOp?| pf '|Rule|))
```

————-

15.5.131 defun pfSecond

— defun pfSecond 0 —

```
(defun |pfSecond| (form)
 (caddr form))
```

————-

15.5.132 defun Construct a Sequence node

[pfTree p492]

— defun pfSequence —

```
(defun |pfSequence| (pfargs)
 (|pfTree| '|Sequence| (list pfargs)))
```

———-

15.5.133 defun Return the Args of a Sequence node

— **defun pfSequenceArgs 0** —

```
(defun |pfSequenceArgs| (pf)
 (cadr pf))
```

———-

15.5.134 defun Is this a Sequence node?

[pfAbSynOp? p624]

— **defun pfSequence?** —

```
(defun |pfSequence?| (pf)
 (|pfAbSynOp?| pf '|Sequence|))
```

———-

15.5.135 defun Return the parts of the Args of a Sequence node

[pfParts p489]
[pfSequenceArgs p522]

— **defun pf0SequenceArgs** —

```
(defun |pf0SequenceArgs| (pf)
 (|pfParts| (|pfSequenceArgs| pf)))
```

———-

15.5.136 defun Create a Suchthat node

[pfTree p492]

— **defun pfSuchthat** —

```
(defun |pfSuchthat| (pfcond)
 (|pfTree| '|Suchthat| (list pfcond)))
```

———-

15.5.137 defun Is this a SuchThat node?

[pfAbSynOp? p624]

— defun pfSuchthat? —

```
(defun |pfSuchthat?| (pf)
 (|pfAbSynOp?| pf '|Suchthat|))
```

———-

15.5.138 defun Return the Cond part of a SuchThat node

— defun pfSuchthatCond 0 —

```
(defun |pfSuchthatCond| (pf)
 (cadr pf))
```

———-

15.5.139 defun Create a Tagged node

[pfTree p492]

— defun pfTagged —

```
(defun |pfTagged| (pftag pfexpr)
 (|pfTree| '|Tagged| (list pftag pfexpr)))
```

———-

15.5.140 defun Is this a Tagged node?

[pfAbSynOp? p624]

— defun pfTagged? —

```
(defun |pfTagged?| (pf)
 (|pfAbSynOp?| pf '|Tagged|))
```

———-

15.5.141 defun Return the Expression portion of a Tagged node

— defun pfTaggedExpr 0 —

```
(defun |pfTaggedExpr| (pf)
 (caddr pf))
```

———-

15.5.142 defun Return the Tag of a Tagged node

— defun pfTaggedTag 0 —

```
(defun |pfTaggedTag| (pf)
 (cadr pf))
```

———-

15.5.143 defun pfTaggedToTyped

[pfTagged? p523]
[pfTaggedExpr p523]
[pfNothing p486]
[pfTaggedTag p524]
[pfId? p487]
[pfId p486]
[pfTyped p525]
[pfSuch p485]
[pfInfApplication p508]

— 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)))))
```

———-

15.5.144 defun pfTweakIf

[pfIfElse p507]
[pfNothing? p486]
[pfListOf p485]
[pfTree p492]

[pfIfCond p507]
[pfIfThen p507]

— **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))))
```

————-

15.5.145 defun Construct a Typed node

[pfTree p492]

— **defun pfTyped** —

```
(defun |pfTyped| (pfid pftype)
 (|pfTree| '|Typed| (list pfid pftype)))
```

————-

15.5.146 defun Is this a Typed node?

[pfAbSynOp? p624]

— **defun pfTyped?** —

```
(defun |pfTyped?| (pf)
 (|pfAbSynOp?| pf '|Typed|))
```

————-

15.5.147 defun Return the Type of a Typed node

— **defun pfTypedType 0** —

```
(defun |pfTypedType| (pf)
 (caddr pf))
```

————-

15.5.148 defun Return the Id of a Typed node

— **defun pfTypedId 0** —

```
(defun |pfTypedId| (pf)
 (cadr pf))
```

———-

15.5.149 defun Construct a Typing node

[pfTree p492]

— **defun pfTyping** —

```
(defun |pfTyping| (pfitems)
 (|pfTree| '|Typing| (list pfitems)))
```

———-

15.5.150 defun Return a Tuple node

[pfTree p492]

— **defun pfTuple** —

```
(defun |pfTuple| (pfparts)
 (|pfTree| '|Tuple| (list pfparts)))
```

———-

15.5.151 defun Return a Tuple from a List

[pfTuple p526]
[pfListOf p485]

— **defun pfTupleListOf** —

```
(defun |pfTupleListOf| (pfparts)
 (|pfTuple| (|pfListOf| pfparts)))
```

———-

15.5.152 defun Is this a Tuple node?

[pfAbSynOp? p624]

— **defun pfTuple?** —

```
(defun |pfTuple?| (pf)
 (|pfAbSynOp?| pf '|Tuple|))
```

———-

15.5.153 defun Return the Parts of a Tuple node

— defun pfTupleParts 0 —

```
(defun |pfTupleParts| (pf)
 (cadr pf))
```

————-

15.5.154 defun Return the parts of a Tuple

[pfParts p489]
[pfTupleParts p527]

— defun pf0TupleParts —

```
(defun |pf0TupleParts| (pf)
 (|pfParts| (|pfTupleParts| pf)))
```

————-

15.5.155 defun Return a list from a Sequence node

[pfSequence? p522]
[pfAppend p494]
[pf0SequenceArgs p522]
[pfListOf p485]

— defun pfUnSequence —

```
(defun |pfUnSequence| (x)
 (if (|pfSequence?| x)
  (|pfListOf| (|pfAppend| (|pf0SequenceArgs| x)))
  (|pfListOf| x)))
```

————-

15.5.156 defun The comment is attached to all signatutres

[pfWDeclare p528]
[pfParts p489]

— defun pfWDec —

```
(defun |pfWDec| (doc name)
 (mapcar #'(lambda (i) (|pfWDeclare| i doc)) (|pfParts| name)))
```

————-

15.5.157 defun Construct a WDeclare node

[pfTree p492]

— **defun pfWDeclare** —

```
(defun |pfWDeclare| (pfsignature pfdoc)
 (|pfTree| '|WDeclare| (list pfsignature pfdoc)))
```

———-

15.5.158 defun Construct a Where node

[pfTree p492]

— **defun pfWhere** —

```
(defun |pfWhere| (pfcontext pfexpr)
 (|pfTree| '|Where| (list pfcontext pfexpr)))
```

———-

15.5.159 defun Is this a Where node?

[pfAbSynOp? p624]

— **defun pfWhere?** —

```
(defun |pfWhere?| (pf)
 (|pfAbSynOp?| pf '|Where|))
```

———-

15.5.160 defun Return the parts of the Context of a Where node

[pfParts p489]
[pfWhereContext p528]

— **defun pf0WhereContext** —

```
(defun |pf0WhereContext| (pf)
 (|pfParts| (|pfWhereContext| pf)))
```

———-

15.5.161 defun Return the Context of a Where node

— **defun pfWhereContext 0** —

```
(defun |pfWhereContext| (pf)
 (cadr pf))
```

———-

15.5.162 defun Return the Expr part of a Where node

— **defun pfWhereExpr 0** —

```
(defun |pfWhereExpr| (pf)
 (caddr pf))
```

———-

15.5.163 defun Construct a While node

[pfTree p492]

— **defun pfWhile** —

```
(defun |pfWhile| (pfcond)
 (|pfTree| '|While| (list pfcond)))
```

———-

15.5.164 defun Is this a While node?

[pfAbSynOp? p624]

— **defun pfWhile?** —

```
(defun |pfWhile?| (pf)
 (|pfAbSynOp?| pf '|While|))
```

———-

15.5.165 defun Return the Cond part of a While node

— **defun pfWhileCond 0** —

```
(defun |pfWhileCond| (pf)
 (cadr pf))
```

———-

15.5.166 defun Construct a With node

[pfTree p492]

— **defun pfWith** —

```
(defun |pfWith| (pfbase pfwithin pfwithon)
 (|pfTree| '|With| (list pfbase pfwithin pfwithon)))
```

————-

15.5.167 defun Create a Wrong node

[pfTree p492]

— **defun pfWrong** —

```
(defun |pfWrong| (pfwhy pfrubble)
 (|pfTree| '|Wrong| (list pfwhy pfrubble)))
```

————-

15.5.168 defun Is this a Wrong node?

[pfAbSynOp? p624]

— **defun pfWrong?** —

```
(defun |pfWrong?| (pf)
 (|pfAbSynOp?| pf '|Wrong|))
```

————-

Chapter 16

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.

16.0.169 defun Pftree to s-expression translation

[pf2Sex1 p531]
[$insideSEQ p??]
[$insideApplication p??]
[$insideRule p??]
[$QuietCommand p306]

— **defun pf2Sex** —

```
(defun |pf2Sex| (pf)
 (let (|$insideSEQ| |$insideApplication| |$insideRule|)
 (declare (special |$insideSEQ| |$insideApplication| |$insideRule|
                   |$QuietCommand|))
  (setq |$QuietCommand| nil)
  (setq |$insideRule| nil)
  (setq |$insideApplication| nil)
  (setq |$insideSEQ| nil)
  (|pf2Sex1| pf)))
```

————-

16.0.170 defun Pftree to s-expression translation inner function

[pfNothing? p486]
[pfSymbol? p491]
[pfSymbolSymbol p492]
[pfLiteral? p488]

[pfLiteral2Sex p536]
[pfIdSymbol p487]
[pfApplication? p494]
[pfApplication2Sex p537]
[pfTuple? p526]
[pf2Sex1 p531]
[pf0TupleParts p527]
[pfIf? p507]
[pfIfCond p507]
[pfIfThen p507]
[pfIfElse p507]
[pfTagged? p523]
[pfTaggedTag p524]
[pfTaggedExpr p523]
[pfCoerceto? p497]
[pfCoercetoExpr p498]
[pfCoercetoType p498]
[pfPretend? p517]
[pfPretendExpr p518]
[pfPretendType p518]
[pfFromdom? p505]
[opTran p552]
[pfFromdomWhat p506]
[pfFromdomDomain p506]
[pfSequence? p522]
[pfSequence2Sex p541]
[pfExit? p501]
[pfExitCond p502]
[pfExitExpr p502]
[pfLoop? p512]
[loopIters2Sex p542]
[pf0LoopIterators p512]
[pfCollect? p499]
[pfCollect2Sex p545]
[pfForin? p504]
[pf0ForinLhs p504]
[pfForinWhole p505]
[pfWhile? p529]
[pfWhileCond p529]
[pfSuchthat? p523]
[keyedSystemError p**??**]
[pfSuchthatCond p523]
[pfDo? p500]
[pfDoBody p501]
[pfTyped? p525]
[pfTypedType p525]
[pfTypedId p525]
[pfAssign? p495]
[pf0AssignLhsItems p495]

[pfAssignRhs p495]
[pfDefinition? p500]
[pfDefinition2Sex p545]
[pfLambda? p510]
[pfLambda2Sex p548]
[pfMLambda? p514]
[pfRestrict? p518]
[pfRestrictExpr p519]
[pfRestrictType p519]
[pfFree? p503]
[pf0FreeItems p503]
[pfLocal? p511]
[pf0LocalItems p511]
[pfWrong? p530]
[spadThrow p**??**]
[pfAnd? p493]
[pfAndLeft p494]
[pfAndRight p494]
[pfOr? p516]
[pfOrLeft p516]
[pfOrRight p517]
[pfNot? p515]
[pfNotArg p515]
[pfNovalue? p516]
[pfNovalueExpr p516]
[pfRule? p521]
[pfRule2Sex p548]
[pfBreak? p497]
[pfBreakFrom p497]
[pfMacro? p513]
[pfReturn? p519]
[pfReturnExpr p520]
[pfIterate? p508]
[pfWhere? p528]
[pf0WhereContext p528]
[pfWhereExpr p529]
[pfAbSynOp p624]
[tokPart p625]
[$insideSEQ p**??**]
[$insideRule p**??**]
[$QuietCommand p306]

— defun pf2Sex1 —

```
(defun |pf2Sex1| (pf)
 (let (args idList type op tagPart tag s)
 (declare (special |$insideSEQ| |$insideRule| |$QuietCommand|))
  (cond
   ((|pfNothing?| pf) '|noBranch|)
   ((|pfSymbol?| pf)
    (if (eq |$insideRule| '|left|)
```

```
    (progn
     (setq s (|pfSymbolSymbol| pf))
     (list '|constant| (list 'quote s)))
    (list 'quote (|pfSymbolSymbol| pf))))
  ((|pfLiteral?| pf) (|pfLiteral2Sex| pf))
  ((|pfId?| pf)
    (if |$insideRule|
     (progn
       (setq s (|pfIdSymbol| pf))
       (if (member s '(|%pi| |%e| |%i|))
         s
         (list 'quote s)))
     (|pfIdSymbol| pf)))
  ((|pfApplication?| pf) (|pfApplication2Sex| pf))
  ((|pfTuple?| pf) (cons '|Tuple| (mapcar #'|pf2Sex1| (|pf0TupleParts| pf))))
  ((|pfIf?| pf)
    (list 'if (|pf2Sex1| (|pfIfCond| pf))
              (|pf2Sex1| (|pfIfThen| pf))
              (|pf2Sex1| (|pfIfElse| pf))))
  ((|pfTagged?| pf)
   (setq tag (|pfTaggedTag| pf))
   (setq tagPart
    (if (|pfTuple?| tag)
     (cons '|Tuple| (mapcar #'|pf2Sex1| (|pf0TupleParts| tag)))
     (|pf2Sex1| tag)))
   (list '|:| tagPart (|pf2Sex1| (|pfTaggedExpr| pf))))
  ((|pfCoerceto?| pf)
   (list '|::| (|pf2Sex1| (|pfCoercetoExpr| pf))
               (|pf2Sex1| (|pfCoercetoType| pf))))
  ((|pfPretend?| pf)
   (list '|pretend| (|pf2Sex1| (|pfPretendExpr| pf))
                    (|pf2Sex1| (|pfPretendType| pf))))
  ((|pfFromdom?| pf)
   (setq op (|opTran| (|pf2Sex1| (|pfFromdomWhat| pf))))
   (when (eq op '|braceFromCurly|) (setq op 'seq))
   (list '|$elt| (|pf2Sex1| (|pfFromdomDomain| pf)) op))
  ((|pfSequence?| pf) (|pfSequence2Sex| pf))
  ((|pfExit?| pf)
   (if |$insideSEQ|
    (list '|exit| (|pf2Sex1| (|pfExitCond| pf))
                  (|pf2Sex1| (|pfExitExpr| pf)))
    (list 'if (|pf2Sex1| (|pfExitCond| pf))
              (|pf2Sex1| (|pfExitExpr| pf)) '|noBranch|)))
  ((|pfLoop?| pf) (cons 'repeat (|loopIters2Sex| (|pf0LoopIterators| pf))))
  ((|pfCollect?| pf) (|pfCollect2Sex| pf))
  ((|pfForin?| pf)
   (cons 'in
    (append (mapcar #'|pf2Sex1| (|pf0ForinLhs| pf))
            (list (|pf2Sex1| (|pfForinWhole| pf))))))
  ((|pfWhile?| pf) (list 'while (|pf2Sex1| (|pfWhileCond| pf))))
  ((|pfSuchthat?| pf)
   (if (eq |$insideRule| '|left|)
    (|keyedSystemError| "Unexpected error in call to system function %1"
      (list "pf2Sex1: pfSuchThat"))
```

```
     (list '|\|| (|pf2Sex1| (|pfSuchthatCond| pf)))))
    ((|pfDo?| pf) (|pf2Sex1| (|pfDoBody| pf)))
    ((|pfTyped?| pf)
     (setq type (|pfTypedType| pf))
     (if (|pfNothing?| type)
      (|pf2Sex1| (|pfTypedId| pf))
      (list '|:| (|pf2Sex1| (|pfTypedId| pf)) (|pf2Sex1| (|pfTypedType| pf)))))
    ((|pfAssign?| pf)
     (setq idList (mapcar #'|pf2Sex1| (|pf0AssignLhsItems| pf)))
     (if (not (eql (length idList) 1))
      (setq idList (cons '|Tuple| idList))
      (setq idList (car idList)))
     (list 'let idList (|pf2Sex1| (|pfAssignRhs| pf))))
    ((|pfDefinition?| pf) (|pfDefinition2Sex| pf))
    ((|pfLambda?| pf) (|pfLambda2Sex| pf))
    ((|pfMLambda?| pf) '|/throwAway|)
    ((|pfRestrict?| pf)
     (list '@ (|pf2Sex1| (|pfRestrictExpr| pf))
              (|pf2Sex1| (|pfRestrictType| pf))))
    ((|pfFree?| pf) (cons '|free| (mapcar #'|pf2Sex1| (|pf0FreeItems| pf))))
    ((|pfLocal?| pf) (cons '|local| (mapcar #'|pf2Sex1| (|pf0LocalItems| pf))))
    ((|pfWrong?| pf) (|spadThrow|))
    ((|pfAnd?| pf)
     (list '|and| (|pf2Sex1| (|pfAndLeft| pf))
                  (|pf2Sex1| (|pfAndRight| pf))))
    ((|pfOr?| pf)
     (list '|or| (|pf2Sex1| (|pfOrLeft| pf))
                 (|pf2Sex1| (|pfOrRight| pf))))
    ((|pfNot?| pf) (list '|not| (|pf2Sex1| (|pfNotArg| pf))))
    ((|pfNovalue?| pf)
     (setq |$QuietCommand| t)
     (list 'seq (|pf2Sex1| (|pfNovalueExpr| pf))))
    ((|pfRule?| pf) (|pfRule2Sex| pf))
    ((|pfBreak?| pf) (list '|break| (|pfBreakFrom| pf)))
    ((|pfMacro?| pf) '|/throwAway|)
    ((|pfReturn?| pf) (list '|return| (|pf2Sex1| (|pfReturnExpr| pf))))
    ((|pfIterate?| pf) (list '|iterate|))
    ((|pfWhere?| pf)
     (setq args (mapcar #'|pf2Sex1| (|pf0WhereContext| pf)))
     (if (eql (length args) 1)
      (cons '|where| (cons (|pf2Sex1| (|pfWhereExpr| pf)) args))
      (list '|where| (|pf2Sex1| (|pfWhereExpr| pf)) (cons 'seq args))))
;   -- under strange circumstances/piling, system commands can wind
;   -- up in expressions. This just passes it through as a string for
;   -- the user to figure out what happened.
    ((eq (|pfAbSynOp| pf) '|command|) (|tokPart| pf))
    (t (|keyedSystemError| "Unexpected error in call to system function %1"
        (list "pf2Sex1"))))))
```

———-

16.0.171 defun Convert a Literal to an S-expression

[pfLiteralClass p489]
[pfLiteralString p489]
[float2Sex p536]
[pfSymbolSymbol p492]
[pfLeafToken p488]
[keyedSystemError p??]
[$insideRule p??]

— **defun pfLiteral2Sex** —

```
(defun |pfLiteral2Sex| (pf)
 (let (s type)
 (declare (special |$insideRule|))
  (setq type (|pfLiteralClass| pf))
  (cond
   ((eq type '|integer|) (read-from-string (|pfLiteralString| pf)))
   ((or (eq type '|string|) (eq type '|char|))
    (|pfLiteralString| pf))
   ((eq type '|float|) (|float2Sex| (|pfLiteralString| pf)))
   ((eq type '|symbol|)
    (if |$insideRule|
     (progn
      (setq s (|pfSymbolSymbol| pf))
      (list 'quote s))
     (|pfSymbolSymbol| pf)))
   ((eq type '|expression|) (list 'quote (|pfLeafToken| pf)))
   (t
    (|keyedSystemError| "Unexpected error in call to system function %1"
     (list "pfLiteral2Sex: unexpected form"))))))
```

———-

16.0.172 defun Convert a float to an S-expression

[$useBFasDefault p??]

— **defun float2Sex** —

```
(defun |float2Sex| (num)
 (let (exp frac bfForm fracPartString intPart dotIndex expPart mantPart eIndex)
 (declare (special |$useBFasDefault|))
  (setq eIndex (search "e" num))
  (if eIndex
   (setq mantPart (subseq num 0 eIndex))
   (setq mantPart num))
  (if eIndex
   (setq expPart (read-from-string (subseq num (+ eIndex 1))))
   (setq expPart 0))
  (setq dotIndex (search "." mantPart))
  (if dotIndex
   (setq intPart (read-from-string (subseq mantPart 0 dotIndex)))
```

```
   (setq intPart (read-from-string mantPart)))
  (if dotIndex
   (setq fracPartString (subseq mantPart (+ dotIndex 1)))
   (setq fracPartString 0))
  (setq bfForm
   (make-float intPart (read-from-string fracPartString)
                       (length fracPartString) expPart))
  (if |$useBFasDefault|
   (progn
    (setq frac (cadr bfForm))
    (setq exp (cddr bfForm))
    (list (list '|$elt| (list '|Float|) '|float|) frac exp 10))
   bfForm)))
```

———-

16.0.173 defun Change an Application node to an S-expression

[pfOp2Sex p539]
[pfApplicationOp p493]
[opTran p552]
[pf0TupleParts p527]
[pfApplicationArg p493]
[pfTuple? p526]
[pf2Sex1 p531]
[pf2Sex p531]
[pfSuchThat2Sex p539]
[hasOptArgs? p540]
[$insideApplication p??]
[$insideRule p??]

— **defun pfApplication2Sex** —

```
(defun |pfApplication2Sex| (pf)
 (let (|$insideApplication| x val realOp tmp1 qt argSex typeList args op)
 (declare (special |$insideApplication| |$insideRule|))
  (setq |$insideApplication| t)
  (setq op (|pfOp2Sex| (|pfApplicationOp| pf)))
  (setq op (|opTran| op))
  (cond
   ((eq op '->)
     (setq args (|pf0TupleParts| (|pfApplicationArg| pf)))
     (if (|pfTuple?| (car args))
      (setq typeList (mapcar #'|pf2Sex1| (|pf0TupleParts| (car args))))
      (setq typeList (list (|pf2Sex1| (car args)))))
     (setq args (cons (|pf2Sex1| (cadr args)) typeList))
     (cons '|Mapping| args))
   ((and (eq op '|:|) (eq |$insideRule| '|left|))
    (list '|multiple| (|pf2Sex| (|pfApplicationArg| pf))))
   ((and (eq op '?) (eq |$insideRule| '|left|))
    (list '|optional| (|pf2Sex| (|pfApplicationArg| pf))))
   (t
```

```
(setq args (|pfApplicationArg| pf))
(cond
 ((|pfTuple?| args)
  (if (and (eq op '|\||) (eq |$insideRule| '|left|))
   (|pfSuchThat2Sex| args)
   (progn
    (setq argSex (cdr (|pf2Sex1| args)))
    (cond
     ((eq op '>) (list '< (cadr argSex) (car argSex)))
     ((eq op '>=) (list '|not| (list '< (car argSex) (cadr argSex))))
     ((eq op '<=) (list '|not| (list '< (cadr argSex) (car argSex))))
     ((eq op 'and) (list '|and| (car argSex) (cadr argSex)))
     ((eq op 'or) (list '|or| (car argSex) (cadr argSex)))
     ((eq op '|Iterate|) (list '|iterate|))
     ((eq op '|by|) (cons 'by argSex))
     ((eq op '|braceFromCurly|)
      (if (and (consp argSex) (eq (car argSex) 'seq))
       argSex
       (cons 'seq argSex)))
     ((and (consp op)
           (progn
            (setq qt (car op))
            (setq tmp1 (cdr op))
            (and (consp tmp1)
                 (eq (cdr tmp1) nil)
                 (progn
                  (setq realOp (car tmp1))
                  t)))
           (eq qt 'quote))
      (cons '|applyQuote| (cons op argSex)))
     ((setq val (|hasOptArgs?| argSex)) (cons op val))
     (t (cons op argSex))))))
 ((and (consp op)
       (progn
        (setq qt (car op))
        (setq tmp1 (cdr op))
        (and (consp tmp1)
             (eq (cdr tmp1) NIL)
             (progn
              (setq realOp (car tmp1))
              t)))
       (eq qt 'quote))
  (list '|applyQuote| op (|pf2Sex1| args)))
 ((eq op '|braceFromCurly|)
  (setq x (|pf2Sex1| args))
  (if (and (consp x) (eq (car x) 'seq))
   x
   (list 'seq x)))
 ((eq op '|by|) (list 'by (|pf2Sex1| args)))
 (t (list op (|pf2Sex1| args)))))))
```

———-

16.0.174 defun Convert a SuchThat node to an S-expression

[pf0TupleParts p527]
[pf2Sex1 p531]
[pf2Sex p531]
[$predicateList p??]

— **defun pfSuchThat2Sex** —

```
(defun |pfSuchThat2Sex| (args)
 (let (rhsSex lhsSex argList name)
 (declare (special |$predicateList|))
  (setq name (gentemp))
  (setq argList (|pf0TupleParts| args))
  (setq lhsSex (|pf2Sex1| (car argList)))
  (setq rhsSex (|pf2Sex| (cadr argList)))
  (setq |$predicateList|
   (cons (cons name (cons lhsSex rhsSex)) |$predicateList|))
  name))
```

———-

16.0.175 defun pfOp2Sex

[pf2Sex1 p531]
[pmDontQuote? p540]
[pfSymbol? p491]
[$quotedOpList p??]
[$insideRule p??]

— **defun pfOp2Sex** —

```
(defun |pfOp2Sex| (pf)
 (let (realOp tmp1 op alreadyQuoted)
 (declare (special |$quotedOpList| |$insideRule|))
  (setq alreadyQuoted (|pfSymbol?| pf))
  (setq op (|pf2Sex1| pf))
  (cond
   ((and (consp op)
         (eq (car op) 'quote)
         (progn
          (setq tmp1 (cdr op))
          (and (consp tmp1)
               (eq (cdr tmp1) nil)
               (progn
                (setq realOp (car tmp1)) t))))
    (cond
     ((eq |$insideRule| '|left|) realOp)
     ((eq |$insideRule| '|right|)
      (cond
       ((|pmDontQuote?| realOp) realOp)
       (t
        (setq |$quotedOpList| (cons op |$quotedOpList|))
```

```
       op)))
    ((eq realOp '|\||) realOp)
    ((eq realOp '|:|) realOp)
    ((eq realOp '?) realOp)
    (t op)))
  (t op))))
```

———-

16.0.176 defun pmDontQuote?

— defun pmDontQuote? 0 —

```
(defun |pmDontQuote?| (sy)
 (member sy
  '(+ - * ** ^ / |log| |exp| |pi| |sqrt| |ei| |li| |erf| |ci|
      |si| |dilog| |sin| |cos| |tan| |cot| |sec| |csc| |asin|
      |acos| |atan| |acot| |asec| |acsc| |sinh| |cosh| |tanh|
      |coth| |sech| |csch| |asinh| |acosh| |atanh| |acoth|
      |asech| |acsc|)))
```

———-

16.0.177 defun hasOptArgs?

— defun hasOptArgs? 0 —

```
(defun |hasOptArgs?| (argSex)
 (let (rhs  lhs opt nonOpt tmp1 tmp2)
 (dolist (arg argSex)
  (cond
   ((and (consp arg)
         (eq (car arg) 'optarg)
         (progn
          (setq tmp1 (cdr arg))
          (and (consp tmp1)
               (progn
                (setq lhs (car tmp1))
                (setq tmp2 (cdr tmp1))
                (and (consp tmp2)
                     (eq (cdr tmp2) nil)
                     (progn
                      (setq rhs (car tmp2))
                      t))))))
    (setq opt (cons (list lhs rhs) opt)))
   (t (setq nonOpt (cons arg nonOpt)))))
 (when opt
  (nconc (nreverse nonOpt) (list (cons '|construct| (nreverse opt)))))))
```

———-

16.0.178 defun Convert a Sequence node to an S-expression

[pf2Sex1 p531]
[pf0SequenceArgs p522]
[$insideSEQ p??]

— **defun pfSequence2Sex** —

```
(defun |pfSequence2Sex| (pf)
 (let (|$insideSEQ| tmp1 ruleList seq)
 (declare (special |$insideSEQ|))
  (setq |$insideSEQ| t)
  (setq seq (|pfSequence2Sex0| (mapcar #'|pf2Sex1| (|pf0SequenceArgs| pf))))
  (cond
   ((and (consp seq)
         (eq (car seq) 'seq)
         (progn (setq ruleList (cdr seq)) 't)
         (consp ruleList)
         (progn
          (setq tmp1 (car ruleList))
          (and (consp tmp1) (eq (car tmp1) '|rule|))))
    (list '|ruleset| (cons '|construct| ruleList)))
   (t seq))))
```

———-

16.0.179 defun pfSequence2Sex0

TPDHERE: rewrite this using (dolist (item seqList)...)

```
;pfSequence2Sex0 seqList ==
;  null seqList => "noBranch"
;  seqTranList := []
;  while seqList ^= nil repeat
;    item := first seqList
;    item is ["exit", cond, value] =>
;      item := ["IF", cond, value, pfSequence2Sex0 rest seqList]
;      seqTranList := [item, :seqTranList]
;      seqList := nil
;    seqTranList := [item ,:seqTranList]
;    seqList := rest seqList
;  #seqTranList = 1 => first seqTranList
;  ["SEQ", :nreverse seqTranList]
```

[pfSequence2Sex0 p541]

— **defun pfSequence2Sex0** —

```
(defun |pfSequence2Sex0| (seqList)
 (let (value tmp2 cond tmp1 item seqTranList)
  (if (null seqList)
   '|noBranch|
    (progn
     ((lambda ()
```

```
      (loop
       (if (not seqList)
        (return nil)
        (progn
          (setq item (car seqList))
          (cond
           ((and (consp item)
                 (eq (car item) '|exit|)
                 (progn
                  (setq tmp1 (cdr item))
                  (and (consp tmp1)
                       (progn
                        (setq cond (car tmp1))
                        (setq tmp2 (cdr tmp1))
                        (and (consp tmp2)
                             (eq (cdr tmp2) nil)
                             (progn
                              (setq value (car tmp2))
                              t))))))
              (setq item
               (list 'if cond value (|pfSequence2Sex0| (cdr seqList))))
              (setq seqTranList (cons item seqTranList))
              (setq seqList nil))
           (t
            (progn
             (setq seqTranList (cons item seqTranList))
             (setq seqList (cdr seqList))))))))
    (if (eql (length seqTranList) 1)
     (car seqTranList)
     (cons 'seq (nreverse seqTranList)))))))
```

———-

16.0.180 defun Convert a loop node to an S-expression

TPDHERE: rewrite using dsetq

```
;loopIters2Sex iterList ==
;  result := nil
;  for iter in iterList repeat
;    sex := pf2Sex1 iter
;    sex is ['IN, var, ['SEGMENT, i, ["BY", incr]]] =>
;      result := [ ['STEP, var, i, incr], :result]
;    sex is ['IN, var, ["BY", ['SEGMENT, i, j], incr]] =>
;      result := [ ['STEP, var, i, incr, j], :result]
;    sex is ['IN, var, ['SEGMENT, i, j]] =>
;      result := [ ['STEP, var, i, 1, j], :result]
;    result := [sex, :result]
;  nreverse result
```

[pf2Sex1 p531]

— **defun loopIters2Sex** —

```
(defun |loopIters2Sex| (iterList)
 (let (j incr i var sex result tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 tmp8)
  (dolist (iter iterList (nreverse result))
   (setq sex (|pf2Sex1| iter))
   (cond
    ((and (consp sex)
          (eq (car sex) 'in)
          (progn
           (setq tmp1 (cdr sex))
           (and (consp tmp1)
                (progn
                 (setq var (car tmp1))
                 (setq tmp2 (cdr tmp1))
                 (and (consp tmp2)
                      (eq (cdr tmp2) nil)
                      (progn
                       (setq tmp3 (car tmp2))
                       (and (consp tmp3)
                            (eq (car tmp3) 'segment)
                            (progn
                             (setq tmp4 (cdr tmp3))
                             (and (consp tmp4)
                                  (progn
                                   (setq i (car tmp4))
                                   (setq tmp5 (cdr tmp4))
                                   (and (consp tmp5)
                                        (eq (cdr tmp5) nil)
                                        (progn
                                         (setq tmp6 (car tmp5))
                                         (and (consp tmp6)
                                              (eq (car tmp6) 'by)
                                              (progn
                                               (setq tmp7 (cdr tmp6))
                                               (and (consp tmp7)
                                                    (eq (cdr tmp7) nil)
                                                    (progn
                                                     (setq incr (car tmp7))
                                                     t)))))))))))))))))
     (setq result (cons (list 'step var i incr) result)))
    ((and (consp sex)
          (eq (car sex) 'in)
          (progn
           (setq tmp1 (cdr sex))
           (and (consp tmp1)
                (progn
                 (setq var (car tmp1))
                 (setq tmp2 (cdr tmp1))
                 (and (consp tmp2)
                      (eq (cdr tmp2) nil)
                      (progn
                       (setq tmp3 (car tmp2))
                       (and (consp tmp3)
                            (eq (car tmp3) 'by)
                            (progn
```

```
                          (setq tmp4 (cdr tmp3))
                          (and (consp tmp4)
                           (progn
                            (setq tmp5 (car tmp4))
                            (and (consp tmp5)
                                 (eq (car tmp5) 'segment)
                                 (progn
                                  (setq tmp6 (cdr tmp5))
                                  (and (consp tmp6)
                                       (progn
                                        (setq i (car tmp6))
                                        (setq tmp7 (cdr tmp6))
                                        (and (consp tmp7)
                                             (eq (cdr tmp7) nil)
                                             (progn
                                              (setq j (car tmp7))
                                              t)))))))
                           (progn
                            (setq tmp8 (cdr tmp4))
                            (and (consp tmp8)
                                 (eq (cdr tmp8) nil)
                                 (progn
                                  (setq incr (car tmp8))
                                  t)))))))))))
 (setq result (cons (list 'step var i incr j) result)))
((and (consp sex)
      (eq (car sex) 'in)
      (progn
       (setq tmp1 (cdr sex))
       (and (consp tmp1)
            (progn
             (setq var (car tmp1))
             (setq tmp2 (cdr tmp1))
             (and (consp tmp2)
                  (eq (cdr tmp2) nil)
                  (progn
                   (setq tmp3 (car tmp2))
                   (and (consp tmp3)
                        (eq (car tmp3) 'segment)
                        (progn
                         (setq tmp4 (cdr tmp3))
                         (and (consp tmp4)
                              (progn
                               (setq i (car tmp4))
                               (setq tmp5 (cdr tmp4))
                               (and (consp tmp5)
                                    (eq (cdr tmp5) nil)
                                    (progn
                                     (setq j (car tmp5))
                                     t)))))))))))))
 (setq result (cons (list 'step var i 1 j) result)))
(t (setq result (cons sex result)))))))
```

———-

16.0.181 defun Change a Collect node to an S-expression

[loopIters2Sex p542]
[pfParts p489]
[pfCollectIterators p498]
[pf2Sex1 p531]
[pfCollectBody p498]

— **defun pfCollect2Sex** —

```
(defun |pfCollect2Sex| (pf)
 (let (var cond sex tmp1 tmp2 tmp3 tmp4)
  (setq sex
   (cons 'collect
    (append (|loopIters2Sex| (|pfParts| (|pfCollectIterators| pf)))
            (list (|pf2Sex1| (|pfCollectBody| pf))))))
  (cond
   ((and (consp sex)
         (eq (car sex) 'collect)
         (progn
          (setq tmp1 (cdr sex))
          (and (consp tmp1)
               (progn
                (setq tmp2 (car tmp1))
                (and (consp tmp2)
                     (eq (car tmp2) '|\||)
                         (progn
                          (setq tmp3 (cdr tmp2))
                          (and (consp tmp3)
                               (eq (cdr tmp3) nil)
                               (progn
                                (setq cond (car tmp3))
                                t)))))
               (progn
                (setq tmp4 (cdr tmp1))
                (and (consp tmp4)
                     (eq (cdr tmp4) nil)
                         (progn (setq var (car tmp4)) t)))))
         (symbolp var))
    (list '|\|| var cond))
   (t sex))))
```

———-

16.0.182 defun Convert a Definition node to an S-expression

[pf2Sex1 p531]
[pf0DefinitionLhsItems p500]
[pfDefinitionRhs p499]

[systemError p??]
[pfLambdaTran p546]
[$insideApplication p??]

— **defun pfDefinition2Sex** —

```
(defun |pfDefinition2Sex| (pf)
 (let (body argList tmp1 rhs id idList)
 (declare (special |$insideApplication|))
  (if |$insideApplication|
   (list 'optarg
    (|pf2Sex1| (car (|pf0DefinitionLhsItems| pf)))
    (|pf2Sex1| (|pfDefinitionRhs| pf)))
   (progn
    (setq idList (mapcar #'|pf2Sex1| (|pf0DefinitionLhsItems| pf)))
    (if (not (eql (length idList) 1))
     (|systemError|
       "lhs of definition must be a single item in the interpreter")
     (progn
      (setq id (car idList))
      (setq rhs (|pfDefinitionRhs| pf))
      (setq tmp1 (|pfLambdaTran| rhs))
      (setq argList (car tmp1))
      (setq body (cdr tmp1))
      (cons 'def
       (cons
        (if (eq argList '|id|)
         id
         (cons id argList))
        body))))))))
```

———-

16.0.183 defun Convert a Lambda node to an S-expression

[pfLambda? p510]
[pf0LambdaArgs p510]
[pfTyped? p525]
[pfCollectArgTran p547]
[pfTypedId p525]
[pfNothing? p486]
[pfTypedType p525]
[pf2Sex1 p531]
[systemError p??]
[pfLambdaRets p510]
[pfLambdaBody p510]

— **defun pfLambdaTran** —

```
(defun |pfLambdaTran| (pf)
 (let (retType argList argTypeList)
  (cond
```

```
  ((|pfLambda?| pf)
   (dolist (arg (|pf0LambdaArgs| pf))
    (if (|pfTyped?| arg)
     (progn
      (setq argList
       (cons (|pfCollectArgTran| (|pfTypedId| arg)) argList))
      (if (|pfNothing?| (|pfTypedType| arg))
       (setq argTypeList (cons nil argTypeList))
       (setq argTypeList
        (cons (|pf2Sex1| (|pfTypedType| arg)) argTypeList))))
     (|systemError| "definition args should be typed")))
   (setq argList (nreverse argList))
   (unless (|pfNothing?| (|pfLambdaRets| pf))
    (setq retType (|pf2Sex1| (|pfLambdaRets| pf))))
   (setq argTypeList (cons retType (nreverse argTypeList)))
   (cons argList
    (list argTypeList
          (mapcar #'(lambda (x) (declare (ignore x)) nil) argTypeList)
          (|pf2Sex1| (|pfLambdaBody| pf)))))
  (t (cons '|id| (list '(nil) '(nil) (|pf2Sex1| pf)))))))
```

———-

16.0.184 defun pfCollectArgTran

[pfCollect? p499]
[pf2sex1 p**??**]
[pfParts p489]
[pfCollectIterators p498]
[pfCollectBody p498]

— **defun pfCollectArgTran** —

```
(defun |pfCollectArgTran| (pf)
 (let (cond tmp2 tmp1 id conds)
  (cond
   ((|pfCollect?| pf)
    (setq conds (mapcar #'|pf2sex1| (|pfParts| (|pfCollectIterators| pf))))
    (setq id (|pf2Sex1| (|pfCollectBody| pf)))
    (cond
     ((and (consp conds)                    ; conds is [ ["|", cond] ]
           (eq (cdr conds) nil)
           (progn
            (setq tmp1 (car conds))
            (and (consp tmp1)
                 (eq (car tmp1) '|\||)
                 (progn
                  (setq tmp2 (cdr tmp1))
                  (and (consp tmp2)
                       (eq (cdr tmp2) nil)
                       (progn
                        (setq cond (car tmp2))
                        t))))))
```

```
            (list '|\|| id cond))
     (t (cons id conds))))
   (t (|pf2Sex1| pf)))))
```

———-

16.0.185 defun Convert a Lambda node to an S-expression

[pfLambdaTran p546]

— **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))))
```

———-

16.0.186 defun Convert a Rule node to an S-expression

[pfLhsRule2Sex p549]
[pfRuleLhsItems p521]
[pfRhsRule2Sex p549]
[pfRuleRhs p521]
[ruleLhsTran p552]
[rulePredicateTran p549]
[$multiVarPredicateList p??]
[$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)))))
```

———-

16.0.187 defun Convert the Lhs of a Rule to an S-expression

[pf2Sex1 p531]
[$insideRule p??]

— **defun pfLhsRule2Sex** —

```
(defun |pfLhsRule2Sex| (lhs)
 (let (|$insideRule|)
 (declare (special |$insideRule|))
  (setq |$insideRule| '|left|)
  (|pf2Sex1| lhs)))
```

———-

16.0.188 defun Convert the Rhs of a Rule to an S-expression

[pf2Sex1 p531]
[$insideRule p??]

— **defun pfRhsRule2Sex** —

```
(defun |pfRhsRule2Sex| (rhs)
 (let (|$insideRule|)
 (declare (special |$insideRule|))
  (setq |$insideRule| '|right|)
  (|pf2Sex1| rhs)))
```

———-

16.0.189 defun Convert a Rule predicate to an S-expression

```
;rulePredicateTran rule ==
;  null $multiVarPredicateList => rule
;  varList := patternVarsOf [rhs for [.,.,:rhs] in $multiVarPredicateList]
;  predBody :=
;    CDR $multiVarPredicateList =>
;      ['AND, :[:pvarPredTran(rhs, varList) for [.,.,:rhs] in
;        $multiVarPredicateList]]
;    [ [.,.,:rhs],:.] := $multiVarPredicateList
;    pvarPredTran(rhs, varList)
;  ['suchThat, rule,
;   ['construct, :[ ["QUOTE", var] for var in varList]],
;    ['ADEF, '(predicateVariable),
;     '((Boolean) (List (Expression (Integer)))), '(() ()),
;      predBody]]
```

[patternVarsOf p551]
[pvarPredTran p552]
[$multiVarPredicateList p??]

— **defun rulePredicateTran** —

```
(defun |rulePredicateTran| (rule)
 (let (predBody varList rhs tmp1 result)
 (declare (special |$multiVarPredicateList|))
  (if (null |$multiVarPredicateList|)
   rule
   (progn
   (setq varList
    (|patternVarsOf|
     ((lambda (t1 t2 t3)
      (loop
       (cond
        ((or (atom t2)
             (progn
              (setq t3 (car t2))
              nil))
           (return (nreverse t1)))
        (t
         (and (consp t3)
              (progn
               (setq tmp1 (cdr t3))
               (and (consp tmp1)
                    (progn
                     (setq rhs (cdr tmp1))
                     t)))
               (setq t1 (cons rhs t1)))))
       (setq t2 (cdr t2))))
      nil |$multiVarPredicateList| nil)))
   (setq predBody
    (cond
     ((cdr |$multiVarPredicateList|)
      (cons 'and
       ((lambda (t4 t5 t6)
        (loop
         (cond
          ((or (atom t5)
               (progn
                (setq t6 (car t5))
                nil))
            (return (nreverse t4)))
          (t
           (and (consp t6)
                (progn
                 (setq tmp1 (cdr t6))
                 (and (consp tmp1)
                      (progn
                       (setq rhs (cdr tmp1))
                       t)))
                      (setq t4
                       (append (reverse (|pvarPredTran| rhs varList))
                               t4)))))
         (setq t5 (cdr t5))))
        nil |$multiVarPredicateList| nil)))
     (t
      (progn
```

```
      (setq rhs (cddar |$multiVarPredicateList|))
      (|pvarPredTran| rhs varList)))))
  (dolist (var varList) (push (list 'quote var) result))
  (list '|suchThat| rule
   (cons '|construct| (nreverse result))
   (list 'adef '(|predicateVariable|)
                    '(((|Boolean|)
                      (|List| (|Expression| (|Integer|))))
                    '(nil nil) predBody))))))
```

———-

16.0.190 defun patternVarsOf

[patternVarsOf1 p551]

— **defun patternVarsOf** —

```
(defun |patternVarsOf| (expr)
 (|patternVarsOf1| expr nil))
```

———-

16.0.191 defun patternVarsOf1

[patternVarsOf1 p551]

— **defun patternVarsOf1** —

```
(defun |patternVarsOf1| (expr varList)
 (let (argl op)
  (cond
   ((null expr) varList)
   ((atom expr)
    (cond
     ((null (symbolp expr)) varList)
     ((member expr varList) varList)
     (t (cons expr varList))))
   ((and (consp expr)
         (progn
          (setq op (car expr))
          (setq argl (cdr expr))
          t))
         (progn
           (dolist (arg argl)
            (setq varList (|patternVarsOf1| arg varList)))
            varList))
   (t varList))))
```

———-

16.0.192 defun pvarPredTran

— **defun pvarPredTran** —

```
(defun |pvarPredTran| (rhs varList)
 (let ((i 0))
  (dolist (var varList rhs)
   (setq rhs (nsubst (list '|elt| '|predicateVariable| (incf i)) var rhs)))))
```

———-

16.0.193 defun Convert the Lhs of a Rule node to an S-expression

[patternVarsOf p551]
[nsubst p??]
[$predicateList p??]
[$multiVarPredicateList p??]

— **defun ruleLhsTran** —

```
(defun |ruleLhsTran| (ruleLhs)
 (let (predicate var vars predRhs predLhs name)
 (declare (special |$predicateList| |$multiVarPredicateList|))
 (dolist (pred |$predicateList|)
  (setq name (car pred))
  (setq predLhs (cadr pred))
  (setq predRhs (cddr pred))
  (setq vars (|patternVarsOf| predRhs))
  (cond
   ((cdr vars)
    (setq ruleLhs (nsubst predLhs name ruleLhs))
    (setq |$multiVarPredicateList| (cons pred |$multiVarPredicateList|)))
   (t
    (setq var (cadr predLhs))
    (setq predicate
      (list '|suchThat| predLhs (list 'adef (list var)
            '((|Boolean|) (|Expression| (|Integer|))) '(nil nil) predRhs)))
    (setq ruleLhs (nsubst predicate name ruleLhs)))))
  ruleLhs))
```

———-

16.0.194 defun Translate ops into internal symbols

— **defun opTran 0** —

```
(defun |opTran| (op)
 (cond
  ((equal op '|..|) 'segment)
  ((eq op '[]) '|construct|)
```

```
((eq op '{}) '|braceFromCurly|)
((eq op 'is) '|is|)
(t op)))
```

Chapter 17

Stream Utilities

The input stream is parsed into a large s-expression by repeated calls to Delay. Delay takes a function f and an argument x and returns a list consisting of `("nonnullstream" f x)`. Eventually multiple calls are made and a large list structure is created that consists of `("nonnullstream" f x ("nonnullstream" f1 x1 ("nonnullstream" f2 x2...`

This delay structure is given to StreamNull which walks along the list looking at the head. If the head is "nonnullstream" then the function is applied to the argument.

So, in effect, the input is "zipped up" into a Delay data structure which is then evaluated by calling StreamNull. This "zippered stream" parser was a research project at IBM and Axiom was the testbed (which explains the strange parsing technique).

17.0.195 defun npNull

[StreamNull p555]

— **defun npNull** —

```
(defun |npNull| (x) (|StreamNull| x))
```

————-

17.0.196 defun StreamNull

[eqcar p**??**]

StreamNull : **Delay** → **Union(T,NIL)**

— **defun StreamNull 0** —

```
(defun |StreamNull| (delay)
 (let (parsepair)
  (cond
   ((or (null delay) (eqcar delay '|nullstream|)) t)
   (t
    ((lambda nil
     (loop
```

```
(cond
 ((not (eqcar delay '|nonnullstream|)) (return nil))
 (t
  (setq parsepair (apply (cadr delay) (cddr delay)))
  (rplaca delay (car parsepair))
  (rplacd delay (cdr parsepair)))))))
(eqcar delay '|nullstream|))))))
```

———-

Chapter 18

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

18.0.197 defun insertpile

[npNull p555]
[pilePlusComment p558]
[pilePlusComments p558]
[pileTree p558]
[pileCforest p561]

— **defun insertpile** —

```
(defun |insertpile| (s)
 (let (stream a t1 h1 t2 h tmp1)
  (cond
   ((|npNull| s) (list nil 0 nil s))
   (t
    (setq tmp1 (list (car s) (cdr s)))
    (setq h (car tmp1))
    (setq t2 (cadr tmp1))
    (cond
     ((|pilePlusComment| h)
       (setq tmp1 (|pilePlusComments| s))
       (setq h1 (car tmp1))
       (setq t1 (cadr tmp1))
       (setq a (|pileTree| (- 1) t1))
       (cons (list (|pileCforest|
                     (append h1 (cons (elt a 2) nil))))
             (elt a 3)))
```

```
     (t
      (setq stream (cadar s))
      (setq a (|pileTree| -1 s))
      (cons (list (list (elt a 2) stream)) (elt a 3))))))))
```

———-

18.0.198 defun pilePlusComment

[tokType p625]
[npNull p555]
[pilePlusComment p558]
[pilePlusComments p558]

— **defun pilePlusComment** —

```
(defun |pilePlusComment| (arg)
 (eq (|tokType| (caar arg)) '|comment|))
```

———-

18.0.199 defun pilePlusComments

— **defun pilePlusComments** —

```
(defun |pilePlusComments| (s)
 (let (t1 h1 t2 h tmp1)
  (cond
   ((|npNull| s) (list nil s))
   (t
    (setq tmp1 (list (car s) (cdr s)))
    (setq h (car tmp1))
    (setq t2 (cadr tmp1))
    (cond
     ((|pilePlusComment| h)
      (setq tmp1 (|pilePlusComments| t2))
      (setq h1 (car tmp1))
      (setq t1 (cadr tmp1))
      (list (cons h h1) t1))
     (t
      (list nil s)))))))
```

———-

18.0.200 defun pileTree

[npNull p555]
[pileColumn p559]
[pileForests p559]

— **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)))))))
```

———-

18.0.201 defun pileColumn

[tokPosn p625]

— **defun pileColumn** —

```
(defun |pileColumn| (arg)
 (cdr (|tokPosn| (caar arg))))
```

———-

18.0.202 defun pileForests

[pileForest p560]
[npNull p555]
[pileForests p559]
[pileCtree p561]

— **defun pileForests** —

```
(defun |pileForests| (h n s)
 (let (t1 h1 tmp1)
  (setq tmp1 (|pileForest| n s))
  (setq h1 (car tmp1))
  (setq t1 (cadr tmp1))
  (cond
   ((|npNull| h1) (list t n h s))
   (t (|pileForests| (|pileCtree| h h1) n t1)))))
```

———-

18.0.203 defun pileForest

[pileTree p558]
[pileForest1 p560]

— **defun pileForest** —

```
(defun |pileForest| (n s)
 (let (t1 h1 t2 h hh b tmp)
  (setq tmp (|pileTree| n s))
  (setq b (car tmp))
  (setq hh (cadr tmp))
  (setq h (caddr tmp))
  (setq t2 (cadddr tmp))
  (cond
   (b
    (setq tmp (|pileForest1| hh t2))
    (setq h1 (car tmp))
    (setq t1 (cadr tmp))
    (list (cons h h1) t1))
   (t
    (list nil s)))))
```

———-

18.0.204 defun pileForest1

[eqpileTree p561]
[pileForest1 p560]

— **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)))))
```

———-

18.0.205 defun eqpileTree

[npNull p555]
[pileColumn p559]
[pileForests p559]

— **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)))))))
```

———-

18.0.206 defun pileCtree

[dqAppend p566]
[pileCforest p561]

— **defun pileCtree** —

```
(defun |pileCtree| (x y)
 (|dqAppend| x (|pileCforest| y)))
```

———-

18.0.207 defun pileCforest

Only enpiles forests with >= 2 trees

[tokPart p625]
[enPile p562]
[separatePiles p562]

— **defun pileCforest** —

```
(defun |pileCforest| (x)
 (let (f)
  (cond
   ((null x) nil)
   ((null (cdr x)) (setq f (car x))
    (cond
     ((eq (|tokPart| (caar f)) 'if) (|enPile| f))
```

```
    (t f)))
  (t (|enPile| (|separatePiles| x))))))
```

———-

18.0.208 defun enPile

[dqConcat p565]
[dqUnit p565]
[tokConstruct p623]
[firstTokPosn p562]
[lastTokPosn p562]

— **defun enPile** —

```
(defun |enPile| (x)
 (|dqConcat|
  (list
   (|dqUnit| (|tokConstruct| '|key| 'settab (|firstTokPosn| x)))
   x
   (|dqUnit| (|tokConstruct| '|key| 'backtab (|lastTokPosn| x))))))
```

———-

18.0.209 defun firstTokPosn

[tokPosn p625]

— **defun firstTokPosn** —

```
(defun |firstTokPosn| (arg) (|tokPosn| (caar arg)))
```

———-

18.0.210 defun lastTokPosn

[tokPosn p625]

— **defun lastTokPosn** —

```
(defun |lastTokPosn| (arg) (|tokPosn| (cadr arg)))
```

———-

18.0.211 defun separatePiles

[dqUnit p565]
[tokConstruct p623]
[lastTokPosn p562]

[dqConcat p565]
[separatePiles p562]

— **defun separatePiles** —

```
(defun |separatePiles| (x)
 (let (semicolon a)
  (cond
   ((null x) nil)
   ((null (cdr x)) (car x))
   (t
    (setq a (car x))
    (setq semicolon
      (|dqUnit| (|tokConstruct| '|key| 'backset (|lastTokPosn| a))))
    (|dqConcat| (list a semicolon (|separatePiles| (cdr x))))))))
```

————-

Chapter 19

Dequeue Functions

The dqUnit makes a unit dq i.e. a dq with one item, from the item

19.0.212 defun dqUnit

— **defun dqUnit 0** —

```
(defun |dqUnit| (s)
 (let (a)
  (setq a (list s))
  (cons a a)))
```

————-

19.0.213 defun dqConcat

The dqConcat function concatenates a list of dq's, destroying all but the last

[dqAppend p566]
[dqConcat p565]

— **defun dqConcat** —

```
(defun |dqConcat| (ld)
 (cond
  ((null ld) nil)
  ((null (cdr ld)) (car ld))
  (t (|dqAppend| (car ld) (|dqConcat| (cdr ld))))))
```

————-

19.0.214 defun dqAppend

The dqAppend function appends 2 dq's, destroying the first

— **defun dqAppend 0** —

```
(defun |dqAppend| (x y)
 (cond
  ((null x) y)
  ((null y) x)
  (t
   (rplacd (cdr x) (car y))
   (rplacd x (cdr y)) x)))
```

———-

19.0.215 defun dqToList

— **defun dqToList 0** —

```
(defun |dqToList| (s)
 (when s (car s)))
```

———-

Chapter 20

Message Handling

20.1 The Line Object

20.1.1 defun Line object creation

This is called in only one place, the incLine1 function.

— **defun lnCreate 0** —

```
(defun |lnCreate| (extraBlanks string globalNum &rest optFileStuff)
 (let ((localNum (first optFileStuff))
       (filename (second optFileStuff)))
  (unless localNum (setq localNum 0))
  (list extraBlanks string globalNum localNum filename)))
```

———-

20.1.2 defun Line element 0; Extra blanks

— **defun lnExtraBlanks 0** —

```
(defun |lnExtraBlanks| (lineObject) (elt lineObject 0))
```

———-

20.1.3 defun Line element 1; String

— **defun lnString 0** —

```
(defun |lnString| (lineObject) (elt lineObject 1))
```

———-

20.1.4 defun Line element 2; Globlal number

— **defun lnGlobalNum 0** —

```
(defun |lnGlobalNum| (lineObject) (elt lineObject 2))
```

————-

20.1.5 defun Line element 2; Set Global number

— **defun lnSetGlobalNum 0** —

```
(defun |lnSetGlobalNum| (lineObject num)
 (setf (elt lineObject 2) num))
```

————-

20.1.6 defun Line elemnt 3; Local number

— **defun lnLocalNum 0** —

```
(defun |lnLocalNum| (lineObject) (elt lineObject 3))
```

————-

20.1.7 defun Line element 4; Place of origin

— **defun lnPlaceOfOrigin 0** —

```
(defun |lnPlaceOfOrigin| (lineObject) (elt lineObject 4))
```

————-

20.1.8 defun Line element 4: Is it a filename?

[lnFileName? p569]

— **defun lnImmediate? 0** —

```
(defun |lnImmediate?| (lineObject) (null (|lnFileName?| lineObject)))
```

————-

20.1.9 defun Line element 4: Is it a filename?

— defun lnFileName? 0 —

```
(defun |lnFileName?| (lineObject)
 (let (filename)
  (when (consp (setq filename (elt lineObject 4))) filename)))
```

————-

20.1.10 defun Line element 4; Get filename

[lnFileName? p569]
[ncBug p587]

— defun lnFileName —

```
(defun |lnFileName| (lineObject)
 (let (fN)
  (if (setq fN (|lnFileName?| lineObject))
   fN
   (|ncBug| "there is no file name in %1" (list lineObject)))))
```

————-

20.2 Messages

20.2.1 defun msgCreate

```
msgObject
  tag -- catagory of msg
          -- attributes as a-list
               'imPr  => dont save for list processing
                toWhere, screen or file
                'norep => only display once in list
  pos -- position with possible FROM/TO tag
  key -- key for message database
  argL -- arguments to be placed in the msg test
  prefix -- things like "Error: "
  text -- the actual text
```

[setMsgForcedAttrList p583]
[putDatabaseStuff p585]
[initImPr p586]
[initToWhere p587]

— 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))
```

———-

20.2.2 defmacro getMsgPosTagOb

— **defmacro getMsgPosTagOb 0** —

```
(defmacro |getMsgPosTagOb| (msg)
 `(elt ,msg 1))
```

———-

20.2.3 defmacro getMsgKey

— **defmacro getMsgKey 0** —

```
(defmacro |getMsgKey| (msg)
 `(elt ,msg 2))
```

———-

20.2.4 defmacro getMsgArgL

— **defmacro getMsgArgL 0** —

```
(defmacro |getMsgArgL| (msg)
 `(elt ,msg 3))
```

———-

20.2.5 defmacro getMsgPrefix

— **defmacro getMsgPrefix 0** —

```
(defmacro |getMsgPrefix| (msg)
 `(elt ,msg 4))
```

———-

20.2.6 defmacro setMsgPrefix

— **defmacro setMsgPrefix 0** —

```
(defmacro |setMsgPrefix| (msg val)
 `(setf (elt ,msg 4) ,val))
```

————-

20.2.7 defmacro getMsgText

— **defmacro getMsgText 0** —

```
(defmacro |getMsgText| (msg)
 `(elt ,msg 5))
```

————-

20.2.8 defmacro setMsgText

— **defmacro setMsgText 0** —

```
(defmacro |setMsgText| (msg val)
  `(setf (elt ,msg 5) ,val))
```

————-

20.2.9 defmacro getMsgPrefix?

— **defmacro getMsgPrefix? 0** —

```
(defmacro |getMsgPrefix?| (msg)
 `(let ((pre (|getMsgPrefix| ,msg)))
   (unless (eq pre '|noPre|) pre)))
```

————-

20.2.10 defmacro getMsgTag

The valid message tags are: line, old, error, warn, bug, unimple, remark, stat, say, debug [ncTag p627]

— **defmacro getMsgTag 0** —

```
(defmacro |getMsgTag| (msg)
 `(|ncTag| ,msg))
```

———-

20.2.11 defmacro getMsgTag?

[ifcar p??]
[getMsgTag p571]

— **defmacro getMsgTag? 0** —

```
(defmacro |getMsgTag?| (msg)
 `(ifcar (member (|getMsgTag| ,msg)
         (list '|line| '|old| '|error| '|warn| '|bug|
               '|unimple| '|remark| '|stat| '|say| '|debug|))))
```

———-

20.2.12 defmacro line?

[getMsgTag p571]

— **defmacro line?** —

```
(defmacro |line?| (msg)
 `(eq (|getMsgTag| ,msg) '|line|))
```

———-

20.2.13 defmacro leader?

[getMsgTag p571]

— **defmacro leader?** —

```
(defmacro |leader?| (msg)
 `(eq (|getMsgTag| ,msg) '|leader|))
```

———-

20.2.14 defmacro toScreen?

[getMsgToWhere p582]

— **defmacro toScreen?** —

```
(defmacro |toScreen?| (msg)
 `(not (eq (|getMsgToWhere| ,msg) '|fileOnly|)))
```

———-

20.2.15 defun ncSoftError

Messages for the USERS of the compiler. The program being compiled has a minor error. Give a message and continue processing.

[desiredMsg p573]
[processKeyedError p574]
[msgCreate p569]
[$newcompErrorCount p288]

— **defun ncSoftError** —

```
(defun |ncSoftError| (pos erMsgKey erArgL &rest optAttr)
 (declare (special |$newcompErrorCount|))
  (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
  (when (|desiredMsg| erMsgKey)
   (|processKeyedError|
    (|msgCreate| '|error| pos erMsgKey erArgL
                 "Error" optAttr))))
```

———-

20.2.16 defun ncHardError

The program being compiled is seriously incorrect. Give message and throw to a recovery point.

[desiredMsg p573]
[processKeyedError p574]
[msgCreate p569]
[ncError p325]
[$newcompErrorCount p288]

— **defun ncHardError** —

```
(defun |ncHardError| (pos erMsgKey erArgL &rest optAttr)
 (let (erMsg)
 (declare (special |$newcompErrorCount|))
  (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
  (if (|desiredMsg| erMsgKey)
   (setq erMsg
    (|processKeyedError|
     (|msgCreate| '|error| pos erMsgKey erArgL "Error" optAttr)))
   (|ncError|))))
```

———-

20.2.17 defun desiredMsg

— **defun desiredMsg 0** —

```
(defun |desiredMsg| (erMsgKey &rest optCatFlag)
```

```
 (declare (ignore erMsgKey))
 (cond
  ((null (null optCatFlag)) (car optCatFlag))
  (t t)))
```

———-

20.2.18 defun processKeyedError

[getMsgTag? p572]
[getMsgKey p570]
[getMsgPrefix? p571]
[sayBrightly p??]
[CallerName p??]
[msgImPr? p578]
[msgOutputter p574]
[$ncMsgList p287]

— **defun processKeyedError** —

```
(defun |processKeyedError| (msg)
 (prog (pre erMsg)
 (declare (special |$ncMsgList|))
  (cond
   ((eq (|getMsgTag?| msg) '|old|)
     (setq erMsg (|getMsgKey| msg))
     (cond
      ((setq pre (|getMsgPrefix?| msg))
        (setq erMsg (cons pre erMsg))))
     (|sayBrightly| (cons "old msg from " (cons (|CallerName| 4) erMsg))))
   ((|msgImPr?| msg) (|msgOutputter| msg))
   (t (setq |$ncMsgList| (cons msg |$ncMsgList|))))))
```

———-

20.2.19 defun msgOutputter

[getStFromMsg p575]
[leader? p572]
[line? p572]
[toScreen? p572]
[flowSegmentedMsg p??]
[sayBrightly p??]
[toFile? p583]
[alreadyOpened? p583]
[$linelength p936]

— **defun msgOutputter** —

```
(defun |msgOutputter| (msg)
 (let (alreadyOpened shouldFlow st)
```

```
 (declare (special $linelength))
  (setq st (|getStFromMsg| msg))
  (setq shouldFlow (null (or (|leader?| msg) (|line?| msg))))
  (when (|toScreen?| msg)
    (when shouldFlow (setq st (|flowSegmentedMsg| st $linelength 0)))
    (|sayBrightly| st))
  (when (|toFile?| msg)
    (when shouldFlow (setq st (|flowSegmentedMsg| st (- $linelength 6) 0)))
    (setq alreadyOpened (|alreadyOpened?| msg)))))
```

———-

20.2.20 defun listOutputter

[msgOutputter p574]

— **defun listOutputter** —

```
(defun |listOutputter| (outputList)
 (dolist (msg outputList)
  (|msgOutputter| msg)))
```

———-

20.2.21 defun getStFromMsg

[getPreStL p576]
[getMsgPrefix? p571]
[getMsgTag p571]
[getMsgText p571]
[getPosStL p576]
[getMsgKey? p582]
[pname p1106]
[tabbing p582]

— **defun getStFromMsg** —

```
(defun |getStFromMsg| (msg)
 (let (st posStL preStL)
  (setq preStL (|getPreStL| (|getMsgPrefix?| msg)))
  (cond
   ((eq (|getMsgTag| msg) '|line|)
     (cons ""
       (cons "%x1" (append preStL (cons (|getMsgText| msg) nil)))))
   (t
    (setq posStL (|getPosStL| msg))
    (setq st
     (cons posStL
       (cons " "
        (append preStL
         (cons (|tabbing| msg)
```

```
          (|getMsgText| msg))))))))))
```

———-

20.2.22 defvar $preLength

— **initvars** —

```
(defvar |$preLength| 11)
```

———-

20.2.23 defun getPreStL

[size p1106]
[$preLength p576]

— **defun getPreStL 0** —

```
(defun |getPreStL| (optPre)
 (let (spses extraPlaces)
 (declare (special |$preLength|))
  (cond
   ((null optPre) (list "  "))
   (t
    (setq spses
     (cond
      ((< 0 (setq extraPlaces (- (- |$preLength| (size optPre)) 3)))
       (make-string extraPlaces))
      (t "")))
    (list optPre spses ":")))))
```

———-

20.2.24 defun getPosStL

[showMsgPos? p578]
[getMsgPos p579]
[msgImPr? p578]
[decideHowMuch p579]
[listDecideHowMuch p581]
[ppos p577]
[remLine p582]
[remFile p578]
[$lastPos p??]

— **defun getPosStL** —

```
(defun |getPosStL| (msg)
```

```
(let (printedOrigin printedLineNum printedFileName fullPrintedPos howMuch
      msgPos)
(declare (special |$lastPos|))
(cond
 ((null (|showMsgPos?| msg)) "")
 (t
  (setq msgPos (|getMsgPos| msg))
  (setq howMuch
   (if (|msgImPr?| msg)
    (|decideHowMuch| msgPos |$lastPos|)
    (|listDecideHowMuch| msgPos |$lastPos|)))
  (setq |$lastPos| msgPos)
  (setq fullPrintedPos (|ppos| msgPos))
  (setq printedFileName
   (cons "%x2" (cons "[" (append (|remLine| fullPrintedPos) (cons "]" nil)))))
  (setq printedLineNum
   (cons "%x2" (cons "[" (append (|remFile| fullPrintedPos) (cons "]" nil)))))
  (setq printedOrigin
   (cons "%x2" (cons "[" (append fullPrintedPos (cons "]" nil)))))
  (cond
   ((eq howMuch 'org)
    (cons "" (append printedOrigin (cons '|%l| nil))))
   ((eq howMuch 'line)
    (cons "" (append printedLineNum (cons '|%l| nil))))
   ((eq howMuch 'file)
    (cons "" (append printedFileName (cons '|%l| nil))))
   ((eq howMuch 'all)
    (cons ""
     (append printedFileName
      (cons '|%l|
       (cons ""
        (append printedLineNum
         (cons '|%l| nil)))))))
   (t ""))))))
```

———-

20.2.25 defun ppos

[pfNoPosition? p624]
[pfImmediate? p??]
[pfCharPosn p477]
[pfLinePosn p477]
[porigin p343]
[pfFileName p477]

— **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)))))
```

———-

20.2.26 defun remFile

[ifcdr p??]
[ifcar p??]

— **defun remFile** —

```
(defun |remFile| (positionList) (ifcdr (ifcdr positionList)))
```

———-

20.2.27 defun showMsgPos?

[msgImPr? p578]
[leader? p572]
[$erMsgToss p??]

— **defun showMsgPos? 0** —

```
(defun |showMsgPos?| (msg)
 (declare (special |$erMsgToss|))
 (or |$erMsgToss| (and (null (|msgImPr?| msg)) (null (|leader?| msg)))))
```

———-

20.2.28 defvar $imPrGuys

— **initvars** —

```
(defvar |$imPrGuys| (list '|imPr|))
```

———-

20.2.29 defun msgImPr?

[getMsgCatAttr p579]

— **defun msgImPr?** —

```
(defun |msgImPr?| (msg)
 (eq (|getMsgCatAttr| msg '|$imPrGuys|) '|imPr|))
```

———-

20.2.30 defun getMsgCatAttr

[ifcdr p**??**]
[qassq p**??**]
[ncAlist p627]

— **defun getMsgCatAttr** —

```
(defun |getMsgCatAttr| (msg cat)
 (ifcdr (qassq cat (|ncAlist| msg))))
```

———-

20.2.31 defun getMsgPos

[getMsgFTTag? p579]
[getMsgPosTagOb p570]

— **defun getMsgPos** —

```
(defun |getMsgPos| (msg)
 (if (|getMsgFTTag?| msg)
  (cadr (|getMsgPosTagOb| msg))
  (|getMsgPosTagOb| msg)))
```

———-

20.2.32 defun getMsgFTTag?

[ifcar p**??**]
[getMsgPosTagOb p570]

— **defun getMsgFTTag?** —

```
(defun |getMsgFTTag?| (msg)
 (ifcar (member (ifcar (|getMsgPosTagOb| msg)) (list 'from 'to 'fromto))))
```

———-

20.2.33 defun decideHowMuch

When printing a msg, we wish not to show pos information that was shown for a previous msg with identical pos info. org prints out the word noposition or console [poNopos? p580]
[poPosImmediate? p580]

[poFileName p580]
[poLinePosn p581]

— **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)))
```

———-

20.2.34 defun poNopos?

— **defun poNopos? 0** —

```
(defun |poNopos?| (posn)
 (equal posn (list '|noposition|)))
```

———-

20.2.35 defun poPosImmediate?

[poNopos? p580]
[lnImmediate? p568]
[poGetLineObject p581]

— **defun poPosImmediate?** —

```
(defun |poPosImmediate?| (txp)
 (unless (|poNopos?| txp) (|lnImmediate?| (|poGetLineObject| txp))))
```

———-

20.2.36 defun poFileName

[lnFileName p569]
[poGetLineObject p581]

— **defun poFileName** —

```
(defun |poFileName| (posn)
 (if posn
  (|lnFileName| (|poGetLineObject| posn))
```

```
  (caar posn)))
```

———-

20.2.37 defun poGetLineObject

— **defun poGetLineObject 0** —

```
(defun |poGetLineObject| (posn)
 (car posn))
```

———-

20.2.38 defun poLinePosn

[lnLocalNum p568]
[poGetLineObject p581]

— **defun poLinePosn** —

```
(defun |poLinePosn| (posn)
 (if posn
  (|lnLocalNum| (|poGetLineObject| posn))
  (cdar posn)))
```

———-

20.2.39 defun listDecideHowMuch

[poNopos? p580]
[poPosImmediate? p580]
[poGlobalLinePosn p328]

— **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)))
```

———-

20.2.40 defun remLine

— **defun remLine 0** —

```
(defun |remLine| (positionList) (list (ifcar positionList)))
```

————-

20.2.41 defun getMsgKey?

[identp p1107]

— **defun getMsgKey? 0** —

```
(defun |getMsgKey?| (msg)
 (let ((val (|getMsgKey| msg)))
  (when (identp val) val)))
```

————-

20.2.42 defun tabbing

[getMsgPrefix? p571]
[$preLength p576]

— **defun tabbing** —

```
(defun |tabbing| (msg)
 (let (chPos)
 (declare (special |$preLength|))
  (setq chPos 2)
  (when (|getMsgPrefix?| msg) (setq chPos (- (+ chPos |$preLength|) 1)))
  (cons '|%t| chPos)))
```

————-

20.2.43 defvar $toWhereGuys

— **initvars** —

```
(defvar |$toWhereGuys| (list '|fileOnly| '|screenOnly|))
```

————-

20.2.44 defun getMsgToWhere

[getMsgCatAttr p579]

— **defun getMsgToWhere** —

```
(defun |getMsgToWhere| (msg) (|getMsgCatAttr| msg '|$toWhereGuys|))
```

————-

20.2.45 defun toFile?

[getMsgToWhere p582]
[$fn p??]

— **defun toFile?** —

```
(defun |toFile?| (msg)
 (and (not (eq (|getMsgToWhere| msg) '|screenOnly|))))
```

————-

20.2.46 defun alreadyOpened?

[msgImPr? p578]

— **defun alreadyOpened?** —

```
(defun |alreadyOpened?| (msg) (null (|msgImPr?| msg)))
```

————-

20.2.47 defun setMsgForcedAttrList

[setMsgForcedAttr p583]
[whichCat p584]

— **defun setMsgForcedAttrList** —

```
(defun |setMsgForcedAttrList| (msg attrlist)
 (dolist (attr attrlist)
  (|setMsgForcedAttr| msg (|whichCat| attr) attr)))
```

————-

20.2.48 defun setMsgForcedAttr

[setMsgCatlessAttr p584]
[ncPutQ p628]

— **defun setMsgForcedAttr** —

```
(defun |setMsgForcedAttr| (msg cat attr)
 (if (eq cat '|catless|)
```

```
  (|setMsgCatlessAttr| msg attr)
  (|ncPutQ| msg cat attr)))
```

———-

20.2.49 defvar $attrCats

— **initvars** —

```
(defvar |$attrCats| (list '|$imPrGuys| '|$toWhereGuys| '|$repGuys|))
```

———-

20.2.50 defun whichCat

[ListMember? p??]
[$attrCats p584]

— **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))
```

———-

20.2.51 defun setMsgCatlessAttr

TPDHERE: Changed from —catless— to '—catless—

[ncPutQ p628]
[ifcdr p??]
[qassq p??]
[ncAlist p627]

— **defun setMsgCatlessAttr** —

```
(defun |setMsgCatlessAttr| (msg attr)
 (|ncPutQ| msg 'catless (cons attr (ifcdr (qassq 'catless (|ncAlist| msg))))))
```

———-

20.2.52 defun putDatabaseStuff

TPDHERE: The variable al is undefined [getMsgInfoFromKey p585]
[setMsgUnforcedAttrList p586]
[setMsgText p571]

— **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 attributes))
  (|setMsgText| msg text)))
```

————-

20.2.53 defun getMsgInfoFromKey

[getMsgKey? p582]
[getErFromDbL p??]
[getMsgKey p570]
[segmentKeyedMsg p40]
[removeAttributes p??]
[substituteSegmentedMsg p??]
[getMsgArgL p570]
[$msgDatabaseName p177]

— **defun getMsgInfoFromKey** —

```
(defun |getMsgInfoFromKey| (msg)
 (let (|$msgDatabaseName| attributes tmp msgText msgKey)
 (declare (special |$msgDatabaseName|))
  (setq |$msgDatabaseName| nil)
  (setq msgText
   (cond
    ((setq msgKey (|getMsgKey?| msg))
     msgKey)
    (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)))
```

————-

20.2.54 defun setMsgUnforcedAttrList

[setMsgUnforcedAttr p586]
[whichCat p584]

— **defun setMsgUnforcedAttrList** —

```
(defun |setMsgUnforcedAttrList| (msg attrlist)
 (dolist (attr attrlist)
  (|setMsgUnforcedAttr| msg (|whichCat| attr) attr)))
```

————-

20.2.55 defun setMsgUnforcedAttr

[setMsgCatlessAttr p584]
[qassq p??]
[ncAlist p627]
[ncPutQ p628]

— **defun setMsgUnforcedAttr** —

```
(defun |setMsgUnforcedAttr| (msg cat attr)
 (cond
  ((eq cat '|catless|) (|setMsgCatlessAttr| msg attr))
  ((null (qassq cat (|ncAlist| msg))) (|ncPutQ| msg cat attr))))
```

————-

20.2.56 defvar $imPrTagGuys

— **initvars** —

```
(defvar |$imPrTagGuys| (list '|unimple| '|bug| '|debug| '|say| '|warn|))
```

————-

20.2.57 defun initImPr

[getMsgTag p571]
[setMsgUnforcedAttr p586]
[$imPrTagGuys p586]
[$erMsgToss p??]

— **defun initImPr** —

```
(defun |initImPr| (msg)
 (declare (special |$imPrTagGuys| |$erMsgToss|))
 (when (or |$erMsgToss| (member (|getMsgTag| msg) |$imPrTagGuys|))
   (|setMsgUnforcedAttr| msg '|$imPrGuys| '|imPr|)))
```

———-

20.2.58 defun initToWhere

[getMsgCatAttr p579]
[setMsgUnforcedAttr p586]

— defun initToWhere —

```
(defun |initToWhere| (msg)
 (if (member '|trace| (|getMsgCatAttr| msg '|catless|))
   (|setMsgUnforcedAttr| msg '|$toWhereGuys| '|screenOnly|)))
```

———-

20.2.59 defun Report a bug in the compiler

Bug in the compiler: something which shouldn't have happened did.

[processKeyedError p574]
[msgCreate p569]
[enable-backtrace p??]
[ncAbort p??]
[$nopos p288]
[$newcompErrorCount p288]

— defun ncBug —

```
(defun |ncBug| (erMsgKey erArgL &rest optAttr)
 (let (erMsg)
 (declare (special |$nopos| |$newcompErrorCount|))
  (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
  (setq erMsg
   (|processKeyedError|
    (|msgCreate| '|bug| |$nopos| erMsgKey erArgL "Bug!" optAttr)))
  (break)
  (|ncAbort|)))
```

———-

20.2.60 defun processMsgList

[erMsgSort p588]
[makeMsgFromLine p589]
[poGlobalLinePosn p328]
[getMsgPos p579]
[queueUpErrors p590]
[listOutputter p575]
[$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|))))
```

————-

20.2.61 defun erMsgSort

[erMsgSep p589]
[listSort p??]

— **defun erMsgSort** —

```
(defun |erMsgSort| (erMsgList)
 (let (msgWOPos msgWPos tmp)
  (setq tmp (|erMsgSep| erMsgList))
  (setq msgWPos (car tmp))
  (setq msgWOPos (cadr tmp))
  (setq msgWPos (|listSort| #'|erMsgCompare| msgWPos))
  (setq msgWOPos (reverse msgWOPos))
  (append msgWPos msgWOPos)))
```

————-

20.2.62 defun erMsgCompare

[compareposns p589]
[getMsgPos p579]

— **defun erMsgCompare** —

```
(defun |erMsgCompare| (ob1 ob2)
 (|compareposns| (|getMsgPos| ob2) (|getMsgPos| ob1)))
```

————-

20.2.63 defun compareposns

[poGlobalLinePosn p328]
[poCharPosn p594]

— **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)))))
```

———-

20.2.64 defun erMsgSep

[poNopos? p580]
[getMsgPos p579]

— **defun erMsgSep** —

```
(defun |erMsgSep| (erMsgList)
 (let (msgWOPos msgWPos)
  (dolist (msg erMsgList)
   (if (|poNopos?| (|getMsgPos| msg))
    (setq msgWOPos (cons msg msgWOPos))
    (setq msgWPos (cons msg msgWPos))))
  (list msgWPos msgWOPos)))
```

———-

20.2.65 defun makeMsgFromLine

[getLinePos p590]
[getLineText p590]
[poGlobalLinePosn p328]
[poLinePosn p581]
[concat p1107]
[rep p590]
[char p**??**]
[size p1106]
[$preLength p576]

— **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
    (concat (|rep| #\space (- |$preLength| 7 (size stNum))) stNum))
  (list '|line| posOfLine nil nil (concat "Line" localNumOfLine) textOfLine)))
```

———-

20.2.66 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))
  ""))
```

———-

20.2.67 defun getLinePos

— defun getLinePos 0 —

```
(defun |getLinePos| (line) (car line))
```

———-

20.2.68 defun getLineText

— defun getLineText 0 —

```
(defun |getLineText| (line) (cdr line))
```

———-

20.2.69 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 p594]
[$outputList p??]

— defun queueUpErrors —

```
(DEFUN |queueUpErrors| (|globalNumOfLine| |msgList|)
  (PROG (|notThisPosMsgs| |notThisLineMsgs| |thisPosMsgs|)
    (DECLARE (SPECIAL |$outputList|))
    (RETURN
      (PROGN
        (SETQ |thisPosMsgs| NIL)
        (SETQ |notThisLineMsgs| NIL)
        ((LAMBDA (|bfVar#7| |msg|)
           (LOOP
             (COND
               ((OR (ATOM |bfVar#7|)
                    (PROGN (SETQ |msg| (CAR |bfVar#7|)) NIL)
                    (NOT (|thisPosIsLess| (|getMsgPos| |msg|)
                              |globalNumOfLine|)))
                (RETURN NIL))
               ('T
                (PROGN
                  (COND
                    ((NULL (|redundant| |msg| |notThisPosMsgs|))
                     (SETQ |notThisPosMsgs|
                           (CONS |msg| |notThisPosMsgs|))))
                  (SETQ |msgList| (CDR |msgList|)))))
             (SETQ |bfVar#7| (CDR |bfVar#7|))))
         |msgList| NIL)
        ((LAMBDA (|bfVar#8| |msg|)
           (LOOP
             (COND
               ((OR (ATOM |bfVar#8|)
                    (PROGN (SETQ |msg| (CAR |bfVar#8|)) NIL)
                    (NOT (|thisPosIsEqual| (|getMsgPos| |msg|)
                              |globalNumOfLine|)))
                (RETURN NIL))
               ('T
                (PROGN
                  (COND
                    ((NULL (|redundant| |msg| |thisPosMsgs|))
                     (SETQ |thisPosMsgs| (CONS |msg| |thisPosMsgs|))))
```

```
                (SETQ |msgList| (CDR |msgList|)))))
            (SETQ |bfVar#8| (CDR |bfVar#8|))))
         |msgList| NIL)
        (COND
          (|thisPosMsgs|
              (SETQ |thisPosMsgs|
                    (|processChPosesForOneLine| |thisPosMsgs|))
              (SETQ |$outputList| (NCONC |thisPosMsgs| |$outputList|))))
        (COND
          (|notThisPosMsgs|
              (SETQ |$outputList|
                    (NCONC |notThisPosMsgs| |$outputList|))))
        |msgList|))))
```

———-

20.2.70 defun thisPosIsLess

[poNopos? p580]
[poGlobalLinePosn p328]

— **defun thisPosIsLess** —

```
(defun |thisPosIsLess| (pos num)
 (unless (|poNopos?| pos) (< (|poGlobalLinePosn| pos) num)))
```

———-

20.2.71 defun thisPosIsEqual

[poNopos? p580]
[poGlobalLinePosn p328]

— **defun thisPosIsEqual** —

```
(defun |thisPosIsEqual| (pos num)
 (unless (|poNopos?| pos) (equal (|poGlobalLinePosn| pos) num)))
```

———-

20.2.72 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? p593]
[sameMsg? p593]
[$noRepList p??]

— defun redundant —

```
(defun |redundant| (msg thisPosMsgs)
 (prog (found)
 (declare (special |$noRepList|))
  (return
   (progn
    (cond
     ((|msgNoRep?| msg)
      ((lambda (Var9 item)
       (loop
        (cond
         ((or (atom Var9) (progn (setq item (car Var9)) nil))
           (return nil))
         (t
          (cond
           ((|sameMsg?| msg item) (return (setq found t))))))
        (setq Var9 (cdr Var9))))
       |$noRepList| nil)
      (setq |$noRepList| (list msg |$noRepList|))))
   (or found (member msg thisPosMsgs))))))
```

———-

20.2.73 defvar $repGuys

— initvars —

```
(defvar |$repGuys| (list '|noRep| '|rep|))
```

———-

20.2.74 defun msgNoRep?

[getMsgCatAttr p579]

— defun msgNoRep? —

```
(defun |msgNoRep?| (msg) (eq (|getMsgCatAttr| msg '|$repGuys|) '|noRep|))
```

———-

20.2.75 defun sameMsg?

[getMsgKey p570]
[getMsgArgL p570]

— **defun sameMsg?** —

```
(defun |sameMsg?| (msg1 msg2)
 (and (equal (|getMsgKey| msg1) (|getMsgKey| msg2))
      (equal (|getMsgArgL| msg1) (|getMsgArgL| msg2))))
```

———-

20.2.76 defun processChPosesForOneLine

[posPointers p595]
[getMsgFTTag? p579]
[putFTText p597]
[poCharPosn p594]
[getMsgPos p579]
[getMsgPrefix p570]
[setMsgPrefix p571]
[concat p1107]
[size p1106]
[makeLeaderMsg p595]
[$preLength p576]

— **defun processChPosesForOneLine** —

```
(defun |processChPosesForOneLine| (msgList)
 (let (leaderMsg oldPre posLetter chPosList)
 (declare (special |$preLength|))
  (setq chPosList (|posPointers| msgList))
  (dolist (msg msgList)
   (when (|getMsgFTTag?| msg) (|putFTText| msg chPosList))
   (setq posLetter (cdr (assoc (|poCharPosn| (|getMsgPos| msg)) chPosList)))
   (setq oldPre (|getMsgPrefix| msg))
   (|setMsgPrefix| msg
    (concat oldPre
     (make-string (- |$preLength| 4 (size oldPre))) posLetter)))
  (setq leaderMsg (|makeLeaderMsg| chPosList))
  (nconc msgList (list leaderMsg))))
```

———-

20.2.77 defun poCharPosn

— **defun poCharPosn 0** —

```
(defun |poCharPosn| (posn)
 (cdr posn))
```

———-

20.2.78 defun makeLeaderMsg

```
makeLeaderMsg chPosList ==
    st := MAKE_-FULL_-CVEC ($preLength- 3)
    oldPos := -1
    for [posNum,:posLetter] in reverse chPosList repeat
        st := CONCAT(st, _
            rep(char ".", (posNum - oldPos - 1)),posLetter)
        oldPos := posNum
    ['leader,$nopos,'nokey,NIL,NIL,[st] ]
```

[$nopos p288]
[$preLength p576]

— defun makeLeaderMsg —

```
(defun |makeLeaderMsg| (chPosList)
 (let (posLetter posNum oldPos st)
 (declare (special |$nopos| |$preLength|))
  (setq st (make-string (- |$preLength| 3)))
  (setq oldPos -1)
  ((lambda (Var15 Var14)
   (loop
    (cond
     ((or (atom Var15) (progn (setq Var14 (car Var15)) nil))
       (return nil))
     (t
      (and (consp Var14)
           (progn
            (setq posNum (car Var14))
            (setq posLetter (cdr Var14))
            t)
           (progn
            (setq st
             (concat st (|rep| #\. (- posNum oldPos 1)) posLetter))
            (setq oldPos posNum)))))
    (setq Var15 (cdr Var15))))
   (reverse chPosList) nil)
  (list '|leader| |$nopos| '|nokey| nil nil (list st))))
```

———-

20.2.79 defun posPointers

TPDHERE: getMsgFTTag is nonsense

[poCharPosn p594]
[getMsgPos p579]
[ifcar p??]
[getMsgPos2 p596]
[insertPos p596]
[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))
```

———-

20.2.80 defun getMsgPos2

[getMsgFTTag? p579]
[getMsgPosTagOb p570]
[ncBug p587]

— **defun getMsgPos2** —

```
(defun |getMsgPos2| (msg)
 (if (|getMsgFTTag?| msg)
  (caddr (|getMsgPosTagOb| msg))
  (|ncBug| "not a from to" nil)))
```

———-

20.2.81 defun insertPos

This function inserts a position in the proper place of a position list. This is used for the 2nd pos of a fromto [done p**??**]

— **defun insertPos 0** —

```
(defun |insertPos| (newPos posList)
 (let (pos top bot done)
 (setq bot (cons 0 posList))
 (do () (done)
  (setq top (cons (car bot) top))
```

```
  (setq bot (cdr bot))
  (setq pos (car bot))
  (setq done
   (cond
    ((< pos newPos) nil)
    ((equal pos newPos) t)
    ((< newPos pos)
     (setq top (cons newPos top))
     t))))
 (cons (cdr (reverse top)) bot)))
```

———-

20.2.82 defun putFTText

[getMsgFTTag? p579]
[poCharPosn p594]
[getMsgPos p579]
[setMsgText p571]
[getMsgText p571]
[getMsgPos2 p596]

— **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)))))))
```

———-

20.2.83 defun From

This is called from parameter list of nc message functions

— **defun From 0** —

```
(defun |From| (pos) (list 'from pos))
```

———-

20.2.84 defun To

This is called from parameter list of nc message functions

— **defun To 0** —

```
(defun |To| (pos) (list 'to pos))
```

———-

20.2.85 defun FromTo

This is called from parameter list of nc message functions

— **defun FromTo 0** —

```
(defun |FromTo| (pos1 pos2) (list 'fromto pos1 pos2))
```

———-

Chapter 21

The Interpreter Syntax

21.1 syntax assignment

— assignment.help —

```
Immediate, Delayed, and Multiple Assignment

=====================================================================
Immediate Assignment
=====================================================================

A variable in Axiom refers to a value. A variable has a name beginning
with an uppercase or lowercase alphabetic character, "%", or "!".
Successive characters (if any) can be any of the above, digits, or "?".
Case is distinguished. The following are all examples of valid, distinct
variable names:

  a       tooBig?     a1B2c3%!?
  A       %j          numberOfPoints
  beta6   %J          numberofpoints

The ":=" operator is the immediate assignment operator. Use it to
associate a value with a variable. The syntax for immediate assignment
for a single variable is:

   variable := expression

The value returned by an immediate assignment is the value of expression.

  a := 1
    1
            Type: PositiveInteger

The right-hand side of the expression is evaluated, yielding 1. The value
is then assigned to a.
```

```
  b := a
    1
            Type: PositiveInteger
```

The right-hand side of the expression is evaluated, yieldig 1. This value
is then assigned to b. Thus a and b both have the value 1 after the sequence
of assignments.

```
  a := 2
    2
            Type: PositiveInteger
```

What is the value of b if a is assigned the value 2?

```
  b
    1
            Type: PositiveInteger
```

The value of b is left unchanged.

This is what we mean when we say this kind of assignment is immediate.
The variable b has no dependency on a after the initial assignment. This
is the usual notion of assignment in programming languages such as C,
Pascal, and Fortran.

```
========================================================================
Delayed Assignment
========================================================================
```

Axiom provides delayed assignment with "==". This implements a delayed
evaluation of the right-hand side and dependency checking. The syntax for
delayed assignment is

```
   variable == expression
```

The value returned by a delayed assignment is the unique value of Void.

```
  a == 1
            Type: Void

  b == a
            Type: Void
```

Using a and b as above, these are the corresponding delayed assignments.

```
  a
   Compiling body of rule a to compute value of type PositiveInteger
   1
            Type: PositiveInteger
```

The right-hand side of each delayed assignment is left unevaluated until
the variables on the left-hand sides are evaluated.

```
  b
```

```
   Compiling body of rule b to compute value of type PositiveInteger
   1
           Type: PositiveInteger

This gives the same results as before. But if we change a to 2

  a == 2
   Compiled code for a has been cleared.
   Compiled code for b has been cleared.
   1 old definition(s) deleted for function or rule a
           Type: Void

Then a evaluates to 2, as expected

  a
   Compiling body of rule a to compute value of type PositiveInteger
   2
           Type: PositiveInteger

but the value of b reflects the change to a

  b
   Compiling body of rule b to compute value of type PositiveInteger
   2
           Type: PositiveInteger

========================================================================
Multiple Immediate Assignments
========================================================================

It is possible to set several variables at the same time by using a
tuple of variables and a tuple of expressions. A tuple is a collection
of things separated by commas, often surrounded by parentheses. The
syntax for multiple immediate assignment is

 ( var1, var2, ..., varN ) := ( expr1, expr2, ..., exprN )

The value returned by an immediate assignment is the value of exprN.

 ( x, y ) := ( 1, 2 )
   2
           Type: PositiveInteger

This sets x to 1 and y to 2. Multiple immediate assignments are parallel
in the sense that the expressions on the right are all evaluated before
any assignments on the left are made. However, the order of evaluation
of these expressions is undefined.

 ( x, y ) := ( y, x )
   1
           Type: PositiveInteger

  x
   2
```

```
          Type: PositiveInteger

The variable x now has the previous value of y.

  y
   1
          Type: PositiveInteger

The variable y now has the previous value of x.

There is no syntactic form for multiple delayed assignments.
```

21.2 syntax blocks

— blocks.help —

```
=======================================================================
Blocks
=======================================================================

A block is a sequence of expressions evaluated in the order that they
appear, except as modified by control expressions such as leave, return,
iterate, and if-then-else constructions. The value of a block is the
value of the expression last evaluated in the block.

To leave a block early, use "=>". For example,

    i < 0 => x

The expression before the "=>" must evaluate to true or false. The
expression following the "=>" is the return value of the block.

A block can be constructed in two ways:

  1. the expressions can be separated by semicolons and the resulting
     expression surrounded by parentheses, and
  2. the expressions can be written on succeeding lines with each line
     indented the same number of spaces (which must be greater than zero).
     A block entered in this form is called a pile

Only the first form is available if you are entering expressions directly
to Axiom. Both forms are available in .input files. The syntax for a simple
block of expressions entered interactively is

  ( expression1 ; expression2 ; ... ; expressionN )

The value returned by a block is the value of an "=>" expression, or
expressionN if no "=>" is encountered.
```

```
In .input files, blocks can also be written in piles. The examples
given here are assumed to come from .input files.

  a :=
    i := gcd(234,672)
    i := 2*i**5 - i + 1
    1 / i

      1
    -----
    23323
              Type: Fraction Integer

In this example, we assign a rational number to a using a block consisting
of three expressions. This block is written as a pile. Each expression in
the pile has the same indentation, in this case two spaces to the right of
the first line.

  a := ( i := gcd(234,672); i := 2*i**5 - i + 1; 1 / i )

      1
    -----
    23323
              Type: Fraction Integer

Here is the same block written on one line. This is how you are required
to enter it at the input prompt.

  ( a := 1; b := 2; c := 3; [a,b,c] )
    [1,2,3]
              Type: List PositiveInteger

AAxiom gives you two ways of writing a block and the preferred way in
an .input file is to use a pile. Roughly speaking, a pile is a block
whose consituent expressions are indented the same amount. You begin a
pile by starting a new line for the first expression, indenting it to
the right of the previous line. You then enter the second expression on
a new line, vertically aligning it with the first line. And so on. If
you need to enter an inner pile, further indent its lines to the right
of the outer pile. Axiom knows where a pile ends. It ends when a subsequent
line is indented to the left of the pile or the end of the file.

Also See:
o )help if
o )help repeat
o )help while
o )help for
o )help suchthat
o )help parallel
o )help lists
```

———-

[1]

21.3 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.
^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
```

[1] "if" (21.6 p 610) "repeat" (21.10 p 616) "while" (38.1.2 p 1101) "for" (21.5 p 606) "suchthat" (21.11 p 620) "parallel" (21.9 p 614) "lists" (?? p ??)

21.4 syntax collection

— collection.help —

```
====================================================================
Collection -- Creating Lists and Streams with Iterators
====================================================================

All of the loop expressions which do not use the repeat leave or
iterate words can be used to create lists and streams. For example:

This creates a simple list of the integers from 1 to 10:

  list := [i for i in 1..10]
   [1,2,3,4,5,6,7,8,9,10]
                      Type: List PositiveInteger

Create a stream of the integers greater than or equal to 1:

  stream := [i for i in 1..]
   [1,2,3,4,5,6,7,...]
                      Type: Stream PositiveInteger

This is a list of the prime numbers between 1 and 10, inclusive:

  [i for i in 1..10 | prime? i]
   [2,3,5,7]
                      Type: List PositiveInteger

This is a stream of the prime integers greater than or equal to 1:

  [i for i in 1.. | prime? i]
   [2,3,5,7,11,13,17,...]
                      Type: Stream PositiveInteger

This is a list of the integers between 1 and 10, inclusive, whose
squares are less than 700:

  [i for i in 1..10 while i*i < 700]
   [1,2,3,4,5,6,7,8,9,10]
                      Type: List PositiveInteger

This is a stream of the integers greater than or equal to 1 whose
squares are less than 700:

  [i for i in 1.. while i*i < 700]
   [1,2,3,4,5,6,7,...]
                      Type: Stream PositiveInteger
```

```
The general syntax of a collection is

  [ collectExpression iterator1 iterator2 ... iteratorN ]

where each iterator is either a for or a while clause. The loop
terminates immedidately when the end test of any iterator succeeds
or when a return expression is evaluated in collectExpression. The
value returned by the collection is either a list or a stream of
elements, one for each iteration of the collectExpression.

Be careful when you use while to create a stream. By default Axiom
tries to compute and display the first ten elements of a stream. If
the while condition is not satisfied quickly, Axiom can spend a long
(potentially infinite) time trying to compute the elements. Use

  )set streams calculate

to change the defaults to something else. This also affects the number
of terms computed and displayed for power series. For the purposes of
these examples we have use this system command to display fewer than
ten terms.
```

———-

21.5 syntax for

— for.help —

```
========================================================================
for loops
========================================================================

Axiom provide the for and in keywords in repeat loops, allowing you
to integrate across all elements of a list, or to have a variable take
on integral values from a lower bound to an upper bound. We shall refer
to these modifying clauses of repeat loops as for clauses. These clauses
can be present in addition to while clauses (See )help while). As with
all other types of repeat loops, leave (see )help leave) can be used to
prematurely terminate evaluation of the loop.

The syntax for a simple loop using for is

  for iterator repeat loopbody

The iterator has several forms. Each form has an end test which is
evaluted before loopbody is evaluated. A for loop terminates immediately
when the end test succeeds (evaluates to true) or when a leave or return
expression is evaluated in loopbody. The value returned by the loop is
the unique value of Void.
```

```
========================================================================
for i in n..m repeat
========================================================================

If for is followed by a variable name, the in keyword and then an integer
segment of the form n..m, the end test for this loop is the predicate
i > m. The body of the loop is evaluated m-n+1 times if this number is
greater than 0. If this number is less than or equal to 0, the loop body
is not evaluated at all.

The variable i has the value n, n+1, ..., m for successive iterations
of the loop body. The loop variable is a local variable within the loop
body. Its value is not available outside the loop body and its value and
type within the loop body completely mask any outer definition of a
variable with the same name.

  for i in 10..12 repeat output(i**3)
   1000
   1331
   1728
                      Type: Void

The loop prints the values of 10^3, 11^3, and 12^3.

  a := [1,2,3]
   [1,2,3]
                      Type: List PositiveInteger

  for i in 1..#a repeat output(a.i)
   1
   2
   3
                      Type: Void

Iterate across this list using "." to access the elements of a list
and the # operation to count its elements.

This type of iteration is applicable to anything that uses ".". You
can also use it with functions that use indices to extract elements.

   m := matrix [ [1,2],[4,3],[9,0] ]
    +-    -+
    | 1  2 |
    | 4  3 |
    | 9  0 |
    +-    -+
                      Type: Matrix Integer

Define m to be a matrix.

   for i in 1..nrows(m) repeat output row(m.i)
    [1,2]
    [4,3]
    [9,0]
```

```
                    Type: Void

Display the rows of m.

You can iterate with for-loops.

   for i in 1..5 repeat
     if odd?(i) then iterate
     output(i)
    2
    4
                    Type: Void

Display the even integers in a segment.

======================================================================
for i in n..m by s repeat
======================================================================

By default, the difference between values taken on by a variable in
loops such as

  for i in n..m repeat ...

is 1. It is possible to supply another, possibly negative, step value
by using the by keyword along with for and in. Like the upper and lower
bounds, the step value following the by keyword must be an integer. Note
that the loop

  for i in 1..2 by 0 repeat output(i)

will not terminate by itself, as the step value does not change the
index from its initial value of 1.

  for i in 1..5 by 2 repeat output(i)
   1
   3
   5
                    Type: Void

This expression displays the odd integers between two bounds.

  for i in 5..1 by -2 repeat output(i)
   5
   3
   1
                    Type: Void

Use this to display the numbers in reverse order.

======================================================================
for i in n.. repeat
======================================================================
```

```
If the value after the ".." is omitted, the loop has no end test. A
potentially infinite loop is thus created. The variable is given the
successive values n, n+1, n+2, ... and the loop is terminated only
if a leave or return expression is evaluated in the loop body. However,
you may also add some other modifying clause on the repeat, for example,
a while clause, to stop the loop.

  for i in 15.. while not prime?(i) repeat output(i)
   15
   16
                      Type: Void

This loop displays the integers greater than or equal to 15 and less
than the first prime number greater than 15.

========================================================================
for x in l repeat
========================================================================

Another variant of the for loop has the form:

  for x in list repeat loopbody

This form is used when you want to iterate directly over the elements
of a list. In this form of the for loop, the variable x takes on the
value of each successive element in l. The end test is most simply
stated in English: "are there no more x in l?"

  l := [0, -5, 3]
   [0, -5, 3]
                      Type: List Integer

  for x in l repeat output(x)
   0
   -5
   3
                      Type: Void

This displays all of the elements of the list l, one per line.

Since the list constructing expression

  expand [n..m]

creates the list

  [n, n+1, ..., m]

you might be tempted to think that the loops

  for i in n..m repeat output(i)

and
```

```
  for x in expand [n..m] repeat output(x)

are equivalent. The second form first creates the expanded list
(no matter how large it might be) and then does the iteration. The
first form potentially runs in much less space, as the index variable
i is simply incremented once per loop and the list is not actually
created. Using the first form is much more efficient.

Of course, sometimes you really want to iterate across a specific list.
This displays each of the factors of 2400000:

  for f in factors(factor(2400000)) repeat output(f)
   [factor= 2, exponent= 8]
   [factor= 3, exponent= 1]
   [factor= 5, exponent= 5]
                      Type: Void
```

21.6 syntax if

— if.help —

```
========================================================================
If-then-else
========================================================================

Like many other programming languages, Axiom uses the three keywords
if, then, and else to form conditional expressions. The else part of
the conditional is optional. The expression between the if and then
keywords is a predicate: an expression that evaluates to or is
convertible to either true or false, that is, a Boolean.

The syntax for conditional expressions is

   if predicate then expression1 else expression2

where the "else expression2" part is optional. The value returned from
a conditional expression is expression1 if the predicate evaluates to
true and expression2 otherwise. If no else clause is given, the value
is always the unique value of Void.

An if-then-else expression always returns a value. If the else clause
is missing then the entire expression returns the unique value of Void.
If both clauses are present, the type of the value returned by if is
obtained by resolving the types of the values of the two clauses.

The predicate must evaluate to, or be convertible to, an object of type
Boolean: true or false. By default, the equal sign "=" creates an equation.

   x + 1 = y
```

```
    x + 1 = y
                Type: Equation Polynomial Integer
```

This is an equation, not a boolean condition. In particular, it is an object of type Equation Polynomial Integer.

However, for predicates in if expressions, Axiom places a default target type of Boolean on the predicate and equality testing is performed. Thus you need not qualify the "=" in any way. In other contexts you may need to tell Axiom that you want to test for equality rather than create an equation. In these cases, use "@" and a target type of Boolean.

The compound symbol meaning "not equal" in Axiom is "~=". This can be used directly without a package call or a target specification. The expression "a ~= b" is directly translated to "not(a = b)".

Many other functions have return values of type Boolean. These include <, <=, >, >=, ~=, and member?. By convention, operations with names ending in "?" return Boolean values.

The usual rules for piles are suspended for conditional expressions. In .input files, the then and else keywords can begin in the same column as the corresponding if by may also appear to the right. Each of the following styles of writing if-then-else expressions is acceptable:

```
  if i>0 then output("positive") else output("nonpositive")

  if i>0 then output("positive")
    else output("nonpositive")

  if i>0 then output("positive")
  else output("nonpositive")

  if i>0
  then output("positive")
  else output("nonpositive")

  if i>0
    then output("positive")
    else output("nonpositive")
```

A block can follow the then or else keywords. In the following two assignments to a, the then and else clauses each are followed by two line piles. The value returned in each is the value of the second line.

```
  a :=
    if i > 0 then
      j := sin(i * pi())
      exp(j + 1/j)
    else
      j := cos(i * 0.5 * pi())
      log(abs(j)**5 + i)
```

```
  a :=
    if i > 0
      then
        j := sin(i * pi())
        exp(j + 1/j)
      else
        j := cos(i * 0.5 * pi())
        log(abs(j)**5 + i)
```

These are both equivalent to the following:

```
  a :=
    if i > 0 then (j := sin(i * pi()); exp(j + 1/j))
    else (j := cos(i * 0.5 * pi()); log(abs(j)**5 + i))
```

———-

21.7 syntax iterate

— **iterate.help** —

```
====================================================================
iterate in loops
====================================================================

Axiom provides an iterate expression that skips over the remainder
of a loop body and starts the next loop execution. We first initialize
a counter.

  i := 0
   0
                      Type: NonNegativeInteger

Display the even integers from 2 to 5:

  repeat
    i := i + 1
    if i > 5 then leave
    if odd?(i) then iterate
    output(i)
   2
   4
                      Type: Void
```

———-

21.8 syntax leave

— leave.help —

```
==========================================================================
leave in loops
==========================================================================

The leave keyword is often more useful in terminating a loop. A
leave causes control to transfer to the expression immediately following
the loop. As loops always return the unique value of Void, you cannot
return a value with leave. That is, leave takes no argument.

  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then leave
      i := i + 1
    i
                        Type: Void

This example is a modification of the last example in the previous
section. Instead of using return we'll use leave.

  f()
   7
                        Type: PositiveInteger

The loop terminates when factorial(i) gets big enough. The last line
of the function evaluates to the corresponding "good" value of i
and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Lets
consider a loop within a loop, that is, a loop with a nested loop.
First, we initialize two counter variables.

  (i,j) := (1,1)
   1
                        Type: PositiveInteger

  repeat
    repeat
      if (i + j) > 10 then leave
      j := j + 1
    if (i + j) > 10 then leave
    i := i + 1
                        Type: Void

Nested loops must have multiple leave expressions at the appropriate
nesting level. How would you rewrite this so (i + j) > 10 is only
evaluated once?

==========================================================================
```

```
leave vs => in loop bodies
========================================================================

Compare the following two loops:

  i := 1                      i := 1
  repeat                      repeat
    i := i + 1                  i := i + 1
    i > 3 => i                  if i > 3 then leave
    output(i)                   output(i)

In the example on the left, the values 2 and 3 for i are displayed but
then the "=>" does not allow control to reach the call to output again.
The loop will not terminate until you run out of space or interrupt the
execution. The variable i will continue to be incremented because the
"=>" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed,
and both the block and the loop will terminate. This is one of the reasons
why both "=>" and leave are provided. Using a while clase with the "=>"
lets you simulate the action of leave.
```

———-

21.9 syntax parallel

— **parallel.help** —

```
========================================================================
parallel iteration
========================================================================

Sometimes you want to iterate across two lists in parallel, or perhaps
you want to traverse a list while incrementing a variable.

The general syntax of a repeat loop is

 iterator1, iterator2, ..., iteratorN repeat loopbody

where each iterator is either a for or a while clause. The loop
terminates immediately when the end test of any iterator succeeds or
when a leave or return expression is evaluated in loopbody. The value
returned by the loop is the unique value of Void.

  l := [1,3,5,7]
   [1,3,5,7]
                      Type: List PositiveInteger

  m := [100,200]
   [100,200]
                      Type: List PositiveInteger
```

```
  sum := 0
   0
                        Type: NonNegativeInteger
```

Here we write a loop to iterate across two lists, computing the sum
of the pairwise product of the elements:

```
  for x in l for y in m repeat
    sum := sum + x*y
                        Type: Void
```

The last two elements of l are not used in the calculation because
m has two fewer elements than l.

```
  sum
   700
                        Type: NonNegativeInteger
```

This is the "dot product".

Next we write a loop to compute the sum of the products of the loop
elements with their positions in the loop.

```
  l := [2,3,5,7,11,13,17,19,23,29,31,37]
   [2,3,5,7,11,13,17,19,23,29,31,37]
                        Type: List PositiveInteger

  sum := 0
   0
                        Type: NonNegativeInteger

  for i in 0.. for x in l repeat sum := i * x
                        Type: Void
```

Here looping stops when the list l is exhaused, even though the
for i in 0.. specifies no terminating condition.

```
  sum
   407
                        Type: NonNegativeInteger
```

When "|" is used to qualify any of the for clauses in a parallel
iteration, the variables in the predicates can be from an outer
scope or from a for clause in or to the left of the modified clause.

This is correct:

```
  for i in 1..10 repeat
    for j in 200..300 | ood? (i+j) repeat
      output [i,j]
```

But this is not correct. The variable j has not been defined outside
the inner loop:

```
  for i in 1..01 | odd? (i+j) repeat -- wrong, j not defined
    for j in 200..300 repeat
      output [i,j]

It is possible to mix several of repeat modifying clauses on a loop:

  for i in 1..10
    for j in 151..160 | odd? j
      while i + j < 160 repeat
        output [i,j]
   [1,151]
   [3,153]
                      Type: Void

Here are useful rules for composing loop expressions:

 1. while predicates can only refer to variables that are global (or
    in an outer scope) or that are defined in for clauses to the left
    of the predicate.
 2. A "such that" predicate (somthing following "|") must directly
    follow a for clause and can only refer to variables that are
    global (or in an outer scope) or defined in the modified for clause
    or any for clause to the left.
```

21.10 syntax repeat

— repeat.help —

```
========================================================================
Repeat Loops
========================================================================

A loop is an expression that contains another expression, called the loop
body, which is to be evaluated zero or more times. All loops contain the
repeat keyword and return the unique value of Void. Loops can contain
inner loops to any depth.

The most basic loop is of the form

  repeat loopbody

Unless loopbody contains a leave or return expression, the loop repeats
foreer. The value returned by the loop is the unique value of Void.

Axiom tries to determine completely the type of every object in a loop
and then to translate the loop body to Lisp or even to machine code. This
translation is called compilation.
```

```
If Axiom decides that it cannot compile the loop, it issues a message
stating the problem and then the following message:

  We will attemp to step through and interpret the code

It is still possible that Axiom can evalute the loop but in interpret-code
mode.

========================================================================
Return in Loops
========================================================================

A return expression is used to exit a function with a particular value.
In particular, if a return is in a loop within the function, the loop
is terminated whenever the return is evaluated.

  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then return i
      i := i + 1
                      Type: Void

  f()
                      Type: Void

When factorial(i) is big enough, control passes from inside the loop
all the way outside the function, returning the value of i (so we think).
What went wrong? Isn't it obvious that this function should return an
integer? Well, Axiom makes no attempt to analyze the structure of a
loop to determine if it always returns a value because, in general, this
is impossible. So Axiom has this simple rule: the type of the function is
determined by the type of its body, in this case a block. The normal value
of a block is the value of its last expression, in this case, a loop. And
the value of every loop is the unique value of Void. So the return type
of f is Void.

There are two ways to fix this. The best way is for you to tell Axiom
what the return type of f is. You do this by giving f a declaration

   f:() -> Integer

prior to calling for its value. This tells Axiom "trust me -- an integer
is returned". Another way is to add a dummy expression as follows.

  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then return i
      i := i + 1
    0
                      Type: Void

Note that the dummy expression will never be evaluated but it is the
```

```
last expression in the function and will determine the return type.

  f()
   7
                      Type: PositiveInteger

========================================================================
leave in loops
========================================================================

The leave keyword is often more useful in terminating a loop. A
leave causes control to transfer to the expression immediately following
the loop. As loops always return the unique value of Void, you cannot
return a value with leave. That is, leave takes no argument.

  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then leave
      i := i + 1
    i
                      Type: Void

This example is a modification of the last example in the previous
section. Instead of using return we'll use leave.

  f()
   7
                      Type: PositiveInteger

The loop terminates when factorial(i) gets big enough. The last line
of the function evaluates to the corresponding "good" value of i
and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Lets
consider a loop within a loop, that is, a loop with a nested loop.
First, we initialize two counter variables.

  (i,j) := (1,1)
   1
                      Type: PositiveInteger

  repeat
    repeat
      if (i + j) > 10 then leave
      j := j + 1
    if (i + j) > 10 then leave
    i := i + 1
                      Type: Void

Nested loops must have multiple leave expressions at the appropriate
nesting level. How would you rewrite this so (i + j) > 10 is only
evaluated once?
```

```
========================================================================
leave vs => in loop bodies
========================================================================

Compare the following two loops:

  i := 1                      i := 1
  repeat                      repeat
    i := i + 1                  i := i + 1
    i > 3 => i                  if i > 3 then leave
    output(i)                   output(i)

In the example on the left, the values 2 and 3 for i are displayed but
then the "=>" does not allow control to reach the call to output again.
The loop will not terminate until you run out of space or interrupt the
execution. The variable i will continue to be incremented because the
"=>" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed,
and both the block and the loop will terminate. This is one of the reasons
why both "=>" and leave are provided. Using a while clase with the "=>"
lets you simulate the action of leave.

========================================================================
iterate in loops
========================================================================

Axiom provides an iterate expression that skips over the remainder
of a loop body and starts the next loop execution. We first initialize
a counter.

  i := 0
   0
                      Type: NonNegativeInteger

Display the even integers from 2 to 5:

  repeat
    i := i + 1
    if i > 5 then leave
    if odd?(i) then iterate
    output(i)
   2
   4
                      Type: Void

Also See:
o )help blocks
o )help if
o )help while
o )help for
o )help suchthat
o )help parallel
o )help lists
```

———-

[2]

21.11 syntax suchthat

— **suchthat.help** —

```
====================================================================
Such that predicates
====================================================================

A for loop can be followed by a "|" and then a predicate. The predicate
qualifies the use of the values from the iterator that follows the for.
Think of the vertical bar "|" as the phrase "such that".

  for n in 0..4 | odd? n repeat output n
   1
   3
                      Type: Void

This loop expression prints out the integers n in the given segment
such that n is odd.

A for loop can also be written

  for iterator | predicate repeat loopbody

which is equivalent to:

  for iterator repeat if predicate then loopbody else iterate

The predicate need not refer only to the variable in the for clause.
Any variable in an outer scope can be part of the predicate.

  for i in 1..50 repeat
    for j in 1..50 | factorial(i+j) < 25 repeat
      output [i,j]
   [1,1]
   [1,2]
   [1,3]
   [2,1]
   [2,2]
   [3,1]
                      Type: Void
```

———-

[2] "blocks" (21.2 p 602) "if" (21.6 p 610) "while" (38.1.2 p 1101) "for" (21.5 p 606) "suchthat" (21.11 p 620) "parallel" (21.9 p 614) "lists" (**??** p **??**)

21.12 syntax syntax

— **syntax.help** —

```
The Axiom Interactive Language has the following features documented here.

More information is available by typing

  )help feature

where feature is one of:

  assignment -- Immediate and delayed assignments
  blocks     -- Blocks of expressions
  collection -- creating lists with iterators
  for        -- for loops
  if         -- If-then-else statements
  iterate    -- using iterate in loops
  leave      -- using leave in loops
  parallel   -- parallel iterations
  repeat     -- repeat loops
  suchthat   -- suchthat predicates
  while      -- while loops
```

———-

21.13 syntax while

— **while.help** —

```
========================================================================
while loops
========================================================================

The repeat in a loop can be modified by adding one or more while
clauses. Each clause contains a predicate immediately following the
while keyword. The predicate is tested before the evaluation of the
body of the loop. The loop body is evaluated whenever the predicate
in a while clause is true.

The syntax for a simple loop using while is

  while predicate repeat loopbody

The predicate is evaluated before loopbody is evaluated. A while loop
terminates immediately when predicate evaluates to false or when a
leave or return expression is evaluted. See )help repeat for more
information on leave and return.

Here is a simple example of using while in a loop. We first initialize
```

```
the counter.

  i := 1
   1
                      Type: PositiveInteger

  while i < 1 repeat
    output "hello"
    i := i + 1
                      Type: Void

The steps involved in computing this example are
 (1) set i to 1
 (2) test the condition i < 1 and determine that it is not true
 (3) do not evaluate the loop body and therefore do not display "hello"

  (x, y) := (1, 1)
   1
                      Type: PositiveInteger

If you have multiple predicates to be tested use the logical and
operation to separate them. Axiom evaluates these predicates from
left to right.

  while x < 4 and y < 10 repeat
    output [x,y]
    x := x + 1
    y := y + 2
   [1,1]
   [2,3]
   [3,5]
                      Type: Void


A leave expression can be included in a loop body to terminate a loop
even if the predicate in any while clauses are not false.

  (x, y) := (1, 1)
   1
                      Type: PositiveInteger

  while x < 4 and y < 10 repeat
    if x + y > 7 then leave
    output [x,y]
    x := x + 1
    y := y + 2
   [1,1]
   [2,3]
                      Type: Void
```

Chapter 22

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)
```

22.0.1 defun Construct a leaf token

The tokConstruct function is a constructer and selectors for leaf tokens. A leaf token looks like [head, token, position] where head is either an id or (id . alist)

[ifcar p**??**]
[pfNoPosition? p624]
[ncPutQ p628]

— **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))))
```

———-

22.0.2 defun Return a part of a node

[ifcar p??]

— **defun pfAbSynOp** —

```
(defun |pfAbSynOp| (form)
 (let (hd)
  (setq hd (car form))
  (or (ifcar hd) hd)))
```

————-

22.0.3 defun Compare a part of a node

[eqcar p??]

— **defun pfAbSynOp?** —

```
(defun |pfAbSynOp?| (form op)
 (let (hd)
  (setq hd (car form))
  (or (eq hd op) (eqcar hd op))))
```

————-

22.0.4 defun pfNoPosition?

[poNoPosition? p624]

— **defun pfNoPosition?** —

```
(defun |pfNoPosition?| (pos)
 (|poNoPosition?| pos))
```

————-

22.0.5 defun poNoPosition?

[eqcar p??]

— **defun poNoPosition? 0** —

```
(defun |poNoPosition?| (pos)
 (eqcar pos '|noposition|))
```

————-

22.0.6 defun tokType

[ncTag p627]

— **defun tokType** —

```
(defun |tokType| (x) (|ncTag| x))
```

———-

22.0.7 defun tokPart

— **defun tokPart 0** —

```
(defun |tokPart| (x) (cdr x))
```

———-

22.0.8 defun tokPosn

[qassq p**??**]
[ncAlist p627]
[pfNoPosition p625]

— **defun tokPosn** —

```
(defun |tokPosn| (x)
 (let (a)
  (setq a (qassq '|posn| (|ncAlist| x)))
  (cond
   (a (cdr a))
   (t (|pfNoPosition|)))))
```

———-

22.0.9 defun pfNoPosition

[poNoPosition p625]

— **defun pfNoPosition** —

```
(defun |pfNoPosition| () (|poNoPosition|))
```

———-

22.0.10 defun poNoPosition

[$nopos p288]

— defun poNoPosition 0 —

```
(defun |poNoPosition| ()
 (declare (special |$nopos|))
 |$nopos|)
```

———-

Chapter 23

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.

23.0.11 defun ncTag

Pick off the tag [ncBug p587]
[qcar p??]
[identp p1107]

— **defun ncTag** —

```
(defun |ncTag| (x)
 (cond
  ((null (consp x)) (|ncBug| "bad object" nil))
  (t
   (setq x (qcar x))
   (cond
    ((identp x) x)
    ((null (consp x)) (|ncBug| "bad object" nil))
    (t (qcar x))))))
```

———-

23.0.12 defun ncAlist

Pick off the property list [ncBug p587]
[qcar p??]
[identp p1107]
[qcdr p??]

— **defun ncAlist** —

```
(defun |ncAlist| (x)
 (cond
  ((null (consp x)) (|ncBug| "bad object" nil))
```

```
  (t
   (setq x (qcar x))
   (cond
    ((identp x) nil)
    ((null (consp x)) (|ncBug| "bad object" nil))
    (t (qcdr x))))))
```

———-

23.0.13 defun ncEltQ

Get the entry for key k on x's association list

[qassq p??]
[ncAlist p627]
[ncBug p587]

— **defun ncEltQ** —

```
(defun |ncEltQ| (x k)
 (let (r)
  (setq r (qassq k (|ncAlist| x)))
  (cond
   ((null r) (|ncBug| "Association list search failed on %1" (list k)))
   (t (cdr r)))))
```

———-

23.0.14 defun ncPutQ

```
;-- Put (k . v) on the association list of x and return v
;-- case1: ncPutQ(x,k,v) where k is a key (an identifier), v a value
;--        put the pair (k . v) on the association list of x and return v
;-- case2: ncPutQ(x,k,v) where k is a list of keys, v a list of values
;--        equivalent to [ncPutQ(x,key,val) for key in k for val in v]
;ncPutQ(x,k,v) ==
;   LISTP k =>
;      for key in k for val in v repeat ncPutQ(x,key,val)
;      v
;   r := QASSQ(k,ncAlist x)
;   if NULL r then
;      r := CONS( CONS(k,v), ncAlist x)
;      RPLACA(x,CONS(ncTag x,r))
;   else
;      RPLACD(r,v)
;   v
```

[qassq p??]
[ncAlist p627]
[ncTag p627]

— **defun ncPutQ** —

```
(defun |ncPutQ| (x k v)
 (let (r)
  (cond
   ((listp k)
    ((lambda (Var1 key Var2 val)
     (loop
      (cond
       ((or (atom Var1)
            (progn (setq key (car Var1)) nil)
            (atom Var2)
            (progn (setq val (car Var2)) nil))
          (return nil))
        (t
          (|ncPutQ| x key val)))
      (setq Var1 (cdr Var1))
      (setq Var2 (cdr Var2))))
     k nil v nil)
    v)
   (t
    (setq r (qassq k (|ncAlist| x)))
    (cond
     ((null r)
      (setq r (cons (cons k v) (|ncAlist| x)))
      (rplaca x (cons (|ncTag| x) r)))
     (t
      (rplacd r v)))
    v))))
```

———-

23.0.15 Special Category Names

23.0.16 defvar $EmptyMode

The CONTAINED predicate is used to walk internal structures such as modemaps to see if the X object occurs within Y. One particular use is in a function called isPartialMode to decide if a modemap is only partially complete. If this is true then the modemap will contain the constant `$EmptyMode`. So the call ends up being CONTAINED `$EmptyMode` Y.

— **initvars** —

```
(defvar |$EmptyMode| '|$EmptyMode|)
```

———-

23.0.17 defvar $AnonymousFunction

— **initvars** —

```
(defvar |$AnonymousFunction| '(|AnonymousFunction|))
```

———-

23.0.18 defvar $Any

— initvars —

```
(defvar |$Any|   '(|Any|))
```

———-

23.0.19 defvar $BFtag

— initvars —

```
(defvar |$BFtag| '|:BF:|)
```

———-

23.0.20 defvar $Boolean

— initvars —

```
(defvar |$Boolean| '(|Boolean|))
```

———-

23.0.21 defvar $Category

— initvars —

```
(defvar |$Category| '(|Category|))
```

———-

23.0.22 defvar $Domain

— initvars —

```
(defvar |$Domain| '(|Domain|))
```

———-

23.0.23 defvar $Exit

— initvars —

```
(defvar |$Exit|  '(|Exit|))
```

———-

23.0.24 defvar $Expression

— initvars —

```
(defvar |$Expression| '(|OutputForm|))
```

———-

23.0.25 defvar $OutputForm

— initvars —

```
(defvar |$OutputForm| '(|OutputForm|))
```

———-

23.0.26 defvar $BigFloat

— initvars —

```
(defvar |$BigFloat| '(|Float|))
```

———-

23.0.27 defvar $Float

— initvars —

```
(defvar |$Float| '(|Float|))
```

———-

23.0.28 defvar $DoubleFloat

— initvars —

```
(defvar |$DoubleFloat| '(|DoubleFloat|))
```

———-

23.0.29 defvar $FontTable

— **initvars** —

```
(defvar |$FontTable| '(|FontTable|))
```

———-

23.0.30 defvar $Integer

— **initvars** —

```
(defvar |$Integer| '(|Integer|))
```

———-

23.0.31 defvar $ComplexInteger

— **initvars** —

```
(defvar |$ComplexInteger| (LIST '|Complex| |$Integer|))
```

———-

23.0.32 defvar $Mode

— **initvars** —

```
(defvar |$Mode| '(|Mode|))
```

———-

23.0.33 defvar $NegativeInteger

— **initvars** —

```
(defvar |$NegativeInteger| '(|NegativeInteger|))
```

———-

23.0.34 defvar $NonNegativeInteger

— **initvars** —

```
(defvar |$NonNegativeInteger| '(|NonNegativeInteger|))
```

———-

23.0.35 defvar $NonPositiveInteger

— **initvars** —

```
(defvar |$NonPositiveInteger| '(|NonPositiveInteger|))
```

———-

23.0.36 defvar $PositiveInteger

— **initvars** —

```
(defvar |$PositiveInteger| '(|PositiveInteger|))
```

———-

23.0.37 defvar $RationalNumber

— **initvars** —

```
(defvar |$RationalNumber| '(|Fraction| (|Integer|)))
```

———-

23.0.38 defvar $String

— **initvars** —

```
(defvar |$String| '(|String|))
```

———-

23.0.39 defvar $StringCategory

— **initvars** —

```
(defvar |$StringCategory| '(|StringCategory|))
```

———-

23.0.40 defvar $Symbol

— **initvars** —

```
(defvar |$Symbol| '(|Symbol|))
```

———-

23.0.41 defvar $Void

— **initvars** —

```
(defvar |$Void|  '(|Void|))
```

———-

23.0.42 defvar $QuotientField

— **initvars** —

```
(defvar |$QuotientField| '|Fraction|)
```

———-

23.0.43 defvar $FunctionalExpression

— **initvars** —

```
(defvar |$FunctionalExpression| '|Expression|)
```

———-

23.0.44 defvar $defaultFunctionTargets

— **initvars** —

```
(defvar |$defaultFunctionTargets| '(()))
```

———-

;; Old names

23.0.45 defvar $SmallInteger

— **initvars** —

```
(defvar |$SmallInteger| '(|SingleInteger|))
```

———-

;; New Names

23.0.46 defvar $SingleFloat

— **initvars** —

```
(defvar |$SingleFloat| '(|SingleFloat|))
```

———-

23.0.47 defvar $DoubleFloat

— **initvars** —

```
(defvar |$DoubleFloat| '(|DoubleFloat|))
```

———-

23.0.48 defvar $SingleInteger

— **initvars** —

```
(defvar |$SingleInteger| '(|SingleInteger|))
```

———-

Chapter 24

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
```

24.0.49 defun ofCategory

[identp p1107]
[ofCategory p637]
[hasCaty p638]
[$Subst p??]
[$hope p??]

— **defun ofCategory** —

```
(defun |ofCategory| (dom cat)
 (let (|$Subst| |$hope|)
 (declare (special |$Subst| |$hope|))
  (cond
   ((identp dom) nil)
   ((and (listp cat) (eq (car cat) '|Join|))
    (every #'(lambda (c) (|ofCategory| dom c)) (cdr cat)))
   (t (not (eq (|hasCaty| dom cat nil) '|failed|))))))
```

———-

24.0.50 defun isPartialMode

The isPartialMode function tests whether m contains `$EmptyMode`. The constant `$EmptyMode` evaluates to `|$EmptyMode|`. This constant is inserted in a modemap during compile time if the modemap is not yet complete.

[contained p**??**]
[$EmptyMode p629]

— **defun isPartialMode** —

```
(defun |isPartialMode| (m)
 (declare (special |$EmptyMode|))
 (contained |$EmptyMode| m))
```

———-

24.0.51 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 p644]
[subCopy p**??**]
[constructSubst p651]
[hasSig p640]
[hasAtt p641]
[hasCat p**??**]
[opOf p**??**]
[mkDomPvar p650]
[domArg p640]
[augmentSub p**??**]
[domArg2 p640]
[unifyStruct p645]
[hasCaty1 p648]
[$domPvar p308]

— **defun hasCaty** —

```
(defun |hasCaty| (d cat sl)
 (let (x y S z cond sp dom zp s1 ncond i)
 (declare (special |$domPvar|))
  (cond
   ((and (consp cat) (eq (qcar cat) 'category) (consp (qcdr cat)))
     (|hasAttSig| d (|subCopy| (qcddr cat) (|constructSubst| d)) sl))
   ((and (consp cat) (eq (qcar cat) 'signature) (consp (qcdr cat))
         (consp (qcddr cat)) (eq (qcdddr cat) nil))
```

```
  (|hasSig| d (qcadr cat) (|subCopy| (qcaddr cat) (|constructSubst| d)) sl))
((and (consp cat) (eq (qcar cat) 'attribute)
      (consp (qcdr cat)) (eq (qcddr cat) nil))
  (|hasAtt| d (|subCopy| (qcadr cat) (|constructSubst| d)) sl))
((setq x (|hasCat| (|opOf| d) (|opOf| cat)))
 (cond
  ((setq y (ifcdr cat))
    (setq s (|constructSubst| d))
    (do ((next x (cdr next)) (endtest nil (null (eq s1 '|failed|))))
        ((or (atom next) endtest) nil)
      (setq z (caar next))
      (setq cond (cdar next))
      (setq sp
       (loop for item in s
        collect (cons (car item) (|mkDomPvar| (car item) (cdr item) z y))))
      (when |$domPvar|
        (setq i -1)
        (setq dom
         (cons (car d)
          (loop for arg in (rest d)
           collect (|domArg| arg (incf i) z y))))
        (setq sl (|augmentSub| |$domPvar| dom (copy sl))))
      (setq zp
       (loop for a in z
        collect (|domArg2| a s sp)))
      (setq s1 (|unifyStruct| y zp (copy sl)))
      (cond
       ((null (eq s1 '|failed|))
         (setq s1
          (cond
           ((atom cond) s1)
           (t
            (setq ncond (|subCopy| cond s))
            (cond
             ((and (consp ncond) (eq (qcar ncond) '|has|)
                   (consp (qcdr ncond)) (equal (qcadr ncond) d)
                   (consp (qcddr ncond)) (eq (qcdddr ncond) nil)
                   (equal (qcaddr ncond) cat))
               '|failed|)
             (t (|hasCaty1| ncond s1)))))))
       (t nil)))
     s1)
  ((atom x) sl)
  (t
   (setq ncond (|subCopy| x (|constructSubst| d)))
   (cond
    ((and (consp ncond) (eq (qcar ncond) '|has|) (consp (qcdr ncond))
          (equal (qcadr ncond) d) (consp (qcddr ncond))
          (eq (qcdddr ncond) nil) (equal (qcaddr ncond) cat))
      '|failed|)
    (t (|hasCaty1| ncond sl))))))
(t '|failed|))))
```

———-

24.0.52 defun domArg

[$FormalMapVariableList p15]

— **defun domArg** —

```
(defun |domArg| (type i subs y)
 (let (p)
  (declare (special |$FormalMapVariableList|))
  (if (setq p (member (elt |$FormalMapVariableList| i) subs))
   (elt y (- (|#| subs) (|#| p)))
   type)))
```

———-

24.0.53 defun domArg2

[isSharpVar p96]
[subCopy p??]
[$domPvar p308]

— **defun domArg2** —

```
(defun |domArg2| (arg sl1 sl2)
 (declare (special |$domPvar|))
 (cond
  ((|isSharpVar| arg) (|subCopy| arg sl1))
  ((and (eq arg '$) |$domPvar|) |$domPvar|)
  (t (|subCopy| arg sl2))))
```

———-

24.0.54 defun hasSig

The function hasSig tests whether domain dom has function foo with signature sig under substitution sl. [constructor? p??]
[cnstructSubst p??]
[assq p1110]
[getOperationAlistFromLisplib p119]
[hasCate p650]
[subCopy p??]
[hasSigAnd p642]
[hasSigOr p643]
[keyedSystemError p??]
[unifyStruct p645]
[$domPvar p308]

— **defun hasSig** —

```
(defun |hasSig| (dom foo sig sl)
 (let (|$domPvar| fun s0 p x cond s)
 (declare (special |$domPvar|))
  (cond
   ((setq fun (|constructor?| (car dom)))
     (setq s0 (|constructSubst| dom))
     (cond
      ((setq p (assq foo (|getOperationAlistFromLisplib| (car dom))))
        (do ((next (cdr p) (cdr next))
             (endtest nil (null (eq s '|failed|))))
            ((or (atom next) endtest) nil)
          (setq x (caar next))
          (setq cond (caddar next))
          (setq s
           (cond
            ((atom cond) (copy sl))
            ((and (consp cond) (eq (qcar cond) '|has|)
                  (consp (qcdr cond)) (consp (qcddr cond))
                  (eq (qcdr (qcddr cond)) nil))
              (|hasCate| (|subCopy| (qcadr cond) s0)
                         (|subCopy| (qcaddr cond) s0)
                         (copy sl)))
            ((and (consp cond)
                  (or (eq (qcar cond) 'and) (eq (qcar cond) '|and|)))
              (|hasSigAnd| (qcdr cond) s0 sl))
            ((and (consp cond)
                  (or (eq (qcar cond) 'or) (eq (qcar cond) '|or|)))
              (|hasSigOr| (qcdr cond) s0 sl))
            (t
             (|keyedSystemError|
              "Unexpected error or improper call to system function %1: %2"
              (list "hasSig" "unexpected condition for signature")))))
          (unless (eq s '|failed|)
            (setq s (|unifyStruct| (|subCopy| x s0) sig s))))
        s)
      (t '|failed|)))
   (t '|failed|))))
```

———-

24.0.55 defun hasAtt

The hasAtt function tests whether dom has attribute att under sl needs s0 similar to hasSig.
[subCopy p**??**]
[getdatabase p1070]
[constructSubst p651]
[getInfovec p**??**]
[unifyStruct p645]
[hasCatExpression p644]
[$domPvar p308]

— **defun hasAtt** —

```
(defun |hasAtt| (dom att sl)
 (let (|$domPvar| fun atts u x cond s)
 (declare (special |$domPvar|))
  (cond
   ((setq fun (car dom))
    (cond
     ((setq atts
        (|subCopy| (getdatabase fun 'attributes) (|constructSubst| dom)))
       (cond
        ((consp (setq u (|getInfovec| (car dom))))
          (do ((next atts (cdr next))
               (endtest nil (null (eq s '|failed|))))
              ((or (atom next) endtest) nil)
            (setq x (caar next))
            (setq cond (cdar next))
            (setq s (|unifyStruct| x att (copy sl)))
            (cond
             ((and (null (atom cond)) (null (eq s '|failed|)))
               (setq s (|hasCatExpression| cond s)))))
         s)
        (t
         (do ((next atts (cdr next))
              (endtest nil (null (eq s '|failed|))))
             ((or (atom next) endtest) nil)
           (setq x (caar next))
           (setq cond (cadar next))
           (setq s (|unifyStruct| x att (copy sl)))
           (cond
            ((and (null (atom cond)) (null (eq s '|failed|)))
              (setq s (|hasCatExpression| cond s)))))
         s)))
     (t '|failed|)))
   (t '|failed|))))
```

———-

24.0.56 defun hasSigAnd

[hasCate p650]
[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|
        "Unexpected error or improper call to system function %1: %2"
        (list  "hasSigAnd" "unexpected condition for signature")))))
   (when (eq sa '|failed|) (setq dead t)))
  sa))
```

———-

24.0.57 defun hasSigOr

[hasCate p650]
[hasSigAnd p642]
[keyedSystemError p??]

— **defun hasSigOr** —

```
(defun |hasSigOr| (orCls s0 sl)
 (let (sa found)
  (setq sa '|failed|)
  (loop for cls in orCls
   until found
   do
    (setq sa
     (cond
      ((atom cls) (copy sl))
      ((and (consp cls) (eq (qcar cls) '|has|) (consp (qcdr cls))
            (consp (qcddr cls)) (eq (qcdddr cls) nil))
        (|hasCate| (|subCopy| (qcadr cls) s0)
                   (|subCopy| (qcaddr cls) s0)
                   (copy sl)))
      ((and (consp cls)
            (or (eq (qcar cls) 'and) (eq (qcar cls) '|and|)))
        (|hasSigAnd| (qcdr cls) s0 sl))
      (t
       (|keyedSystemError|
        "Unexpected error or improper call to system function %1: %2"
        (list  "hasSigOr" "unexpected condition for signature")))))
   (unless (eq sa '|failed|) (setq found t)))
  sa))
```

———-

24.0.58 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 p641]
[hasSig p640]
[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|
        "Unexpected error or improper call to system function %1: %2"
        (list  "hasAttSig" "unexpected form of unnamed category"))))))
  sl)
```

———-

24.0.59 defun hasCate1

[hasCate p650]
[$domPvar p308]

— **defun hasCate1** —

```
(defun |hasCate1| (dom cat sl domPvar)
 (let (|$domPvar|)
 (declare (special |$domPvar|))
  (setq |$domPvar| domPvar)
  (|hasCate| dom cat sl)))
```

———-

24.0.60 defun hasCatExpression

[hasCatExpression p644]
[hasCate p650]
[keyedSystemError p??]

— **defun hasCatExpression** —

```
(defun |hasCatExpression| (cond sl)
 (let (y)
  (cond
   ((and (consp cond) (eq (qcar cond) 'or))
    (when
     (let (result)
       (loop for x in (qcdr cond)
        do (setq result
             (or result
                 (not (eq (setq y (|hasCatExpression| x sl)) '|failed|)))))
        result)
     y))
   ((and (consp cond) (eq (qcar cond) 'and))
    (when
     (let ((result t))
       (loop for x in (qcdr cond)
        do (setq result
             (and result
                  (not (eq (setq sl (|hasCatExpression| x sl)) '|failed|)))))
       result)
     sl))
   ((and (consp cond) (eq (qcar cond) '|has|)
         (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
     (|hasCate| (qcadr cond) (qcaddr cond) sl))
   (t
    (|keyedSystemError|
     "Unexpected error or improper call to system function %1: %2"
     (list  "hasSig" "unexpected condition for attribute"))))))
```

———-

24.0.61 defun unifyStruct

[isPatternVar p648]
[unifyStructVar p646]
[unifyStruct p645]

— **defun unifyStruct** —

```
(defun |unifyStruct| (s1 s2 sl)
 (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
  (cond
   ((equal s1 s2) sl)
   (t
    (when (and (consp s1) (eq (qcar s1) '|:|)
               (consp (qcdr s1)) (consp (qcddr s1)) (eq (qcdddr s1) nil))
       (setq s1 (qcadr s1)))
    (when (and (consp s2) (eq (qcar s2) '|:|)
               (consp (qcdr s2)) (consp (qcddr s2)) (eq (qcdddr s2) nil))
       (setq s2 (qcadr s2)))
    (when (and (null (atom s1)) (eq (car s1) '|#|))
```

```
      (setq s1 (length (cadr s1))))
    (when (and (null (atom s2)) (eq (car s2) '|#|))
      (setq s2 (length (cadr s2))))
    (cond
      ((equal s1 s2) sl)
      ((|isPatternVar| s1) (|unifyStructVar| s1 s2 sl))
      ((|isPatternVar| s2) (|unifyStructVar| s2 s1 sl))
      ((or (atom s1) (atom s2)) '|failed|)
      (t
       (loop until (or (null s1) (null s2) (eq sl '|failed|))
        do
         (setq sl (|unifyStruct| (car s1) (car s2) sl))
         (setq s1 (cdr s1))
         (setq s2 (cdr s2)))
       (if (or s1 s2) '|failed| sl))))))
```

———-

24.0.62 defun unifyStructVar

The first argument is a pattern variable, which is not substituted by sl [contained p**??**]
[lassoc p**??**]
[unifyStruct p645]
[constructor? p**??**]
[subCopy p**??**]
[containsVars p647]
[canCoerce p**??**]
[resolveTT p**??**]
[isPatternVar p648]
[augmentSub p**??**]
[$domPvar p308]
[$Coerce p**??**]
[$Subst p**??**]
[$hope p**??**]

— **defun unifyStructVar** —

```
(defun |unifyStructVar| (v ss sl)
 (let (ps s1 s0 s ns0 ns1 s3)
  (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
   (cond
    ((contained v ss) '|failed|)
    (t
     (setq ps (lassoc ss sl))
     (setq s1 (if ps ps ss))
     (cond
      ((or (setq s0 (lassoc v sl)) (setq s0 (lassoc v |$Subst|)))
        (setq s (|unifyStruct| s0 s1 (copy sl)))
        (cond
          ((eq s '|failed|)
           (cond
            ((and |$Coerce| (null (atom s0)) (|constructor?| (car s0)))
```

```
            (cond
             ((or (|containsVars| s0) (|containsVars| s1))
               (setq ns0 (|subCopy| s0 sl))
               (setq ns1 (|subCopy| s1 sl))
               (cond
                 ((or (|containsVars| ns0) (|containsVars| ns1))
                   (setq |$hope| t)
                   '|failed|)
                 (t
                  (cond
                   ((|canCoerce| ns0 ns1) (setq s3 s1))
                   ((|canCoerce| ns1 ns0) (setq s3 s0))
                   (t (setq s3 nil)))
                  (cond
                   (s3
                    (cond
                      ((not (equal s3 s0))
                        (setq sl (|augmentSub| v s3 sl))))
                    (cond
                     ((and (not (equal s3 s1)) (|isPatternVar| ss))
                        (setq sl (|augmentSub| ss s3 sl))))
                           sl)
                    (t '|failed|)))))
              (|$domPvar|
               (setq s3 (|resolveTT| s0 s1))
               (cond
                (s3
                 (cond
                  ((not (equal s3 s0))
                    (setq sl (|augmentSub| v s3 sl))))
                 (cond
                  ((and (not (equal s3 s1)) (|isPatternVar| ss))
                    (setq sl (|augmentSub| ss s3 sl))))
                 sl)
                (t '|failed|)))
              (t '|failed|)))
            (t '|failed|)))
          (t (|augmentSub| v ss s))))
        (t (|augmentSub| v ss sl)))))))
```

———-

24.0.63 defun containsVars

The function containsVars tests whether term t contains a * variable.

[isPatternVar p648]
[containsVars1 p648]

— **defun containsVars** —

```
(defun |containsVars| (arg)
 (if (atom arg)
```

```
  (|isPatternVar| arg)
  (|containsVars1| arg)))
```

———-

24.0.64 defun isPatternVar

— **defun isPatternVar** —

```
(defun |isPatternVar| (v)
  (and (identp v)
       (member v
             '(** *1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15
                 *16 *17 *18 *19 *20))
       t))
```

———-

24.0.65 defun containsVars1

The function containsVars1 tests whether term t contains a * variable. This is a recursive version, which works on a list.

[isPatternVar p648]
[containsVars1 p648]

— **defun containsVars1** —

```
(defun |containsVars1| (arg)
 (let ((t1 (car arg)) (t2 (cdr arg)))
  (if (atom t1)
     (or (|isPatternVar| t1)
         (if (atom t2) (|isPatternVar| t2) (|containsVars1| t2)))
     (or (|containsVars1| t1)
         (if (atom t2)  (|isPatternVar| t2) (|containsVars1| t2))))))
```

———-

24.0.66 defun hasCaty1

The cond is either a (has a b) or an OR clause of such conditions. SL is augmented, if cond is true, otherwise the result is 'failed

[hasCate p650]
[hasCaty1 p648]
[keyedSystemError p??]
[$domPvar p308]

— **defun hasCaty1** —

```
(defun |hasCaty1| (cond sl)
 (let (|$domPvar| a s)
 (declare (special |$domPvar|))
  (setq |$domPvar| nil)
   (cond
    ((and (consp cond) (eq (qcar cond) '|has|)
          (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
      (|hasCate| (qcadr cond) (qcaddr cond) sl))
    ((and (consp cond) (EQ (qcar cond) 'and))
     (loop for x in (qcdr cond)
      while (not (eq s '|failed|))
      do
       (setq s
        (cond
         ((and (consp x) (eq (qcar x) '|has|)
               (consp (qcdr x)) (consp (qcddr x)) (eq (qcdr (qcddr x)) nil))
           (|hasCate| (qcadr x) (qcaddr x) sl))
         ((and (consp x) (eq (qcdr x) nil)
                (consp (qcar x)) (eq (qcaar x) '|has|)
                (consp (qcdar x)) (consp (qcddar x))
                (eq (qcdr (qcddar x)) nil))
           (|hasCate| a (qcaddar x) sl))
         (t (|hasCaty1| x sl)))))
     s)
    ((and (consp cond) (eq (qcar cond) 'or))
      (do ((next (qcdr cond) (cdr next)) (x nil)
           (nextitem nil (null (eq s '|failed|))))
          ((or (atom next)
              (progn (setq x (car next)) nil)
               nextitem)
               nil)
       (setq s
        (cond
         ((and (consp x) (eq (qcar x) '|has|)
               (consp (qcdr x)) (consp (qcddr x)) (eq (qcdddr x) nil))
           (|hasCate| (qcadr x) (qcaddr x) (copy sl)))
         ((and (consp x) (eq (qcdr x) nil) (consp (qcar x))
               (eq (qcaar x) '|has|) (consp (qcdar x)) (consp (qcddar x))
               (eq (qcdddr x) nil))
           (|hasCate| (qcadar x) (qcaddar x) (copy sl)))
         (t (|hasCaty1| x (copy sl))))))
     s)
    (t
     (|keyedSystemError|
      "Unexpected error or improper call to system function %1: %2"
      (list  "hasCaty1" "unexpected condition from category table"))))))
```

———-

24.0.67 defun mkDomPvar

[domArg p640]
[length p??]
[$FormalMapVariableList p15]

— **defun mkDomPvar** —

```
(defun |mkDomPvar| (p d subs y)
 (let (l)
 (declare (special |$FormalMapVariableList|))
  (if (setq l (member p |$FormalMapVariableList|))
   (|domArg| d (- (|#| |$FormalMapVariableList|) (|#| l)) subs y)
   d)))
```

———-

24.0.68 defun hasCate

[isPatternVar p648]
[hasCate1 p644]
[hasCateSpecial p651]
[containsVariables p??]
[subCopy p??]
[hasCaty p638]
[$EmptyMode p629]
[$Subst p??]
[$hope p??]

— **defun hasCate** —

```
(defun |hasCate| (dom cat sl)
 (let (nsl p s sl1)
 (declare (special |$hope| |$Subst| |$EmptyMode|))
  (cond
   ((equal dom |$EmptyMode|) nil)
   ((|isPatternVar| dom)
    (cond
     ((and (setq p (assq dom sl))
           (not (eq (setq nsl (|hasCate| (cdr p) cat sl)) '|failed|)))
      nsl)
     ((or (setq p (assq dom |$Subst|)) (setq p (assq dom sl)))
       (setq s (|hasCate1| (cdr p) cat sl dom))
       (cond
         ((null (eq s '|failed|)) s)
         (t (|hasCateSpecial| dom (cdr p) cat sl))))
     (t
      (when (not (eq sl '|failed|)) (setq |$hope| t))
      '|failed|)))
   (t
    (setq sl1
     (loop for item in sl
```

```
      when (null (|containsVariables| (cdr item)))
      collect item))
    (when sl1 (setq cat (|subCopy| cat sl1)))
    (|hasCaty| dom cat sl)))))
```

———-

24.0.69 defun constructSubst

[internl 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 "#" (princ-to-string i)) x) sl))))
  sl))
```

———-

24.0.70 defun hasCateSpecial

The variable v is a pattern variable, dom is its binding under `$Subst`. We try to change dom so that it has category cat under sl. The result is a substitution list or 'failed.

[eqcar p??]
[isSubDomain p??]
[canCoerceFrom p??]
[containsVars p647]
[augmentSub p??]
[hasCate p650]
[hasCaty p638]
[hasCateSpecialNew p652]
[$Integer p632]
[$QuotientField p634]

— **defun hasCateSpecial** —

```
(defun |hasCateSpecial| (v dom cat sl)
 (let (arg d domp nsl)
 (declare (special |$Integer| |$QuotientField|))
  (cond
   ((and (consp dom) (eq (qcar dom) '|FactoredForm|)
         (consp (qcdr dom)) (eq (qcddr dom) nil))
     (setq arg (qcadr dom))
     (when (|isSubDomain| arg |$Integer|) (setq arg |$Integer|))
     (setq d (list '|FactoredRing| arg))
     (setq sl (|hasCate| arg '(|Ring|) (|augmentSub| v d sl)))
```

```
    (if (eq sl '|failed|)
     '|failed|
     (|hasCaty| d cat sl)))
  ((or (eqcar cat '|Field|) (eqcar cat '|DivisionRing|))
    (when (|isSubDomain| dom |$Integer|) (setq dom |$Integer|))
    (setq d (list |$QuotientField| dom))
    (|hasCaty| dom '(|IntegralDomain|) (|augmentSub| v d sl)))
  ((and (consp cat) (eq (qcar cat) '|PolynomialCategory|)
        (consp (qcdr cat)))
    (setq domp (cons '|Polynomial| (list (qcadr cat))))
    (and (or (|containsVars| (qcadr cat)) (|canCoerceFrom| dom domp))
             (|hasCaty| domp cat (|augmentSub| v domp sl))))
  ((|isSubDomain| dom |$Integer|)
    (setq nsl (|hasCate| |$Integer| cat (|augmentSub| v |$Integer| sl)))
    (if (eq nsl '|failed|)
     (|hasCateSpecialNew| v dom cat sl)
     (|hasCaty| |$Integer| cat nsl)))
  (t
    (|hasCateSpecialNew| v dom cat sl)))))
```

———-

24.0.71 defun hasCateSpecialNew

[member p1108]
[eqcar p??]
[augmentSub p??]
[defaultTargetFE p654]
[isEqualOrSubDomain p655]
[underDomainOf p??]
[hasCaty p638]
[$Integer p632]
[$ComplexInteger p632]
[$RationalNumber p633]

— **defun hasCateSpecialNew** —

```
(defun |hasCateSpecialNew| (v dom cat sl)
 (let (fe alg fefull d partialResult)
 (declare (special |$RationalNumber| |$ComplexInteger| |$Integer|))
  (setq fe
   (|member| (qcar cat)
    '(|ElementaryFunctionCategory|
      |TrigonometricFunctionCategory|
      |ArcTrigonometricFunctionCategory|
      |HyperbolicFunctionCategory|
      |ArcHyperbolicFunctionCategory|
      |PrimitiveFunctionCategory|
      |SpecialFunctionCategory|
      |Evalable|
      |CombinatorialOpsCategory|
      |TranscendentalFunctionCategory|
```

```
    |AlgebraicallyClosedFunctionSpace|
    |ExpressionSpace|
    |LiouvillianFunctionCategory|
    |FunctionSpace|)))
(setq alg
 (|member| (qcar cat)
  '(|RadicalCategory|
    |AlgebraicallyClosedField|)))
(setq fefull
 (or fe alg (eqcar cat '|CombinatorialFunctionCategory|)))
(setq partialResult
 (cond
  ((or (eqcar dom '|Variable|) (eqcar dom '|Symbol|))
    (cond
     ((|member| (car cat)
       '(|SemiGroup|
         |AbelianSemiGroup|
         |Monoid|
         |AbelianGroup|
         |AbelianMonoid|
         |PartialDifferentialRing|
         |Ring|
         |InputForm|))
       (setq d (list '|Polynomial| |$Integer|))
       (|augmentSub| v d sl))
     ((eqcar cat '|Group|)
       (setq d (list '|Fraction| (list '|Polynomial| |$Integer|)))
       (|augmentSub| v d sl))
     (fefull
      (setq d (|defaultTargetFE| dom))
      (|augmentSub| v d sl))
     (t '|failed|)))
  ((|isEqualOrSubDomain| dom |$Integer|)
    (cond
     (fe
       (setq d (|defaultTargetFE| |$Integer|))
       (|augmentSub| v d sl))
     (alg
       (setq d '(|AlgebraicNumber|))
       (|augmentSub| v d sl))
     (t '|failed|)))
  ((equal (|underDomainOf| dom) |$ComplexInteger|)
    (setq d (|defaultTargetFE| |$ComplexInteger|))
    (|hasCaty| d cat (|augmentSub| v d sl)))
  ((and (equal dom |$RationalNumber|) alg)
    (setq d '(|AlgebraicNumber|))
    (|augmentSub| v d sl))
  (fefull
    (setq d (|defaultTargetFE| dom))
    (|augmentSub| v d sl))
  (t '|failed|)))
(if (eq partialResult '|failed|)
  '|failed|
  (|hasCaty| d cat partialResult))))
```

———-

24.0.72 defun defaultTargetFE

[isEqualOrSubDomain p655]
[ifcar p??]
[defaultTargetFE p654]
[$FunctionalExpression p634]
[$Integer p632]
[$Symbol p634]
[$RationalNumber p633]

— **defun defaultTargetFE** —

```
(defun |defaultTargetFE| (&rest dom)
 (let (a options)
 (declare (special |$FunctionalExpression| |$Integer| |$Symbol|
                   |$RationalNumber|))
  (setq a (car dom))
  (setq options (cdr dom))
  (cond
   ((or (and (consp a) (eq (qcar a) '|Variable|)
             (consp (qcdr a)) (eq (qcddr a) nil))
        (equal a |$RationalNumber|)
        (member (qcar a) (list (qcar |$Symbol|) '|RationalRadicals| '|Pi|))
        (equal a |$SingleInteger|)
        (|isEqualOrSubDomain| a |$Integer|)
        (equal a '(|AlgebraicNumber|)))
     (if (ifcar options)
      (list |$FunctionalExpression| (list '|Complex| |$Integer|))
      (list |$FunctionalExpression| |$Integer|)))
   ((and (consp a) (eq (qcar a) '|Complex|)
         (consp (qcdr a)) (eq (qcddr a) nil))
     (|defaultTargetFE| (qcadr a) t))
   ((and (consp a) (consp (qcdr a)) (eq (qcddr a) nil)
         (member (qcar a) '(|Polynomial| |RationalFunction| |Fraction|)))
     (|defaultTargetFE| (qcadr a) (ifcar options)))
   ((and (consp a) (equal (qcar a) |$FunctionalExpression|)
         (consp (qcdr a)) (eq (qcddr a) nil))
     a)
   ((ifcar options)
     (list |$FunctionalExpression| (list '|Complex| a)))
   (t
     (list |$FunctionalExpression| a)))))
```

———-

24.0.73 defun isEqualOrSubDomain

[isSubDomain p**??**]

— **defun isEqualOrSubDomain** —

```
(defun |isEqualOrSubDomain| (d1 d2)
  (or (equal d1 d2)
      (|isSubDomain| d1 d2)
      (and (atom d1)
           (or (and (consp d2) (eq (qcar d2) '|Variable|)
                    (consp (qcdr d2)) (eq (qcddr d2) nil)
                    (equal (qcadr d2) d1))
               (and (consp d2) (eq (qcdr d2) nil)
                    (equal (qcar d2) d1))))
      (and (atom d2)
           (or (and (consp d1) (eq (qcar d1) '|Variable|)
                    (consp (qcdr d1)) (eq (qcddr d1) nil)
                    (equal (qcadr d1) d2))
               (and (consp d1) (eq (qcdr d1) nil)
                    (equal (qcar d1) d2))))))
```

———-

Chapter 25

Coercions

```
 main algorithms for canCoerceFrom and coerceInteractive

coerceInteractive and canCoerceFrom are the two coercion functions
for $InteractiveMode. They translate RN, RF and RR to QF I, QF P
and RE RN, respectively, and call coerceInt or canCoerce, which
both work in the same way (e.g. coercion from t1 to t2):
1. they try to coerce t1 to t2 directly (tower coercion), and, if
  this fails, to coerce t1 to the last argument of t2 and embed
  this last argument into t2. These embedding functions are now only
  defined in the algebra code. (RSS 2-27-87)
2. the tower coercion looks whether there is any applicable local
  coercion, which means, one defined in boot or in algebra code.
  If there is an applicable function from a constructor, which is
  inside the type tower of t1, to the top level constructor of t2,
  then this constructor is bubbled up inside t1. This means,
  special coercion functions (defined in boot) are called, which
  commute two constructors in a tower. Then the local coercion is
  called on these constructors, which both are on top level now.
example:
let t1 = A B C D E (short for (A (B (C (D (E))))), where A ... E are
  type constructors), and t2 = F D G H I J
there is no coercion from t1 to t2 directly, so we try to coerce
  t1 to s1 = D G H I J, the last argument of t2
we create the type s2 = A D B C E and call a local coercion A2A
  from t1 to s2, which, by recursively calling coerce, bubbles up
  the constructor D
then we call a commute coerce from s2 to s3 = D A B C E and a local
  coerce D2D from s3 to s1
finally we embed s1 into t2, which completes the coercion t1 to t2
the result of canCoerceFrom is TRUE or NIL
the result of coerceInteractive is a object or NIL (=failed)
all boot coercion functions have the following result:
1. if u=$fromCoerceable$, then TRUE or NIL
2. if the coercion succeeds, the coerced value (this may be NIL)
3. if the coercion fails, they throw to a catch point in
     coerceByFunction
```

25.0.74 defun coerceInteractive

[objMode p462]
[objVal p462]
[clearDependentMaps p??]
[throwKeyedMsg p??]
[startTimingProcess p??]
[mkObj p460]
[mkObjWrap p461]
[coerceInt0 p659]
[stopTimingProcess p??]
[$insideCoerceInteractive p??]
[$OutputForm p631]
[$mapName p??]
[$compilingMap p308]
[$NoValueMode p??]
[$EmptyMode p629]

— defun coerceInteractive —

```
(defun |coerceInteractive| (triple t2)
 (let (|$insideCoerceInteractive| t1 val expr2 result)
  (declare (special |$insideCoerceInteractive| |$OutputForm|
                    |$mapName| |$compilingMap| |$NoValueMode| |$EmptyMode|))
   (setq t1 (|objMode| triple))
   (setq val (|objVal| triple))
   (cond
    ((or (null t2) (equal t2 |$EmptyMode|)) nil)
    ((equal t2 t1) triple)
    ((equal t2 '|$NoValueMode|) (mkObj val t2))
    (t
     (when (eq (car t2) '|SubDomain|) (setq t2 (second t2)))
     (cond
      ((|member| t1
        '((|Category|) (|Mode|) (|Domain|) (|SubDomain| (|Domain|))))
       (when (equal t2 |$OutputForm|) (mkObj val t2)))
      ((equal t1 '|$NoValueMode|)
       (when |$compilingMap| (|clearDependentMaps| |$mapName| nil))
       (|throwKeyedMsg|
        (format nil
          "You are trying to use something (probably a loop) in a ~
           situation where a value is expected.  In particular, you ~
           are trying to convert this to the type %1p . The following ~
           information may help:  possible function name:  %2p")
         (list t2 |$mapName|)))
      (t
       (setq |$insideCoerceInteractive| t)
       (setq expr2 (equal t2 |$OutputForm|))
       (cond
        (expr2 (|startTimingProcess| '|print|))
        (t (|startTimingProcess| '|coercion|)))
       (setq result
        (cond
```

```
((and expr2 (equal t1 val)) (mkObj val |$OutputForm|))
((and expr2 (eq (car t1) '|Variable|))
  (mkObjWrap (second t1) |$OutputForm|))
(t (|coerceInt0| triple t2))))
 (cond
  (expr2 (|stopTimingProcess| '|print|))
  (t (|stopTimingProcess| '|coercion|)))
 result))))))
```

———-

25.0.75 defun coerceInt

[coerceInt1 p660]
[objMode p462]
[getMinimalVarMode p??]
[unwrap p??]
[objVal p462]
[coerceInt p659]

— **defun coerceInt** —

```
(defun |coerceInt| (triple t2)
 (let (val newMode newVal)
  (if (setq val (|coerceInt1| triple t2))
    val
    (when (eq (car (|objMode| triple)) '|Variable|)
     (setq newMode (|getMinimalVarMode| (|unwrap| (|objVal| triple)) nil))
     (setq newVal (|coerceInt| triple newMode))
     (|coerceInt| newVal t2)))))
```

———-

25.0.76 defun coerceInt0

[objVal p462]
[objMode p462]
[conCoerceFrom p??]
[isWrapped p1485]
[intCodeGenCOERCE p??]
[unwrap p??]
[coerceInt0 p659]
[mkObj p460]
[coerceInt p659]
[objSetMode p461]
[$OutputForm p631]
[$Any p630]
[$genValue p312]
This is the top level interactive coercion, which transfers all RN, RF and RR into equivalent types

— defun coerceInt0 —

```
(defun |coerceInt0| (triple t2)
 (prog (val t1 s1 s2 let1 t1p valp ans x)
  (declare (special |$OutputForm| |$Any| |$genValue|))
  (return
   (progn
    (setq val (|objVal| triple))
    (setq t1 (|objMode| triple))
    (cond
     ((eq val '|$fromCoerceable$|) (|canCoerceFrom| t1 t2))
     ((equal t1 t2) triple)
     (t
      (cond
       ((equal t2 |$OutputForm|) (setq s1 t1) (setq s2 t2))
       (t
        (setq s1 t1)
        (setq s2 t2)
        (when (equal s1 s2) (return (mkObj val t2)))))
      (cond
       ; handle case where we must generate code
       ((and (null (|isWrapped| val))
             (or
              (null (eq (car t1) '|FunctionCalled|))
              (null |$genValue|)))
        (|intCodeGenCOERCE| triple t2))
       ((and (equal t1 |$Any|)
             (nequal t2 |$OutputForm|)
             (progn
              (setq let1 (|unwrap| val))
              (setq t1p (car let1))
              (setq valp (cdr let1))
              let1)
             (setq ans (|coerceInt0| (mkObjWrap valp t1p) t2)))
         ans)
       (t
        (unless (eq s1 t1) (setq triple (mkObj val s1)))
        (when (setq x (|coerceInt| triple s2))
         (cond
          ((eq s2 t2) x)
          (t
           (|objSetMode| x t2)
           x)))))))))))
```

———-

25.0.77 defun coerceInt1

This is general interactive coercion. The result is a new triple with type m2 or NIL (= failed). [NRTcompileEvalForm p**??**]
[absolutelyCanCoerceByCheating p685]
[asTupleAsList p**??**]
[bottomUp p**??**]

[coerceByFunction p665]
[coerceInt1 p660]
[coerceInt2Union p680]
[coerceIntAlgebraicConstant p682]
[coerceIntFromUnion p680]
[coerceIntTower p667]
[coerceIntX p683]
[coerceInt p659]
[coerceRetract p691]
[coerceSubDomain p683]
[compareTypeLists p683]
[deconstructT p??]
[evalDomain p993]
[getFunctionFromDomain p??]
[getValue p??]
[isEqualOrSubDomain p655]
[isSubDomain p??]
[mkAtreeNode p??]
[mkAtree p??]
[mkObjWrap p461]
[mkObj p460]
[nequal p??]
[nreverse0 p??]
[objMode p462]
[objVal p462]
[selectLocalMms p??]
[selectMms1 p??]
[transferPropsToNode p??]
[unwrap p??]
[coerceOrCroaker p??]
[$useCoerceOrCroak p??]
[$Integer p632]
[$QuotientField p634]
[$e p285]
[$genValue p312]
[$Symbol p634]
[$AnonymousFunction p629]
[$OutputForm p631]
[$String p633]
[$Any p630]
[$Void p634]
[$NonNegativeInteger p633]
[$PositiveInteger p633]
[$EmptyMode p629]
[$SingleInteger p635]

— **defun coerceInt1** —

```
(defun |coerceInt1| (triple t2)
 (prog (|$useCoerceOrCroak| t1 sintp t1p valp s body vars tree val symNode
```

```
       mms ml oldName intName t3 triplep let1 arg tt ans)
 (declare (special |$useCoerceOrCroak| |$Integer| |$QuotientField|
                   |$e| |$genValue| |$Symbol| |$AnonymousFunction|
                   |$OutputForm| |$String| |$Any| |$Void| |$SingleInteger|
                   |$NonNegativeInteger| |$PositiveInteger| |$EmptyMode|))
  (return
   (seq
    (progn
     (setq |$useCoerceOrCroak| t)
     (cond
      ((equal t2 |$EmptyMode|) nil)
      (t
       (setq t1 (|objMode| triple))
       (cond
        ((equal t1 t2) triple)
        (t
         (setq val (|objVal| triple))
         (cond
          ((|absolutelyCanCoerceByCheating| t1 t2) (mkObj val t2))
          ((|isSubDomain| t2 t1) (|coerceSubDomain| val t1 t2))
          (t
           (cond
            ((equal t1 |$SingleInteger|)
             (cond
              ((or (equal t2 |$Integer|) (equal t2 |$SingleInteger|))
                (return (mkObj val t2)))
              (t
               (setq sintp (typep val 'fixnum))
               (cond
                ((and sintp (equal t2 |$PositiveInteger|) (> val 0))
                  (return (mkObj val t2)))
                ((and sintp (equal t2 |$NonNegativeInteger|) (>= val 0))
                  (return (mkObj val t2)))))))))
           (cond
            ((and (equal t2 |$SingleInteger|)
                  (|isEqualOrSubDomain| t1 |$Integer|)
                  (integerp val))
              (cond
               ((typep val 'fixnum) (mkObj val t2))
               (t nil)))
            ((equal t2 |$Void|) (mkObj (|voidValue|) |$Void|))
            ((equal t2 |$Any|) (mkObjWrap (cons t1 (|unwrap| val)) '(|Any|)))
            ((and (equal t1 |$Any|)
                  (nequal t2 |$OutputForm|)
                  (progn
                   (setq let1 (|unwrap| val))
                   (setq t1p (car let1))
                   (setq valp (cdr let1))
                   let1)
                  (setq ans (|coerceInt| (mkObjWrap valp t1p) t2)))
              ans)
             ; tagged union selectors
            ((or (and (eq (car t1) '|Variable|) (equal (cadr t1) t2))
                 (and (eq (car t2) '|Variable|) (equal (cadr t2) t1)))
```

```
  (mkObj val t2))
((stringp t2)
     (cond
      ((and (eq (first t1) '|Variable|)
            (equal t2 (pname (second t1))))
        (mkObjWrap t2 t2))
      (t
       (setq valp (|unwrap| val))
       (when (and (equal t2 valp)
                  (or (equal valp t1) (equal t1 |$String|)))
         (mkObj val t2)))))
((eq (first t1) '|Tuple|)
  (|coerceInt1|
   (mkObjWrap
    (|asTupleAsList| (|unwrap| val))
    (list '|List| (setq s (second t1))))
   t2))
((and (consp t1) (eq (qcar t1) '|Union|))
  (|coerceIntFromUnion| triple t2))
((and (consp t2) (eq (qcar t2) '|Union|))
  (|coerceInt2Union| triple t2))
((and (stringp t1) (equal t2 |$String|))
  (mkObj val |$String|))
((and (stringp t1) (eq (car t2) '|Variable|))
  (when (equal t1 (pname (second t2))) (mkObjWrap (second t2) t2)))
((and (stringp t1) (equal t1 (|unwrap| val)))
  (when (equal t2 |$OutputForm|) (mkObj t1 |$OutputForm|)))
((atom t1) nil)
(t
 (cond
  ((and (equal t1 |$AnonymousFunction|)
        (eq (car t2) '|Mapping|))
    (setq |$useCoerceOrCroak| nil)
    (setq let1 (|unwrap| val))
    (setq vars (cadr let1))
    (setq body (cddr let1))
    (setq vars
     (cond
      ((atom vars) (cons vars nil))
      ((and (consp vars) (eq (qcar vars) '|Tuple|)) (cdr vars))
      (t vars)))
    (cond
     ((nequal (|#| (cddr t2)) (|#| vars)) '|continue|)
     (t
      (setq tree
       (|mkAtree|
        (cons 'adef
         (cons vars
          (cons (cons (cadr t2) (cddr t2))
           (cons (loop for x in (cdr t2) collect nil)
            body))))))
      (cond
       ((eq
         (catch '|coerceOrCroaker| (|bottomUp| tree)) '|croaked|)
```

```
          nil)
         (t (return (|getValue| tree))))))))
(cond
 ((and (equal t1 |$Symbol|) (eq (car t2) '|Mapping|))
   (cond
    ((null (setq mms
            (|selectMms1| (|unwrap| val) nil
              (cddr t2) (cddr t2) (cadr t2))))
      nil)
    (t
     (cond
      ((nequal (cadaar mms) (cadr t2)) nil)
      (|$genValue|
       (mkObjWrap
        (|getFunctionFromDomain|
         (|unwrap| val) (caaar mms) (cddaar mms)) t2))
      (t
       (mkObj
        (|NRTcompileEvalForm|
           (|unwrap| val) (cdaar mms) (|evalDomain| (caaar mms)))
         t2))))))
 ((and (eq (car t1) '|Variable|) (eq (car t2) '|Mapping|))
   (setq mms
    (|selectMms1| (cadr t1) (cadr t2) (cddr t2) (cddr t2) nil))
   (cond
    ((and (null mms)
          (null
           (setq mms
            (|selectMms1| (cadr t1) (cadr t2)
                          (cddr t2) (cddr t2) t))))
      nil)
    (t
     (cond
      ((nequal (cadaar mms) (cadr t2)) nil)
      ((eq (caaaar mms) '|_FreeFunction_|)
        (mkObj (cdaaar mms) t2))
      (|$genValue|
        (mkObjWrap
         (|getFunctionFromDomain| (cadr t1) (caaar mms)
                                  (cddaar mms)) t2))
      (t
       (mkObj
         (|NRTcompileEvalForm| (cadr t1) (cdr (caar mms))
          (|evalDomain| (caaar mms)))
         t2))))))
 ((and (eq (car t1) '|FunctionCalled|) (eq (qcar t2) '|Mapping|))
   (setq symNode (|mkAtreeNode| (cadr t1)))
   (|transferPropsToNode| (cadr t1) symNode)
   (cond
    ((null
      (setq mms
       (|selectLocalMms| symNode (cadr t1) (cddr t2) (cadr t2))))
      nil)
    (t
```

```
    (cond
     ((nequal (cadaar mms) (cadr t2)) nil)
     (t
      (setq ml (cons (cadr t2) (cddr t2)))
      (setq intName
       (when
        (some #'(lambda (mm)
                 (setq oldName (second mm))
                 (|compareTypeLists| (cdar mm) ml)) mms)
        (list oldName)))
      (cond
       ((null intName) nil)
       (t (mkObjWrap intName t2))))))))
((eq (car t1) '|FunctionCalled|)
  (setq t3 (|get| (second t1) '|mode| |$e|))
  (when (and (eq (car t3) '|Mapping|)
             (setq triplep (|coerceInt| triple t3)))
   (|coerceInt| triplep t2)))
((and (eq (car t1) '|Variable|)
      (consp t2)
      (or (|isEqualOrSubDomain| t2 |$Integer|)
          (equal t2 (list |$QuotientField| |$Integer|))
          (member (car t2)
           '(|RationalNumber| |BigFloat|
             |NewFloat| |Float| |DoubleFloat|))))
   nil)
(t
 (setq ans
  (or
   (|coerceRetract| triple t2)
   (|coerceIntTower| triple t2)
   (progn
    (setq arg (cdr (|deconstructT| t2)))
    (and arg
     (progn
       (setq tt (|coerceInt| triple (|last| arg)))
       (and tt (|coerceByFunction| tt t2)))))))
 (or ans
     (and (|isSubDomain| t1 |$Integer|)
          (|coerceInt| (mkObj val |$Integer|) t2))
     (|coerceIntAlgebraicConstant| triple t2)
     (|coerceIntX| val t1 t2))))))))))))))))

————————-
```

25.0.78 defun coerceByFunction

— **defun coerceByFunction** —

```
(defun |coerceByFunction| (t$ m2)
 (let ($ m1 ud x tmp1 a tmp2 b funName mm dc tar args slot dcVector fun fn
       d val env code)
```

```
(declare (special $ |$coerceFailure| |$Boolean|))
(setq x (|objVal| T$))
(cond
 ((eq x '|$fromCoerceable$|) nil)
 ((eq (car m2) '|Union|) nil)
 (t
  (setq m1 (|objMode| t$))
  (cond
   ((and (consp m2) (eq (qcar m2) '|Boolean|)
         (consp m1) (eq (qcar m1) '|Equation|)
         (PROGN
          (setq tmp1 (cdr m1))
          (and (consp tmp1) (eq (cdr tmp1) nil)
               (progn (setq ud (car tmp1)) t))))
    (setq dcVector (|evalDomain| ud))
    (setq fun
     (cond
      ((|isWrapped| x)
       (|NRTcompiledLookup| '= (list |$Boolean| '$ '$) dcVector))
      (t
       (|NRTcompileEvalForm| '= (list |$Boolean| '$ '$) dcVector))))
    (setq fn (car fun))
    (setq d (cdr fun))
    (cond
     ((|isWrapped| x)
      (setq x (|unwrap| x))
      (mkObjWrap (spadcall (car x) (cdr x) fun) m2))
     ((null (and (consp x) (eq (car x) 'spadcall)
                 (progn
                  (setq tmp1 (cdr x))
                  (and (consp tmp1)
                       (progn
                        (setq a (car tmp1))
                        (setq tmp2 (cdr tmp1))
                        (and (consp tmp2)
                             (progn
                              (setq b (car tmp2)) t)))))))
      (|keyedSystemError| "Generated code is incorrect for equation" nil))
     (t
      (setq code (list 'spadcall a b fun))
      (mkObj code |$Boolean|))))
   (t
    (cond
     ((null
       (setq mm (|coerceConvertMmSelection| (setq funName '|coerce|) m1 m2)))
      (setq mm
       (|coerceConvertMmSelection| (setq funName '|convert|) m1 m2))))
    (when mm
      (setq dc (caar mm))
      (setq tar (cadar mm))
      (setq args (cddar mm))
      (setq slot (cadr mm))
      (setq dcVector (|evalDomain| dc))
      (setq fun
```

```
 (cond
  ((|isWrapped| x) (|NRTcompiledLookup| funName slot dcVector))
  (t (|NRTcompileEvalForm| funName slot dcVector))))
(setq fn (car fun))
(setq d (cdr fun))
(cond
 ((equal fn #'|Undef|) nil)
 ((|isWrapped| x)
  (setq $ dcVector)
  (setq val (catch '|coerceFailure| (spadcall (|unwrap| x) fun)))
  (cond
   ((equal val |$coerceFailure|) nil)
   (t (mkObjWrap val m2))))
 (t
  (setq env fun)
  (setq code (list '|failCheck| (list 'spadcall x env)))
  (mkObj code m2)))))))))
```

———-

25.0.79 defun coerceIntTower

This tries to find a coercion from top level t2 to somewhere inside t1 It builds a new argument type, for which coercion is called recursively [coerceIntPermute p670]
[coerceIntSpecial p676]
[last p**??**]
[coerceIntTest p668]
[constructT p**??**]
[replaceLast p**??**]
[deconstructT p**??**]
[bubbleConstructor p**??**]
[isValidType p**??**]
[coerceIntCommute p673]
[coerceIntByMap p677]
[coerceIntTableOrFunction p675]

— **defun coerceIntTower** —

```
(defun |coerceIntTower| (triple t2)
 (let (t1 c1 arg1 tt c arg tl let1 c2 arg2 s x)
  (cond
   ((setq x (|coerceIntByMap| triple t2)) x)
   ((setq x (|coerceIntCommute| triple t2)) x)
   ((setq x (|coerceIntPermute| triple t2)) x)
   ((setq x (|coerceIntSpecial| triple t2)) x)
   ((setq x (|coerceIntTableOrFunction| triple t2)) x)
   (t
    (setq t1 (|objMode| triple))
    (setq let1 (|deconstructT| t1))
    (setq c1 (car let1))
    (setq arg1 (cdr let1))
    (and arg1
```

```
(progn
 (setq tl nil)
 (setq arg arg1)
 (loop until (or x (not arg)) do
  (setq tt (|last| arg))
  (setq let1 (|deconstructT| tt))
  (setq c (car let1))
  (setq arg (cdr let1))
  (setq tl (cons c (cons arg tl)))
  (cond
   ((setq x (and arg (|coerceIntTest| tt t2)))
    (cond
     ((cddr tl)
       (setq s
        (|constructT| c1
         (|replaceLast| arg1 (|bubbleConstructor| tl))))
       (cond
        ((null (|isValidType| s)) (setq x nil))
        ((setq x (or (|coerceIntByMap| triple s)
                     (|coerceIntTableOrFunction| triple s)))
          (setq let1 (|deconstructT| (|last| s)))
          (setq c2 (car let1))
          (setq arg2 (cdr let1))
          (setq s (|bubbleConstructor| (list c2 arg2 c1 arg1)))
          (cond
           ((null (|isValidType| s)) (setq x nil))
           ((setq x (|coerceIntCommute| x s))
             (setq x (or (|coerceIntByMap| x t2)
                         (|coerceIntTableOrFunction| x t2))))))))
     (t
      (setq s (|bubbleConstructor| (list c arg c1 arg1)))
      (cond
       ((null (|isValidType| s)) (setq x nil))
       ((setq x (|coerceIntCommute| triple s))
         (setq x (or (|coerceIntByMap| x t2)
                     (|coerceIntTableOrFunction| x t2))))))))))
 x))))))
```

———-

25.0.80 defun coerceIntTest

This looks whether there exists a table entry or a coercion function. Thus the type can be bubbled before coerceIntTableOrFunction is called. [coerceConvertMmSelection p669]
[assq p1110]
[$CoerceTable p**??**]
[$useConvertForCoercions p**??**]

— **defun coerceIntTest** —

```
(defun |coerceIntTest| (t1 t2)
 (let (p b)
  (declare (special |$useConvertForCoercions| |$CoerceTable|))
```

```
  (or (equal t1 t2)
      (setq b
       (and (setq p (assq (car t1) |$CoerceTable|))
            (assq (car t2) (cdr p))))
      (or b
          (|coerceConvertMmSelection| '|coerce| t1 t2)
          (and |$useConvertForCoercions|
               (|coerceConvertMmSelection| '|convert| t1 t2))))))
```

———-

25.0.81 defvar coerceConvertMmSelection;AL

— **initvars** —

```
(defvar |coerceConvertMmSelection;AL| (make-hash-table :test #'equal))
```

———-

25.0.82 defun coerceConvertMmSelection

This calls selectMms with `$Coerce=NIL` and tests for required target type. funName is either 'coerce or 'convert.

```
  mmS := [[sig,[targ,arg],:pred] for x in l | x is [sig,[.,arg],:pred] and
    hasCorrectTarget(m2,sig) and sig is [dc,targ,oarg] and oarg = m1]
```

[selectMms1 p**??**]
[$reportBottomUpFlag p898]
[$declaredMode p**??**]
[$coerceConvertMmSelection;AL p669]

— **defun coerceConvertMmSelection** —

```
(defun |coerceConvertMmSelection| (&rest g1)
 (labels (
  (checktargets (funName m1 m2)
   (let (|$declaredMode| |$reportBottomUpFlag|)
   (declare (special |$declaredMode| |$reportBottomUpFlag|
             |coerceConvertMmSelection;AL|))
    (setq |$declaredMode| nil)
    (setq |$reportBottomUpFlag| nil)
    (car
     (loop for x in (|selectMms1| funName m2 (list m1) (list m1) nil)
      collect
      (when (and (|hasCorrectTarget| m2 (car x)) (equal (caddar x) m1))
       (cons (car x) (cons (cons (cadar x) (list (cadadr x))) (cddr x))))))))))
 (let (g3)
  (if (setq g3 (hget |coerceConvertMmSelection;AL| g1))
    (|CDRwithIncrement| g3)
    (cdr (hput |coerceConvertMmSelection;AL| g1
          (cons 1 (apply #'checktargets g1))))))))
```

———-

25.0.83 defun hasCorrectTarget

This tests whether the target of signature sig is either m or a union containing m. It also discards TEQ as it is not meant to be used at top-level

— **defun hasCorrectTarget 0** —

```
(defun |hasCorrectTarget| (m sig)
 (let (tar)
  (setq tar (second sig))
  (cond
   ((eq (caar sig) '|TypeEquivalence|) nil)
   ((equal m tar) t)
   ((and (eq (car tar) '|Union|)
         (eq (third tar) '|failed|))
    (equal (second tar) m))
   ((and (eq (car tar) '|Union|)
         (eq (second tar) '|failed|)
         (equal (third tar) m))))))
```

———-

25.0.84 defun coerceIntPermute

[member p1108]
[objMode p462]
[computeTTTranspositions p671]
[coerceInt p659]

— **defun coerceIntPermute** —

```
(defun |coerceIntPermute| (object t2)
 (let (t1 towers ok)
  (cond
   ((|member| t2 '((|Integer|) (|OutputForm|))) nil)
   (t
     (setq t1 (|objMode| object))
     (setq towers (|computeTTTranspositions| t1 t2))
     ; At this point, CAR towers = t1 and last towers should be similar
     ; to t2 in the sense that the components of t1 are in the same order
     ; as in t2. If length towers = 2 and t2 = last towers, we quit to
     ; avoid an infinte loop.
     (cond
      ((or (null towers) (null (cdr towers))) nil)
      ((and (null (cddr towers)) (equal t2 (cadr towers))) nil)
      (t
       (setq ok t)
       ;  do the coercions successively, quitting if any fail
       (loop for tt in (cdr towers) while ok do
         (unless (setq object (|coerceInt| object tt)) (setq ok nil)))
```

```
(when ok object)))))))
```

———-

25.0.85 defun computeTTTranspositions

[decomposeTypeIntoTower p673]
[member p1108]
[nequal p??]
[msort p??]
[remdup p??]
[length p??]
[list2vec p??]
[permuteToOrder p672]
[vec2list p1143]
[reassembleTowerIntoType p673]

— **defun computeTTTranspositions** —

```
(defun |computeTTTranspositions| (t1 t2)
 (labels (
  (compress (z start len)
   (cond
    ((>= start len) z)
    ((|member| start z) (compress z (1+ start) len))
    (t
     (compress
      (loop for i in z collect (if (> start i) i (1- i))) start len)))))
 (let (tl1 tl2 p2p n1 p2 perms tower tt towers)
  ;  decompose t1 into its tower parts
  (setq tl1 (|decomposeTypeIntoTower| t1))
  (setq tl2 (|decomposeTypeIntoTower| t2))
  (cond
   ; if not at least 2 parts, don't bother working here
   ((null (and (cdr tl1) (cdr tl2))) nil)
   (t
    ; determine the relative order of the parts of t1 in t2
    (setq p2 (nreverse0 (loop for d1 in tl1 collect (position d1 tl2))))
    (cond
     ; something not present
     ((|member| (- 1) p2) nil)
     (t
      ; if they are all ascending, this function will do nothing
      (setq p2p (msort p2))
      (cond
       ((equal p2 p2p) nil)
       ; if anything is repeated twice, leave
       ((nequal p2p (msort (remdup p2p))) nil)
       (t
        ; create a list of permutations that transform the tower parts
        ; of t1 into the order they are in in t2
        (setq n1 (|#| tl1))
```

```
        (setq p2 (list2vec (compress p2 0 (|#| (remdup tl1)))))
        ; p2 now has the same position numbers as p1, we need to determine
        ; a list of permutations that takes p1 into p2.
        (setq perms (|permuteToOrder| p2 (- n1 1) 0))
        (setq towers (list tl1))
        (setq tower (list2vec tl1))
        (loop for perm in perms do
         (setq tt (elt tower (car perm)))
         (setf (elt tower (car perm)) (elt tower (cdr perm)))
         (setf (elt tower (cdr perm)) tt)
         (setq towers (cons (vec2list tower) towers)))
        (setq towers (nreverse0
         (loop for tower in towers collect (|reassembleTowerIntoType| tower))))
        (unless (equal (car towers) t2) (setq towers (cons t2 towers)))
        (nreverse towers)))))))))
```

———-

25.0.86 defun permuteToOrder

[permuteToOrder p672]

— **defun permuteToOrder** —

```
(defun |permuteToOrder| (p n start)
 (let (r x perms tt stpos)
  (setq r (- n start))
 (cond
  ((<= r 0) nil)
  ((eql r 1)
   (cond
    ((> (elt p (+ r 1)) (elt p r)) nil)
    (t (list (cons r (+ r 1))))))
  ((equal (elt p start) start) (|permuteToOrder| p n (+ start 1)))
  (t
   (setq stpos nil)
   (loop for i from (+ start 1) to n while (not stpos) do
    (when (equal (elt p i) start) (setq stpos i)))
   (setq perms nil)
   (loop while (not (equal stpos start)) do
    (setq x (- stpos 1))
    (setq perms (cons (cons x stpos) perms))
    (setq tt (elt p stpos))
    (setf (elt p stpos) (elt p x))
    (setf (elt p x) tt)
    (setq stpos x))
   (append (nreverse perms) (|permuteToOrder| p n (+ start 1)))))))
```

———-

25.0.87 defun decomposeTypeIntoTower

[decomposeTypeIntoTower p673]
[deconstructT p??]

— **defun decomposeTypeIntoTower** —

```
(defun |decomposeTypeIntoTower| (tt)
 (let (rd)
  (cond
   ((atom tt) (list tt))
   ((null (cdr (|deconstructT| tt))) (list tt))
   (t
    (setq rd (reverse tt))
    (cons (reverse (cdr rd)) (|decomposeTypeIntoTower| (car rd)))))))
```

———-

25.0.88 defun reassembleTowerIntoType

[reassembleTowerIntoType p673]

— **defun reassembleTowerIntoType** —

```
(defun |reassembleTowerIntoType| (tower)
 (let (let1)
  (cond
   ((atom tower) tower)
   ((null (cdr tower)) (car tower))
   (t
    (setq let1 (reverse tower))
    (|reassembleTowerIntoType|
      (append (nreverse (cddr let1))
              (list (append (second let1) (list (first let1)))))))))))
```

———-

25.0.89 defun coerceIntCommute

[objMode p462]
[coerceCommuteTest p674]
[underDomainOf p??]
[getl p1110]
[concat p1107]
[objValUnwrap p462]
[mkObjWrap p461]
[$coerceFailure p??]
[coerceFailure p??]

— **defun coerceIntCommute** —

```
(defun |coerceIntCommute| (obj target)
 (let (source s t$ d fun u c)
  (declare (special |$coerceFailure|))
  (setq source (|objMode| obj))
  (cond
   ((null (|coerceCommuteTest| source target)) nil)
   (t
    (setq s (|underDomainOf| source))
    (setq t$ (|underDomainOf| target))
    (cond
     ((equal source t$) nil)
     ((setq d (car source))
      (setq fun
       (or (getl d '|coerceCommute|)
           (intern (concat "commute" (princ-to-string d)))))
      (cond
       ((canFuncall? fun)
        (put d '|coerceCommute| fun)
        (setq u (|objValUnwrap| obj))
        (setq c (catch '|coerceFailure| (funcall fun u source s target t$)))
        (cond
         ((equal c |$coerceFailure|) nil)
         ((eq u '|$fromCoerceable$|) c)
         (t (mkObjWrap c target)))))))))))
```

———-

25.0.90 defun coerceCommuteTest

[isLegitimateMode p**??**]
[underDomainOf p**??**]
[deconstructT p**??**]

— **defun coerceCommuteTest** —

```
(defun |coerceCommuteTest| (t1 t2)
 (let (u1 u2)
  (cond
   ((null (|isLegitimateMode| t2 nil nil)) nil)
   ((null (setq u1 (|underDomainOf| t1))) nil)
   ((null (setq u2 (|underDomainOf| t2))) nil)
   ((null (|underDomainOf| u1)) nil)
   ((null (|underDomainOf| u2)) nil)
   (t
    (and (equal (car (|deconstructT| t1)) (car (|deconstructT| u2)))
         (equal (car (|deconstructT| t2)) (car (|deconstructT| u1))))))))
```

———-

25.0.91 defun coerceIntTableOrFunction

This function does the actual coercion to t2, but not to an argument type of t2 [isValidType p??]
[isLegitimateMode p??]
[objMode p462]
[assq p1110]
[coerceByTable p675]
[objVal p462]
[coerceByFunction p665]
[$CoerceTable p??]

— **defun coerceIntTableOrFunction** —

```
(defun |coerceIntTableOrFunction| (triple t2)
 (let (t1 p tmp1)
  (declare (special |$CoerceTable|))
  (cond
   ((null (|isValidType| t2)) nil)
   ((null (|isLegitimateMode| t2 nil nil)) nil)
   (t
    (setq t1 (|objMode| triple))
    (setq p (assq (car t1) |$CoerceTable|))
    (cond
     ((and p (setq tmp1 (assq (car t2) (cdr p))))
      (cond
       ((eq (third tmp1) '|Identity|) (mkObj (|objVal| triple) t2))
       ((eq (second tmp1) '|total|)
         (or (|coerceByTable| (third tmp1) (|objVal| triple) t1 t2 t)
             (|coerceByFunction| triple t2)))
       (t
         (or (|coerceByTable| (third tmp1) (|objVal| triple) t1 t2 nil)
             (|coerceByFunction| triple t2)))))
     (t (|coerceByFunction| triple t2)))))))
```

———-

25.0.92 defun coerceByTable

[isWrapped p1485]
[unwrap p??]
[mkObjWrap p461]
[isTotalCoerce p??]
[mkObj p460]
[mkq p??]
[$OutputForm p631]
[$coerceFailure p??]
[coerceFailure p??]

— **defun coerceByTable** —

```
(defun |coerceByTable| (fn x t1 t2 isTotalCoerce)
```

```
 (let (c)
  (declare (special |$coerceFailure| |$OutputForm|))
  (cond
   ((equal t2 |$OutputForm|) nil)
   ((|isWrapped| x)
    (setq x (|unwrap| x))
    (setq c (catch '|coerceFailure| (funcall fn x t1 t2)))
    (unless (equal c |$coerceFailure|) (mkObjWrap c t2)))
   (isTotalCoerce (mkObj (list fn x (mkq t1) (mkq t2)) t2))
   (t
    (mkObj (list '|catchCoerceFailure| (mkq fn) x (mkq t1) (mkq t2)) t2)))))
```

———-

25.0.93 defun catchCoerceFailure

This function is funcalled from code constructed by **coerceByTable**. [unwrap p**??**]
[wrap p1104]
[throwKeyedMsgCannotCoerceWithValue p**??**]
[$coerceFailure p**??**]
[coerceFailure p**??**]

— **defun catchCoerceFailure** —

```
(defun |catchCoerceFailure| (fn x t1 t2)
 (let (c)
  (declare (special |$coerceFailure|))
  (setq c (catch '|coerceFailure| (funcall fn x t1 t2)))
  (if (equal c |$coerceFailure|)
   (|throwKeyedMsgCannotCoerceWithValue| (|wrap| (|unwrap| x)) t1 t2)
   c)))
```

———-

25.0.94 defun coerceIntSpecial

[objMode p462]
[coerceInt p659]

— **defun coerceIntSpecial** —

```
(defun |coerceIntSpecial| (triple t2)
 (let (x)
  (when (and (eq (first t2) '|SimpleAlgebraicExtension|)
             (equal (second t2) (|objMode| triple)))
    (unless (setq x (|coerceInt| triple (third t2)))
     (|coerceInt| x t2))))))
```

———-

25.0.95 defun coerceIntByMap

The idea is this: if t1 is D U1 and t2 is D U2, then look for a map: `(U1 -> U2, D U1) -> D U2`. If it exists, then create a function to do the coercion on the element level and call the map function. [objMode p462]
[length p??]
[deconstructT p??]
[nequal p??]
[valueArgsEqual? p679]
[underDomainOf p??]
[member p1108]
[isSubDomain p??]
[sayFunctionSelection p??]
[selectMms1 p??]
[sayFunctionSelectionResult p??]
[compiledLookup p1097]
[evalDomain p993]
[wrapped2Quote p??]
[objVal p462]
[timedEvaluate p??]
[mkObjWrap p461]
[coerceFailure p??]

— **defun coerceIntByMap** —

```
(defun |coerceIntByMap| (triple t2)
 (let (t1 top u1 u2 args mms fun code val)
  (declare (special |$coerceFailure| |$reportBottomUpFlag|))
  (setq t1 (|objMode| triple))
  (cond
   ((equal t2 t1) triple)
   (t
    (setq u2 (|deconstructT| t2)) ; compute t2 first because of Expression
    (cond
      ((eql 1 (|#| u2)) nil)      ; no under domain
      (t
       (setq u1 (|deconstructT| t1))
       (cond
        ((eql 1 (|#| u1)) nil)
        ((nequal (caar u1) (caar u2)) nil) ; constructors not equal
        ((null (|valueArgsEqual?| t1 t2)) nil)
        (t
         ; handle a couple of special cases for subdomains of Integer
         (setq top (caar u1))
         (setq u1 (|underDomainOf| t1))
         (setq u2 (|underDomainOf| t2))
         (cond
          ((and (|member| top
                  '(|List| |Vector| |Segment| |Stream|
                    |UniversalSegment| |Array|))
                (|isSubDomain| u1 u2))
           (mkObj (|objVal| triple) t2))
          (t
```

```
(setq args (list (list '|Mapping| u2 u1) t1))
(when |$reportBottomUpFlag|
 (|sayFunctionSelection| '|map| args t2 nil
                         "coercion facility (map)"))
(setq mms (|selectMms1| '|map| t2 args args nil))
(when |$reportBottomUpFlag|
  (|sayFunctionSelectionResult| '|map| args mms))
(cond
 ((null mms) nil)
 (t
  (setq fun
   (|compiledLookup| '|map| (cdaar mms) (|evalDomain| (caaar mms))))
  (cond
   ((null fun) nil)
   (t
    (cond
     ((equal (car fun) #'|Undef|) nil)
     (t
      ;  now compile a function to do the coercion
      (setq code
       (list 'spadcall
        (list 'cons
         (list 'function  '|coerceIntByMapInner|)
          (mkq (cons u1 u2)))
         (|wrapped2Quote| (|objVal| triple))
          (mkq fun)))
      ; and apply the function
      (setq val (catch '|coerceFailure| (|timedEvaluate| code)))
      (unless (equal val |$coerceFailure|)
       (mkObjWrap val t2)))))))))))))))))
```

———-

25.0.96 defun coerceIntByMapInner

This is a helper function for **coerceIntByMap** which constructs a **spadcall** and then evaluates it. [coerceOrThrowFailure p678]

— **defun coerceIntByMapInner** —

```
(defun |coerceIntByMapInner| (arg g1)
 (|coerceOrThrowFailure| arg (car g1) (cdr g1)))
```

———-

25.0.97 defun coerceOrThrowFailure

[coerceOrRetract p686]
[mkObjWrap p461]
[coercionFailure p679]
[objValUnwrap p462]

— **defun coerceOrThrowFailure** —

```
(defun |coerceOrThrowFailure| (value t1 t2)
 (let (result)
  (or (setq result (|coerceOrRetract| (mkObjWrap value t1) t2))
      (|coercionFailure|))
  (|objValUnwrap| result)))
```

————-

25.0.98 defun coercionFailure

This does a throw on coercion failure. [coerceFailure p**??**]

— **defun coercionFailure** —

```
(defun |coercionFailure| ()
  (declare (special |$coerceFailure|))
  (throw '|coerceFailure| |$coerceFailure|))
```

————-

25.0.99 defun valueArgsEqual?

`[u1,:u2]` gets passed as the "environment", which is why we have this slightly clumsy locution JHD 31.July,1990

This returns true if the object-valued arguments to t1 and t2 are the same under coercion
[getdatabase p1070]
[getConstructorSignature p**??**]
[replaceSharps p1018]
[coerceInt p659]
[mkObjWrap p461]
[algEqual p680]
[objValUnwrap p462]

— **defun valueArgsEqual?** —

```
(defun |valueArgsEqual?| (t1 t2)
 (let (coSig constrSig tl1 tl2 newVal done value trip)
  (setq coSig (cdr (getdatabase (car t1) 'cosig)))
  (setq constrSig (cdr (|getConstructorSignature| (car t1))))
  (setq tl1 (|replaceSharps| constrSig t1))
  (setq tl2 (|replaceSharps| constrSig t2))
  (cond
   ((null (member nil coSig)) t)
   (t
    (setq done nil)
    (setq value t)
    (loop for a1 in (cdr t1) for a2 in (cdr t2) for cs in coSig
          for m1 in tl1 for m2 in tl2 while (not done) do
```

```
          (cond
           ((null cs)
             (setq trip (mkObjWrap a1 m1))
             (setq newVal (|coerceInt| trip m2))
             (cond
              ((null newVal)
                (setq done t)
                (setq value nil))
              ((null (|algEqual| a2 (|objValUnwrap| newVal) m2))
                (setq done t)
                (setq value nil))))))
     value))))
```

———-

25.0.100 defun algEqual

This function sees if 2 objects of the same domain are equal by using the = from the domain. The objects should not be wrapped. [spadcall p**??**]
[compiledLookupCheck p744]
[evalDomain p993]
[$Boolean p630]

— **defun algEqual** —

```
(defun |algEqual| (object1 object2 domain)
 (declare (special |$Boolean|))
 (spadcall object1 object2
  (|compiledLookupCheck| '= (list |$Boolean| '$ '$) (|evalDomain| domain))))
```

———-

25.0.101 defun coerceIntFromUnion

— **defun coerceIntFromUnion** —

```
(defun |coerceIntFromUnion| (object t2)
  (|coerceInt| (|coerceUnion2Branch| object) t2))
```

———-

25.0.102 defun coerceInt2Union

— **defun coerceInt2Union** —

```
(defun |coerceInt2Union| (object union)
 (let (unionDoms t1 val valp noCoerce)
  (declare (special |$String|))
  (setq unionDoms (|stripUnionTags| (cdr union)))
```

```
  (setq t1 (|objMode| object))
  (cond
   ((|member| t1 unionDoms) (|coerceBranch2Union| object union))
   (t
    (setq val (|objVal| object))
    (setq valp (|unwrap| val))
    (cond
     ((and (equal t1 |$String|) (|member| valp unionDoms))
       (|coerceBranch2Union| (mkObj val valp) union))
     (t
      (setq noCoerce t)
      (setq valp nil)
      (loop for d in unionDoms while noCoerce do
       (when (setq valp (|coerceInt| object d)) (setq noCoerce nil)))
      (when valp (|coerceBranch2Union| valp union))))))))
```

———-

25.0.103 defun coerceBranch2Union

[orderUnionEntries p??]
[mkPredList p??]
[stripUnionTags p690]
[position p??]
[keyedSystemError p??]
[objMode p462]
[objVal p462]
[mkObjWrap p461]
[removeQuote p??]
[unwrap p??]
[mkObj p460]

— **defun coerceBranch2Union** —

```
(defun |coerceBranch2Union| (object union)
 (let (predList doms p val tag)
  (setq doms (|orderUnionEntries| (cdr union)))
  (setq predList (|mkPredList| doms))
  (setq doms (|stripUnionTags| doms))
  (setq p (|position| (|objMode| object) doms))
  (cond
   ((equal p (- 1))
     (|keyedSystemError| "The type %1p is not branch of %2p"
      (list (|objMode| object) union)))
   (t
    (setq val (|objVal| object))
    (if (eq (car (setq tag (elt predlist p))) 'eqcar)
     (mkObjWrap (cons (|removeQuote| (third tag)) (|unwrap| val)) union)
     (mkObj val union))))))
```

———-

25.0.104 defun coerceIntAlgebraicConstant

[objMode p462]
[objValUnwrap p462]
[ofCategory p637]
[mkObjWrap p461]
[getConstantFromDomain p682]

— **defun coerceIntAlgebraicConstant** —

```
(defun |coerceIntAlgebraicConstant| (object t2)
 (let (t1 val)
  (setq t1 (|objMode| object))
  (setq val (|objValUnwrap| object))
  (cond
   ((and (|ofCategory| t1 '(|Monoid|))
         (|ofCategory| t2 '(|Monoid|))
         (equal val (|getConstantFromDomain| '(|One|) t1)))
     (mkObjWrap (|getConstantFromDomain| '(|One|) t2) t2))
   ((and (|ofCategory| t1 '(|AbelianMonoid|))
         (|ofCategory| t2 '(|AbelianMonoid|))
         (equal val (|getConstantFromDomain| '(|Zero|) t1)))
     (mkObjWrap (|getConstantFromDomain| '(|Zero|) t2) t2)))))
```

———-

25.0.105 defun getConstantFromDomain

The function **getConstantFromDomain** is used to look up the constants 0 and 1 from the given domainForm.

If isPartialMode returns true then the domain modemap contains the constant `$EmptyMode` which indicates that the domain is not fully formed. In this case we return nil.

[isPartialMode p638]
[opOf p??]
[lassoc p??]
[getOperationAlistFromLisplib p119]
[getConstantFromDomain p682]
[throwKeyedMsg p??]
[spadcall p??]
[compiledLookupCheck p744]
[evalDomain p993]

— **defun getConstantFromDomain** —

```
(defun |getConstantFromDomain| (form domainForm)
 (let (key entryList)
  (unless (|isPartialMode| domainForm)
   (setq key (|opOf| form))
   (setq entryList
    (lassoc key (|getOperationAlistFromLisplib| (car domainForm))))
   (cond
```

```
    ((null (eq (cdr entryList) nil))
     (cond
      ((eq key '|One|)  (|getConstantFromDomain| (list '|1|) domainForm))
      ((eq key '|Zero|) (|getConstantFromDomain| (list '|0|) domainForm))
      (t
        (|throwKeyedMsg| "No such constant %1 in domain %2p ."
         (list form domainForm)))))
    (t
     ; there should be exactly one item under this key of that form
     (spadcall
      (|compiledLookupCheck| key (caar entryList)
        (|evalDomain| domainForm))))))))
```

———-

25.0.106 defun compareTypeLists

Rreturns true if every type in tl1 is equal or is a subdomain of the corresponding type in tl2

— **defun compareTypeLists** —

```
(defun |compareTypeLists| (tl1 tl2)
 (not
  (loop for t1 in tl1 for t2 in tl2
   do (when (null (|isEqualOrSubDomain| t1 t2)) (return t)))))
```

———-

25.0.107 defun coerceIntX

Try to coerce a (List (None)) into a different domain [unwrap p**??**]
[underDomainOf p**??**]
[coerceInt p659]
[mkObjWrap p461]

— **defun coerceIntX** —

```
(defun |coerceIntX| (val t1 t2)
 (let (t0)
   (when (and (equal t1 '(|List| (|None|)))
              (null (|unwrap| val))
              (setq t0 (|underDomainOf| t2)))
    (|coerceInt| (mkObjWrap val (list '|List| t0)) t2))))
```

———-

25.0.108 defun coerceSubDomain

[getdatabase p1070]
[coerceSubDomain p683]
[coerceImmediateSubDomain p684]

— defun coerceSubDomain —

```
(defun |coerceSubDomain| (val tSuper tSub)
 (let (super)
  (unless (eq val '|$fromCoerceable$|)
   (setq super (getdatabase (car tSub) 'superdomain))
   (cond
    ((equal (car super) tSuper)
      (|coerceImmediateSubDomain| val tSuper tSub (second super)))
    ((|coerceSubDomain| val tSuper (car super))
      (|coerceImmediateSubDomain| val (car super) tSub (second super)))))))
```

———-

25.0.109 defun coerceImmediateSubDomain

[getSubDomainPredicate p684]

— defun coerceImmediateSubDomain —

```
(defun |coerceImmediateSubDomain| (val tSuper tSub pred)
 (when (funcall (|getSubDomainPredicate| tSuper tSub pred) val nil)
  (mkObj val tSub)))
```

———-

25.0.110 defun getSubDomainPredicate

[msubst p??]
[removeZeroOne p??]
[interpret p312]
[mkAtree p??]
[transferPropsToNode p??]
[selectLocalMms p??]
[hput p1105]
[$env p284]
[$superHash p??]
[$Boolean p630]
[$InteractiveFrame p34]

— defun getSubDomainPredicate —

```
(defun |getSubDomainPredicate| (tSuper tSub pred)
 (let (|$env| name decl arg predp defn op predfn)
  (declare (special |$env| |$superHash| |$Boolean| |$InteractiveFrame|))
  (setq |$env| |$InteractiveFrame|)
  (cond
   ((setq predfn (hget |$superHash| (cons tSuper tSub))) predfn)
   (t
    (setq name (gensym))
    (setq decl (list '|:| name (list '|Mapping| |$Boolean| tSuper)))
```

```
    (|interpret| decl nil)
    (setq arg (gensym))
    (setq predp (msubst arg '|#1| pred))
    (setq defn
     (list 'def (list name arg) '(nil nil) '(nil nil) (|removeZeroOne| predp)))
    (|interpret| defn nil)
    (setq op (|mkAtree| name))
    (|transferPropsToNode| name op)
    (setq predfn (cadar (|selectLocalMms| op name (list tSuper) |$Boolean|)))
    (hput |$superHash| (cons tSuper tSub) predfn)
    predfn))))
```

———-

25.0.111 defun absolutelyCanCoerceByCheating

This typically involves subdomains and towers where the only difference is a subdomain
[isEqualOrSubDomain— p**??**]
[deconstructT p**??**]
[nequal p**??**]
[absolutelyCanCoerceByCheating p685]
[$SingleInteger p635]
[$Integer p632]

— defun absolutelyCanCoerceByCheating —

```
(defun |absolutelyCanCoerceByCheating| (t1 t2)
 (let (let1 let2)
  (declare (special |$Integer| |$SingleInteger|))
  (cond
   ((|isEqualOrSubDomain| t1 t2) t)
   ((and (equal t1 |$SingleInteger|) (equal t2 |$Integer|)) t)
   ((or (atom t1) (atom t2)) nil)
   (t
    (setq let1 (|deconstructT| t1))
    (setq let2 (|deconstructT| t2))
    (cond
     ((and (equal (car let1) '(|Stream|))
           (equal (car let2) '(|InfiniteTuple|)))
      (cond
       ((nequal (|#| (cdr let1)) (|#| (cdr let2))) nil)
       (t
        (every #'identity
         (loop for x1 in (cdr let1) for x2 in (cdr let2) collect
          (|absolutelyCanCoerceByCheating| x1 x2))))))
     ((nequal (car let1) (car let2)) nil)
     ((nequal (|#| (cdr let1)) (|#| (cdr let2))) nil)
     (t
      (every #'identity
       (loop for x1 in (cdr let1) for x2 in (cdr let2) collect
        (|absolutelyCanCoerceByCheating| x1 x2)))))))))
```

———-

25.0.112 defun coerceOrRetract

[coerceInteractive p658]
[retract p1137]

— **defun coerceOrRetract** —

```
(defun |coerceOrRetract| (z m)
 (prog (tp tt ans)
  (return
    (cond
     ((setq tp (|coerceInteractive| z m)) tp)
     (t
      (setq tt z)
      (setq ans nil)
      (do () (nil nil)
       (cond
        (ans (return ans))
        (t
         (setq tt (|retract| tt))
         (cond
          ((eq tt '|failed|) (return ans))
          (t (setq ans (|coerceInteractive| tt m)))))))
      ans)))))
```

———-

25.0.113 defun retract2Specialization

Handle some specialization retraction cases, like matrices [objVal p462]
[unwrap p**??**]
[objMode p462]
[mkObjWrap p461]
[coerceUnion2Branch p690]
[coerceInt p659]
[remdup p**??**]
[varsInPoly p**??**]
[mkObj p460]
[member p1108]
[retract p1137]
[objValUnwrap p462]
[objMode p462]
[resolveTypeListAny p**??**]
[isRectangularList p**??**]
[get p**??**]
[isPartialMode p638]
[$e p285]
[$QuotientField p634]

[$Symbol p634]
[$Integer p632]
[$Any p630]
[$NonNegativeInteger p633]
[$PositiveInteger p633]

— **defun retract2Specialization** —

```
(defun |retract2Specialization| (object)
 (prog (val type dom obj dp bad vl tl ep vlp n D num den valp m)
 (declare (special |$e| |$QuotientField| |$Symbol| |$Integer| |$Any|
                   |$NonNegativeInteger| |$PositiveInteger|))
  (return
   (seq
    (progn
     (setq val (|objVal| object))
     (setq valp (|unwrap| val))
     (setq type (|objMode| object))
     (cond
      ; type is Any
      ((equal type |$Any|)
        (setq dom (car valp))
        (setq obj (cdr valp))
        (mkObjWrap obj dom))
      ; type is ['Union,:unionDoms]
      ((eq (car type) '|Union|)
        (|coerceUnion2Branch| object))
      ; type is Symbol
      ((equal type |$Symbol|)
        (mkObjWrap 1 (list '|OrderedVariableList| (list valp))))
      ; type is ['OrderedVariableList,var]
      ((eq (car type) '|OrderedVariableList|)
        (|coerceInt|
         (mkObjWrap (elt (second type) (- valp 1)) |$Symbol|)
         '(|Polynomial| (|Integer|))))
      ; type is ['Polynomial,d]
      ((eq (car type) '|Polynomial|)
        (cond
         ((eql (car valp) 1)
           (when (eql 1 (|#| (remdup (|varsInPoly| valp))))
             (|coerceInt| object
              (list '|UnivariatePolynomial| (second valp)  (second type)))))
         ((eql (car valp) 0) (|coerceInt| object (second type)))
         (t nil)))
      ; type is ['Matrix,d]
      ((eq (car type) '|Matrix|)
        (setq n (|#| valp))
        (setq m (|#| (elt valp 0)))
        (cond
         ((= n m) (mkObj val (list '|SquareMatrix| n (second type))))
         (t (mkObj val (list '|RectangularMatrix| n m (second type))))))
      ; type is ['RectangularMatrix,n,m,d]
      ((eq (first type) '|RectangularMatrix|)
        (setq n (second type))
```

```
  (setq m (third type))
  (setq d (fourth type))
  (when (eql n m) (mkObj val (list '|SquareMatrix| n d))))
; type is [agg,d] agg is |Vector|,|Segment|, or |UniversalSegment|
((|member| (first type) '(|Vector| |Segment| |UniversalSegment|))
  (cond
   ((equal (second type) |$PositiveInteger|)
    (mkObj val (cons (first type) (list |$NonNegativeInteger|))))
   ((equal (second type) |$NonNegativeInteger|)
    (mkObj val (list (first type) |$Integer|)))))
; type is ['Array,bds,d]
((eq (first type) '|Array|)
 (cond
  ((equal (third type) |$PositiveInteger|)
   (mkObj val (list '|Array| (second type) |$NonNegativeInteger|)))
  ((equal (third type) |$NonNegativeInteger|)
   (mkObj val (list '|Array| (second type) |$Integer|)))))
; type is ['List,d]
((eq (car type) '|List|)
 (setq d (second type))
 (setq dp (second d))
 (cond
  ; type isnt ['List,dp]
  ((null (eq (car d) '|List|))
    (cond
     ((equal d |$PositiveInteger|)
       (mkObj val (list '|List| |$NonNegativeInteger|)))
     ((equal d |$NonNegativeInteger|)
       (mkObj val (list '|List| |$Integer|)))
     ((null valp) nil)
     (t
      (setq vl nil)
      (setq tl nil)
      (setq bad nil)
      (loop for e in valp while (not bad) do
       (cond
        ((equal (setq ep (|retract| (mkObjWrap e d))) '|failed|)
          (setq bad t))
        (t
         (setq vl (cons (|objValUnwrap| ep) vl))
         (setq tl (cons (|objMode| ep) tl)))))
      (cond
       (bad nil)
       ((equal (setq m (|resolveTypeListAny| tl)) d) nil)
       ((equal d m) nil)
       (t
        (setq vlp nil)
        (setq ep t)
        (loop for e in vl for tt in tl while ep do
         (cond
          ((equal tt m) (setq vlp (cons e vlp)))
          (t
           (setq ep (|coerceInt| (mkObjWrap e tt) m))
           (when ep (setq vlp (cons (|objValUnwrap| ep) vlp))))))))
```

```
          (mkObjWrap vlp (list '|List| m)))))))
    ((equal dp |$PositiveInteger|)
      (mkObj val (list '|List| (list '|List| |$NonNegativeInteger|))))
    ((equal dp |$NonNegativeInteger|)
      (mkObj val (list '|List| (list '|List| |$Integer|))))
    ((or (eq (car dp) '|Variable|)
         (eq (car dp) '|OrderedVariableList|))
      (|coerceInt| object (list '|List| (list '|List| |$Symbol|))))
    (t
     (setq n (|#| valp))
     (setq m (|#| (elt valp 0)))
     (cond
      ((null (|isRectangularList| valp n m)) nil)
      (t (|coerceInt| object (list '|Matrix| dp)))))))
; type is ['Expression,d]
((eq (car type) '|Expression|)
  (setq num (car valp))
  (setq den (cdr valp))
  (cond
   ((null (equal (car num) 0)) nil)
   ((null (equal (car den) 0)) nil)
   (t
    (mkObjWrap (cons (cdr num) (cdr den))
     (list |$QuotientField| (second type))))))
; type is ['SimpleAlgebraicExtension,k,rep,.]
; try to retract as an element of rep and see if we can get an element of k
((eq (car type) '|SimpleAlgebraicExtension|)
  (setq valp (|retract| (mkObj val (third type))))
  (do ()
      ((null (and (nequal valp '|failed|)
                  (nequal (|objMode| valp) (second type))))
        nil)
   (setq valp (|retract| valp)))
  (unless (equal valp '|failed|) valp))
; type is ['UnivariatePuiseuxSeries,coef,var,cen]
((eq (car type) '|UnivariatePuiseuxSeries|)
  (|coerceInt| object
    (list '|UnivariateLaurentSeries|
     (second type) (third type) (fourth type))))
; type is ['UnivariateLaurentSeries,coef,var,cen]
((eq (car type) '|UnivariateLaurentSeries|)
  (|coerceInt| object
   (list '|UnivariateTaylorSeries|
    (second type) (third type) (fourth type))))
  ; type is ['FunctionCalled,name]
((eq (car type) '|FunctionCalled|)
  (cond
   ((null (setq m (|get| (second type) '|mode| |$e|))) nil)
   ((|isPartialMode| m) nil)
   (t (mkObj val m))))
(t nil)))))))
```

———-

25.0.114 defun coerceUnion2Branch

[orderUnionEntries p??]
[objMode p462]
[mkPredList p??]
[stripUnionTags p690]
[objValUnwrap p462]
[evalSharpOne p690]
[mkObj p460]
[objVal p462]

— **defun coerceUnion2Branch** —

```
(defun |coerceUnion2Branch| (object)
 (let (predList doms valp predicate targetType)
  (setq doms (|orderUnionEntries| (cdr (|objMode| object))))
  (setq predList (|mkPredList| doms))
  (setq doms (|stripUnionTags| doms))
  (setq valp (|objValUnwrap| object))
  (loop for typ in doms for pred in predList while (not targetType) do
   (when (|evalSharpOne| pred valp)
    (setq predicate pred)
    (setq targetType typ)))
  (cond
   ((null targetType)
     (|keyedSystemError| "Cannot determine branch of Union." nil))
   ((eq (car predicate) 'eqcar) (mkObjWrap (cdr valp) targetType))
   (t (mkObj (|objVal| object) targetType)))))
```

———-

25.0.115 defun stripUnionTags

— **defun stripUnionTags** —

```
(defun |stripUnionTags| (doms)
 (loop for dom in doms
  collect (if (eq (first dom) '|:|) (third dom) dom)))
```

———-

25.0.116 defun evalSharpOne

— **defun evalSharpOne 0** —

```
(defun |evalSharpOne| (x |#1|)
 (declare (special |#1|))
 (eval `(let() (declare (special |#1|)) ,x)))
```

———-

25.0.117 defun retractUnderDomain

[underDomainOf p??]
[deconstructT p??]
[nequal p??]
[constructT p??]
[coerceInt p659]

— defun retractUnderDomain —

```
(defun |retractUnderDomain| (object type underDomain)
 (let (ud let1 typep objectp)
  (cond
   ((null (setq ud (|underDomainOf| underDomain))) '|failed|)
   (t
    (setq let1 (|deconstructT| type))
    (cond
     ((nequal 1 (|#| (cdr let1))) '|failed|)
     ((nequal 1 (|#| (car let1))) '|failed|)
     (t
      (setq typep (|constructT| (car let1) (list ud)))
      (cond
       ((setq objectp (|coerceInt| object typep)) objectp)
       (t '|failed|)))))))))
```

———-

25.0.118 defun coerceRetract

[objValUnwrap p462]
[objMode p462]
[isEqualOrSubDomain p655]
[mkObjWrap p461]
[retractByFunction p692]
[getl p1110]
[canFuncall? p1108]
[$coerceFailure p??]
[$SingleInteger p635]
[$OutputForm p631]
[$Symbol p634]
[$Integer p632]
[coerceFailure p??]

— defun coerceRetract —

```
(defun |coerceRetract| (object t2)
 (let (val t1 fun c)
  (declare (special |$coerceFailure| |$OutputForm| |$Symbol| |$Integer|
           |$SingleInteger|))
```

```
  (cond
   ((eq (setq val (|objValUnwrap| object)) '|$fromCoerceable$|) nil)
   (t
    (setq t1 (|objMode| object))
    (cond
     ((equal t2 |$OutputForm|) nil)
     ((and (|isEqualOrSubDomain| t1 |$Integer|)
           (equal t2 |$SingleInteger|)
           (typep val 'fixnum))
        (mkObjWrap val t2))
     ((equal t1 |$Integer|) nil)
     ((equal t1 |$Symbol|) nil)
     ((equal t1 |$OutputForm|) nil)
     ((setq c (|retractByFunction| object t2)) c)
     ((consp t1)
       (setq fun
        (or (getl (car t1) '|retract|)
            (intern (concat "retract" (princ-to-string (car t1))))))
       (when (canFunCall? fun)
        (put (car t1) '|retract| fun)
        (setq c (catch '|coerceFailure| (funcall fun object t2)))
        (unless (equal c |$coerceFailure|) c))))))))
```

———-

25.0.119 defun retractByFunction

[objValUnwrap p462]
[sayFunctionSelection p??]
[findFunctionInDomain p??]
[orderMms p??]
[sayFunctionSelectionResult p??]
[evalDomain p993]
[compiledLookup p1097]
[coerceUnion2Branch p690]
[mkObjWrap p461]
[spadcall p??]
[objMode p462]
[$reportBottomUpFlag p898]
[$dollar p??]

— **defun retractByFunction** —

```
(defun |retractByFunction| (object u)
 (let (|$reportBottomUpFlag| $ tt val target funName mms dcVector fun objectp)
 (declare (special |$reportBottomUpFlag| $))
  (setq tt (|objMode| object))
  (setq val (|objValUnwrap| object))
  (setq target (list '|Union| u "failed"))
  (setq funName '|retractIfCan|)
  (when |$reportBottomUpFlag|
    (|sayFunctionSelection| funName (list tt) target
```

```
    nil "coercion facility (retraction)"))
(when
 (setq mms
  (append
   (|findFunctionInDomain| funName tt target (list tt) (list tt) nil t)
   (|findFunctionInDomain| funName u target (list tt) (list tt) nil t)))
  (setq mms (|orderMms| funName mms (list tt) (list tt) target)))
(when |$reportBottomUpFlag|
  (|sayFunctionSelectionResult| funName (list tt) mms))
(when mms
 (setq dcVector (|evalDomain| (caaar mms)))
 (setq fun (|compiledLookup| funName (list target tt) dcVector))
 (cond
  ((null fun) nil)
  ((equal (car fun) #'|Undef|) nil)
  (t
   (setq $ dcVector)
   (setq objectp
    (|coerceUnion2Branch| (mkObjWrap (spadcall val fun) target)))
   (when (equal u (|objMode| objectp)) objectp))))))
```

———-

Chapter 26

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" (26.3.2 p 731) command
- The "boot" (26.4 p 734) command
- The "browse" (26.5 p 735) command
- The "cd" (26.11 p 739) command
- The "clear" (26.12.3 p 741) command
- The "close" (26.13.3 p 751) command
- The "compile" (26.14 p 753) command
- The "copyright" (26.15.2 p 760) command
- The "credits" (26.16.2 p 762) command
- The "display" (26.18.3 p 768) command
- The "edit" (26.19.2 p 776) command
- The "fin" (26.20.2 p 779) command
- The "frame" (3.5.1 p 22) command
- The "help" (26.21.2 p 782) command
- The "history" (26.23.11 p 791) command
- The "lisp" (26.27 p 831) command
- The "library" (36.1.30 p 1073) command
- The "license" (26.26.2 p 830) command
- The "load" (?? p ??) command

- The “ltrace” (26.28.2 p 832) command
- The “pquit” (26.29.2 p 834) command
- The “quit” (26.30.2 p 836) command
- The “read” (26.31.2 p 838) command
- The “regress” (26.32.4 p 846) command
- The “savesystem” (26.33.3 p 854) command
- The “set” (26.51.1 p 962) command
- The “show” (26.52.2 p 967) command
- The “spool” (26.53 p 980) command
- The “summary” (26.54.2 p 981) command
- The “synonym” (26.55.2 p 984) command
- The “system” (26.56 p 987) command
- The “tangle” (26.57 p 988) command
- The “trace” (7.4.1 p 67) command
- The “trademark” (26.58 p 990) command
- The “undo” (26.59 p 991) command
- The “what” (26.61.3 p 1007) command
- The “with” (?? p ??) command
- The “workfiles” (26.62.2 p 1015) command

26.1 Variables Used

26.1.1 defvar $systemCommands

— **initvars** —

```
(defvar |$systemCommands| nil)
```

————-

— **postvars** —

```
(eval-when (eval load)
 (setq |$systemCommands|
 '(
   (|abbreviations|                . |compiler|   )
   (|boot|                         . |development|)
   (|browse|                       . |development|)
   (|cd|                           . |interpreter|)
   (|clear|                        . |interpreter|)
   (|close|                        . |interpreter|)
   (|compiler|                     . |compiler|   )
```

```
   (|copyright|                    . |interpreter|)
   (|credits|                      . |interpreter|)
   (|describe|                     . |interpreter|)
   (|display|                      . |interpreter|)
   (|edit|                         . |interpreter|)
   (|fin|                          . |development|)
   (|frame|                        . |interpreter|)
   (|help|                         . |interpreter|)
   (|history|                      . |interpreter|)
   (|lisp|                         . |development|)
   (|library|                      . |interpreter|)
   (|license|                      . |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|)
   (|tangle|                       . |interpreter|)
   (|trace|                        . |interpreter|)
   (|trademark|                    . |interpreter|)
   (|undo|                         . |interpreter|)
   (|what|                         . |interpreter|)
   (|with|                         . |interpreter|)
   (|workfiles|                    . |development|)
 )))
```

———-

26.1.2 defvar $syscommands

This table is used to look up a symbol to see if it might be a command.

— **initvars** —

```
(defvar $syscommands nil)
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq $syscommands (mapcar #'car |$systemCommands|)))
```

———-

26.1.3 defvar $noParseCommands

This is a list of the commands which have their arguments passed verbatim. Certain functions, such as the lisp function need to be able to handle all kinds of input that will not be acceptable to the interpreter.

— **initvars** —

```
(defvar |$noParseCommands| nil)
```

————-

— **postvars** —

```
(eval-when (eval load)
 (setq |$noParseCommands|
 '(|boot| |copyright| |credits| |fin| |license| |lisp| |pquit| |quit|
   |synonym| |system| |trademark| )))
```

————-

26.2 Functions

26.2.1 defun handleNoParseCommands

The system commands given by the global variable `$noParseCommands` require essentially no preprocessing/parsing of their arguments. Here we dispatch the functions which implement these commands.

There are four standard commands which receive arguments

- boot
- lisp
- synonym
- system

There are six standard commands which do not receive arguments –

- quit
- fin
- pquit
- credits
- copyright
- trademark

As these commands do not necessarily exhaust those mentioned in `$noParseCommands`, we provide a generic dispatch based on two conventions: commands which do not require an argument name themselves, those which do have their names prefixed by “np”. This makes it possible to dynamically define new system commands provided you handle the argument parsing.

26.2.2 defun Handle a top level command

[concat p1107]
[expand-tabs p??]
[processSynonyms p293]
[substring p293]
[getFirstWord p719]
[unAbbreviateKeyword p719]
[member p1108]
[handleNoParseCommands p698]
[splitIntoOptionBlocks p700]
[handleTokensizeSystemCommands p700]
[handleParsedSystemCommands p718]
[$tokenCommands p724]
[$noParseCommands p698]
[line p1027]

— **defun doSystemCommand** —

```
(defun |doSystemCommand| (string)
 (let (line tok unab optionList)
 (declare (special line |$tokenCommands| |$noParseCommands|))
  (setq string (concat ")" (expand-tabs string)))
  (setq line string)
  (|processSynonyms|)
  (setq string line)
  (setq string (substring string 1 nil))
  (cond
   ((string= string "") nil)
   (t
    (setq tok (|getFirstWord| string))
    (cond
     (tok
      (setq unab (|unAbbreviateKeyword| tok))
      (cond
       ((|member| unab |$noParseCommands|)
        (|handleNoParseCommands| unab string))
       (t
        (setq optionList (|splitIntoOptionBlocks| string))
        (cond
         ((|member| unab |$tokenCommands|)
          (|handleTokensizeSystemCommands| unab optionList))
         (t
          (|handleParsedSystemCommands| unab optionList)
          nil)))))
     (t nil))))))
```

————-

26.2.3 defun Split block into option block

[stripSpaces p721]

— **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)))
```

———-

26.2.4 defun Tokenize a system command

[dumbTokenize p717]
[tokTran p718]
[systemCommand p700]

— **defun handleTokensizeSystemCommands** —

```
(defun |handleTokensizeSystemCommands| (unabr optionList)
 (declare (ignore unabr))
 (let (parcmd)
  (setq optionList (mapcar #'(lambda (x) (|dumbTokenize| x)) optionList))
  (setq parcmd
    (mapcar #'(lambda (opt) (mapcar #'(lambda (tok) (|tokTran| tok)) opt))
     optionLIst))
  (when parcmd (|systemCommand| parcmd))))
```

———-

26.2.5 defun Handle system commands

You can type ")?" and see trivial help information. You can type ")? compile" and see compiler related information

[selectOptionLC p728]
[helpSpad2Cmd p782]
[selectOption p728]
[commandsForUserLevel p701]
[$options p63]
[$e p285]
[$systemCommands p696]
[$syscommands p697]
[$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)))))
```

———-

26.2.6 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 p703]

— **defun commandsForUserLevel** —

```
(defun |commandsForUserLevel| (arg)
 (let (c)
  (dolist (pair arg)
   (when (|satisfiesUserLevel| (cdr pair))
    (setq c (cons (car pair) c))))
  (nreverse c)))
```

———-

26.2.7 defun No command begins with this string

[commandErrorMessage p702]

— **defun commandError** —

```
(defun |commandError| (x u)
 (|commandErrorMessage| '|command| x u))
```

———-

26.2.8 defun No option begins with this string

[commandErrorMessage p702]

— **defun optionError** —

```
(defun |optionError| (x u)
 (|commandErrorMessage| '|option| x u))
```

———-

26.2.9 defvar $oldline

— **initvars** —

```
(defvar $oldline nil "used to output command lines")
```

———-

26.2.10 defun No command/option begins with this string

[commandAmbiguityError p705]
[sayKeyedMsg p39]
[terminateSystemCommand p704]
[$oldline p702]
[line p1027]

— **defun commandErrorMessage** —

```
(defun |commandErrorMessage| (kind x u)
 (declare (special $oldline line))
  (setq $oldline line)
  (if u
   (|commandAmbiguityError| kind x u)
   (progn
    (|sayKeyedMsg| "No %1 begins with %2 ." (list kind x))
    (|terminateSystemCommand|))))
```

———-

26.2.11 defun Option not available at this user level

[userLevelErrorMessage p703]

— **defun optionUserLevelError** —

```
(defun |optionUserLevelError| (x u)
 (|userLevelErrorMessage| '|option| x u))
```

———-

26.2.12 defun Command not available at this user level

[userLevelErrorMessage p703]

— **defun commandUserLevelError** —

```
(defun |commandUserLevelError| (x u)
 (|userLevelErrorMessage| '|command| x u))
```

———-

26.2.13 defun Command not available error message

[commandAmbiguityError p705]
[sayKeyedMsg p39]
[terminateSystemCommand p704]
[$UserLevel p961]

— **defun userLevelErrorMessage** —

```
(defun |userLevelErrorMessage| (kind x u)
 (declare (special |$UserLevel|))
 (if u
  (|commandAmbiguityError| kind x u)
  (progn
   (|sayKeyedMsg|
    "Your %1 is ambiguous. The following are abbreviated by %2 :"
     (list |$UserLevel| kind))
   (|terminateSystemCommand|))))
```

———-

26.2.14 defun satisfiesUserLevel

[$UserLevel p961]

— **defun satisfiesUserLevel 0** —

```
(defun |satisfiesUserLevel| (x)
 (declare (special |$UserLevel|))
```

```
 (cond
  ((eq x '|interpreter|) t)
  ((eq |$UserLevel| '|interpreter|) nil)
  ((eq x '|compiler|) t)
  ((eq |$UserLevel| '|compiler|) nil)
  (t t)))
```

———-

26.2.15 defun hasOption

[stringPrefix? p1254]
[pname p1106]

— **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))
```

———-

26.2.16 defun terminateSystemCommand

[tersyscommand p704]

— **defun terminateSystemCommand** —

```
(defun |terminateSystemCommand| nil (tersyscommand))
```

———-

26.2.17 defun Terminate a system command

[spadThrow p??]

— **defun tersyscommand** —

```
(defun tersyscommand ()
 (let (chr tok)
  (fresh-line)
  (setq chr 'endoflinechr)
  (setq tok 'end_unit)
  (|spadThrow|)))
```

———-

26.2.18 defun commandAmbiguityError

[sayKeyedMsg p39]
[sayMSG p40]
[bright p??]
[terminateSystemCommand p704]

— **defun commandAmbiguityError** —

```
(defun |commandAmbiguityError| (kind x u)
 (|sayKeyedMsg|
  "Your %1 is ambiguous. The following are abbreviated by %2 :"
   (list kind x))
 (dolist (a u) (|sayMSG| (cons "     " (|bright| a))))
 (|terminateSystemCommand|))
```

———-

26.2.19 defun getParserMacroNames

The `$pfMacros` is a list of all of the user-defined macros.

[$pfMacros p352]

— **defun getParserMacroNames 0** —

```
(defun |getParserMacroNames| ()
 (declare (special |$pfMacros|))
 (remove-duplicates (mapcar #'car |$pfMacros|)))
```

———-

26.2.20 defun clearParserMacro

Note that if a macro is defined twice this will clear the last instance. Thus:

```
  a ==> 3
  a ==> 4
  )d macros
  a ==> 4
  )clear prop a
  )d macros
  a ==> 3
  )clear prop a
  )d macros
  nil
```

[ifcdr p??]
[assoc p??]
[remalist p??]
[$pfMacros p352]

— **defun clearParserMacro** —

```
(defun |clearParserMacro| (macro)
 (declare (special |$pfMacros|))
 (when (ifcdr (|assoc| macro |$pfMacros|))
  (setq |$pfMacros| (remalist |$pfMacros| macro))))
```

———-

26.2.21 defun displayMacro

[isInterpMacro p??]
[sayBrightly p??]
[bright p??]
[concat p1107]
[object2String p??]
[mathprint p??]
[$op p??]

— **defun displayMacro** —

```
(defun |displayMacro| (name)
 (let (|$op| m body args)
 (declare (special |$op|))
  (setq m (|isInterpMacro| name))
  (cond
   ((null m)
    (|sayBrightly|
     (cons "  " (append (|bright| name)
                       (cons "is not an interpreter macro." nil)))))
   (t
    (setq |$op| (concat "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)))))))
```

———-

26.2.22 defun displayWorkspaceNames

[getInterpMacroNames p??]
[getParserMacroNames p705]
[sayMessage p??]
[msort p??]
[getWorkspaceNames p707]
[sayAsManyPerLineAsPossible p??]
[sayBrightly p??]

[setdifference p??]

— defun displayWorkspaceNames —

```
(defun |displayWorkspaceNames| ()
 (let (pmacs names imacs)
  (setq imacs (|getInterpMacroNames|))
  (setq pmacs (|getParserMacroNames|))
  (|sayMessage| "Names of User-Defined Objects in the Workspace:")
  (setq names (msort (append (|getWorkspaceNames|) pmacs)))
  (if names
   (|sayAsManyPerLineAsPossible| (mapcar #'|object2String| names))
   (|sayBrightly| "   * None *"))
  (setq imacs (setdifference imacs pmacs))
  (when imacs
   (|sayMessage| "Names of System-Defined Objects in the Workspace:")
   (|sayAsManyPerLineAsPossible| (mapcar #'|object2String| imacs)))))
```

———-

26.2.23 defun getWorkspaceNames

```
;getWorkspaceNames() ==
;  NMSORT [n for [n,:.] in CAAR $InteractiveFrame |
;    (n ^= "--macros--" and n^= "--flags--")]
```

[nmsort p??]
[$InteractiveFrame p34]

— defun getWorkspaceNames —

```
(defun |getWorkspaceNames| ()
 (declare (special |$InteractiveFrame|))
 (nmsort (loop for g2 in (caar |$InteractiveFrame|) collect (car g2))))
```

———-

26.2.24 defun fixObjectForPrinting

The `$msgdbPrims` variable is set to:

```
(|%b| |%d| |%l| |%i| |%u| %U |%n| |%x| |%ce| |%rj|
 "%U" "%b" "%d" "%l" "%i" "%u" "%U" "%n" "%x" "%ce" "%rj")
```

[object2Identifier p??]
[member p1108]
[concat p1107]
[pname p1106]
[$msgdbPrims p38]

— defun fixObjectForPrinting —

```
(defun |fixObjectForPrinting| (v)
 (let (vp)
```

```
(declare (special |$msgdbPrims|))
 (setq vp (|object2Identifier| v))
 (cond
  ((eq vp '%) "\\%")
  ((|member| vp |$msgdbPrims|) (concat "\\" (pname vp)))
  (t v))))
```

26.2.25 defun displayProperties,sayFunctionDeps

```
;displayProperties(option,l) ==
;  $dependentAlist : local := nil
;  $dependeeAlist  : local := nil
;  [opt,:vl]:= (l or ['properties])
;  imacs := getInterpMacroNames()
;  pmacs := getParserMacroNames()
;  macros := REMDUP append(imacs, pmacs)
;  if vl is ['all] or null vl then
;    vl := MSORT append(getWorkspaceNames(),macros)
;  if $frameMessages then sayKeyedMsg("S2IZ0065",[$interpreterFrameName])
;  null vl =>
;    null $frameMessages => sayKeyedMsg("S2IZ0066",NIL)
;    sayKeyedMsg("S2IZ0067",[$interpreterFrameName])
;  interpFunctionDepAlists()
;  for v in vl repeat
;    isInternalMapName(v) => 'iterate
;    pl := getIProplist(v)
;    option = 'flags =>     getAndSay(v,"flags")
;    option = 'value =>     displayValue(v,getI(v,'value),nil)
;    option = 'condition => displayCondition(v,getI(v,"condition"),nil)
;    option = 'mode =>      displayMode(v,getI(v,'mode),nil)
;    option = 'type =>      displayType(v,getI(v,'value),nil)
;    option = 'properties =>
;      v = "--flags--" => nil
;      pl is [ ['cacheInfo,:.],:.] => nil
;      v1 := fixObjectForPrinting(v)
;      sayMSG ['"Properties of",:bright prefix2String v1,'":"]
;      null pl =>
;        v in pmacs =>
;            sayMSG '"   This is a user-defined macro."
;            displayParserMacro v
;        isInterpMacro v =>
;            sayMSG '"   This is a system-defined macro."
;            displayMacro v
;        sayMSG '"   none"
;      propsSeen:= nil
;      for [prop,:val] in pl | ^MEMQ(prop,propsSeen) and val repeat
;        prop in '(alias generatedCode IS_-GENSYM mapBody localVars) =>
;          nil
;        prop = 'condition =>
;          displayCondition(prop,val,true)
;        prop = 'recursive =>
```

```
;          sayMSG '"   This is recursive."
;        prop = 'isInterpreterFunction =>
;          sayMSG '"   This is an interpreter function."
;          sayFunctionDeps v where
;            sayFunctionDeps x ==
;              if dependents := GETALIST($dependentAlist,x) then
;                null rest dependents =>
;                  sayMSG ['"   The following function or rule ",
;                    '"depends on this:",:bright first dependents]
;                sayMSG
;                  '"   The following functions or rules depend on this:"
;                msg := ["%b",'"     "]
;                for y in dependents repeat msg := ['" ",y,:msg]
;                sayMSG [:nreverse msg,"%d"]
;              if dependees := GETALIST($dependeeAlist,x) then
;                null rest dependees =>
;                  sayMSG ['"   This depends on the following function ",
;                    '"or rule:",:bright first dependees]
;                sayMSG
;                  '"   This depends on the following functions or rules:"
;                msg := ["%b",'"     "]
;                for y in dependees repeat msg := ['" ",y,:msg]
;                sayMSG [:nreverse msg,"%d"]
;        prop = 'isInterpreterRule =>
;          sayMSG '"   This is an interpreter rule."
;          sayFunctionDeps v
;        prop = 'localModemap =>
;          displayModemap(v,val,true)
;        prop = 'mode =>
;          displayMode(prop,val,true)
;        prop = 'value =>
;          val => displayValue(v,val,true)
;        sayMSG ['"   ",prop,'":  ",val]
;        propsSeen:= [prop,:propsSeen]
;    sayKeyedMsg("S2IZ0068",[option])
;  terminateSystemCommand()
```

[seq p**??**]
[getalist p**??**]
[exit p**??**]
[sayMSG p40]
[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 "     " nil))
    (do ((G166397 dependents (cdr G166397)) (y nil))
        ((or (atom G166397) (progn (setq y (car G166397)) nil)) nil)
      (seq (exit (setq msg (cons " " (cons y msg))))))
    (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil)))))
   nil)
  (exit
   (if (setq dependees (getalist |$dependeeAlist| x))
    (seq
     (if (null (cdr dependees))
      (exit
       (|sayMSG| (cons "   This depends on the following function "
                  (cons "or rule:" (|bright| (car dependees)))))))
     (|sayMSG| "   This depends on the following functions or rules:")
     (setq msg (cons "     " nil))
     (do ((G166406 dependees (cdr G166406)) (y nil))
         ((or (atom G166406) (progn (setq y (car G166406)) nil)) nil)
       (seq (exit (setq msg (cons " " (cons y msg))))))
     (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil)))))
    nil))))
```

———-

26.2.26 defun displayValue

[sayMSG p40]
[fixObjectForPrinting p707]
[pname p1106]
[objValUnwrap p462]
[objMode p462]
[displayRule p??]
[concat p1107]
[prefix2String p??]
[objMode p462]
[getdatabase p1070]
[concat p1107]
[form2String p??]
[mathprint p??]
[outputFormat p??]
[objMode p462]
[$op p??]
[$EmptyMode p629]

— **defun displayValue** —

```
(defun |displayValue| (|$op| u omitVariableNameIfTrue)
```

```
(declare (special |$op|))
(let (expr op rhs label labmode)
(declare (special |$EmptyMode|))
 (if (null u)
  (|sayMSG|
   (list '|   Value of | (|fixObjectForPrinting| (pname |$op|)) ":  (none)"))
  (progn
   (setq expr (|objValUnwrap| u))
   (if (or (and (consp expr) (progn (setq op (qcar expr)) t) (eq op 'map))
           (equal (|objMode| u) |$EmptyMode|))
    (|displayRule| |$op| expr)
    (progn
      (cond
       (omitVariableNameIfTrue
         (setq rhs "):  ")
         (setq label "Value (has type "))
       (t
        (setq rhs ":  ")
        (setq label (concat "Value of " (pname |$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))))))
```

———-

26.2.27 defun displayType

[sayMSG p40]
[fixObjectForPrinting p707]
[pname p1106]
[prefix2String p??]
[objMode p462]
[concat p1107]
[$op p??]

— **defun displayType** —

```
(defun |displayType| (|$op| u omitVariableNameIfTrue)
 (declare (special |$op|) (ignore omitVariableNameIfTrue))
 (let (type)
  (if (null u)
   (|sayMSG|
    (list  "   Type of value of " (|fixObjectForPrinting| (pname |$op|))
           ":  (none)"))
```

```
(progn
 (setq type (|prefix2String| (|objMode| u)))
 (when (atom type) (setq type (list type)))
 (|sayMSG|
  (|concat|
   (cons "   Type of value of "
    (cons (|fixObjectForPrinting| (pname |$op|))
     (cons ": " type)))))
 nil))))
```

———-

26.2.28 defun getAndSay

[getI p??]
[sayMSG p40]

— **defun getAndSay** —

```
(defun |getAndSay| (v prop)
 (let (val)
  (if (setq val (|getI| v prop))
   (|sayMSG| (cons '|    | (cons val (cons '|%l| nil))))
   (|sayMSG| (cons '|    none| (cons '|%l| nil))))))
```

———-

26.2.29 defun displayProperties

[getInterpMacroNames p??]
[getParserMacroNames p705]
[remdup p??]
[qcdr p??]
[qcar p??]
[msort p??]
[getWorkspaceNames p707]
[sayKeyedMsg p39]
[interpFunctionDepAlists p716]
[isInternalMapName p??]
[getIProplist p??]
[getAndSay p712]
[displayValue p710]
[getI p??]
[displayCondition p715]
[displayMode p717]
[displayType p711]
[fixObjectForPrinting p707]
[sayMSG p40]
[bright p??]
[prefix2String p??]

[member p1108]
[displayParserMacro p715]
[isInterpMacro p??]
[displayMacro p706]
[displayProperties,sayFunctionDeps p708]
[displayModemap p716]
[exit p??]
[seq p??]
[terminateSystemCommand p704]
[$dependentAlist p??]
[$dependeeAlist p??]
[$frameMessages p901]
[$interpreterFrameName p34]

— **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 |$frameMessages|
    (|sayKeyedMsg| "The name of the current frame is %1 ."
     (cons |$interpreterFrameName| nil)))
   (cond
    ((null vl)
     (if (null |$frameMessages|)
      (|sayKeyedMsg| "The workspace is empty." nil))
     (|sayKeyedMsg| "The current frame, %1 , is empty."
      (cons |$interpreterFrameName| nil)))
    (t
     (|interpFunctionDepAlists|)
     (do ((G166440 vl (cdr G166440)) (v nil))
         ((or (atom G166440) (progn (setq v (car G166440)) nil)) nil)
       (seq (exit
        (cond
         ((|isInternalMapName| v) '|iterate|)
         (t
          (setq pl (|getIProplist| v))
          (cond
           ((eq option '|flags|)
```

```
 (|getAndSay| v '|flags|))
((eq option '|value|)
 (|displayValue| v (|getI| v '|value|) nil))
((eq option '|condition|)
 (|displayCondition| v (|getI| v '|condition|) nil))
((eq option '|mode|)
 (|displayMode| v (|getI| v '|mode|) nil))
((eq option '|type|)
 (|displayType| v (|getI| v '|value|) nil))
((eq option '|properties|)
 (cond
  ((eq v '|--flags--|)
   nil)
  ((and (consp pl)
        (progn
         (setq tmp2 (qcar pl))
         (and (consp tmp2) (eq (qcar tmp2) '|cacheInfo|))))
   nil)
  (t
   (setq vone (|fixObjectForPrinting| v))
   (|sayMSG|
    (cons "Properties of"
     (append (|bright| (|prefix2String| vone)) (cons ":" nil))))
   (cond
    ((null pl)
     (cond
      ((|member| v pmacs)
       (|sayMSG| "   This is a user-defined macro.")
       (|displayParserMacro| v))
      ((|isInterpMacro| v)
       (|sayMSG| "   This is a system-defined macro.")
       (|displayMacro| v))
      (t
       (|sayMSG| "   none"))))
    (t
     (setq propsSeen nil)
     (do ((G166451 pl (cdr G166451)) (G166425 nil))
         ((or (atom G166451)
              (progn (setq G166425 (car G166451)) nil)
              (progn
               (progn
                (setq prop (car G166425))
                (setq val (cdr G166425))
                G166425)
               nil))
          nil)
       (seq (exit
        (cond
         ((and (null (member prop propsSeen)) val)
          (cond
           ((|member| prop
             '(|alias| |generatedCode| IS-GENSYM
               |mapBody| |localVars|))
            nil)
```

```
                    ((eq prop '|condition|)
                     (|displayCondition| prop val t))
                    ((eq prop '|recursive|)
                     (|sayMSG| "   This is recursive."))
                    ((eq prop '|isInterpreterFunction|)
                     (|sayMSG| "   This is an interpreter function.")
                     (|displayProperties,sayFunctionDeps| v))
                    ((eq prop '|isInterpreterRule|)
                     (|sayMSG| "   This is an interpreter rule.")
                     (|displayProperties,sayFunctionDeps| v))
                    ((eq prop '|localModemap|)
                     (|displayModemap| v val t))
                    ((eq prop '|mode|)
                     (|displayMode| prop val t))
                    (t
                     (when (eq prop '|value|)
                       (exit
                        (when val
                          (exit (|displayValue| v val t)))))
                      (|sayMSG| (list "   " prop ":  " val))
                      (setq propsSeen (cons prop propsSeen))))))))))))))
          (t
           (|sayKeyedMsg| "There is nothing to display for option %1 ."
            (cons option nil)))))))))
    (|terminateSystemCommand|)))))
```

———-

26.2.30 defun displayParserMacro

[pfPrintSrcLines p??]
[$pfMacros p352]

— **defun displayParserMacro** —

```
(defun |displayParserMacro| (m)
 (let ((m (assq m |$pfMacros|)))
 (declare (special |$pfMacros|))
  (when m (|pfPrintSrcLines| (caddr m)))))
```

———-

26.2.31 defun displayCondition

[bright p??]
[sayBrightly p??]
[concat p1107]
[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)))))
```

———-

26.2.32 defun interpFunctionDepAlists

[putalist p??]
[getalist p??]
[getFlag p??]
[$e p285]
[$dependeeAlist p??]
[$dependentAlist p??]
[$InteractiveFrame p34]

— **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))))
    (setq |$dependeeAlist|
     (putalist |$dependeeAlist| dependent
      (cons dependee (getalist |$dependeeAlist| dependent))))))
   (|getFlag| '|$dependencies|))))
```

———-

26.2.33 defun displayModemap

[bright p??]
[sayBrightly p??]
[concat p1107]
[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|)
     (setq varPart (unless giveVariableIfNil (cons " of" (|bright| v))))
     (setq prefix
      (cons '|   Compiled function type| (append varPart (cons '|: | nil))))
     (|sayBrightly| (|concat| prefix (|formatSignature| signature)))))))
  (mapcar #'(lambda (x) (g v x giveVariableIfNil)) val)))
```

———-

26.2.34 defun displayMode

[bright p??]
[fixObjectForPrinting p707]
[sayBrightly p??]
[concat p1107]
[prefix2String p??]

— **defun displayMode** —

```
(defun |displayMode| (v mode giveVariableIfNil)
 (let (varPart)
  (when mode
   (unless giveVariableIfNil
    (setq varPart (cons '| of| (|bright| (|fixObjectForPrinting| v)))))
   (|sayBrightly|
    (|concat| '|   Declared type or mode| varPart '|:   |
     (|prefix2String| mode))))))
```

———-

26.2.35 defun Split into tokens delimted by spaces

[stripSpaces p721]

— **defun dumbTokenize** —

```
(defun |dumbTokenize| (str)
 (let (inString token (tokenStart 0) previousSpace tokenList)
  (dotimes (i (1- (|#| str)))
   (cond
    ((char= (elt str i) #\") ; don't split strings
     (setq inString (null inString))
     (setq previousSpace nil))
    ((and (char= (elt str i) #\space) (null inString))
     (unless previousSpace
```

```
      (setq token (|stripSpaces| (subseq str tokenStart i)))
      (setq tokenList (cons token tokenList))
      (setq tokenStart (1+ i))
      (setq previousSpace t)))
   (t
    (setq previousSpace nil))))
 (setq tokenList (cons (|stripSpaces| (subseq str tokenStart)) tokenList))
 (nreverse tokenList)))
```

———-

26.2.36 defun Convert string tokens to their proper type

[isIntegerString p718]

— **defun tokTran** —

```
(defun |tokTran| (tok)
 (let (tmp)
  (if (stringp tok)
   (cond
    ((eql (|#| tok) 0) nil)
    ((setq tmp (|isIntegerString| tok)) tmp)
    ((char= (elt tok 0) #\" ) (subseq tok 1 (1- (|#| tok))))
    (t (intern tok)))
   tok)))
```

———-

26.2.37 defun Is the argument string an integer?

— **defun isIntegerString 0** —

```
(defun |isIntegerString| (tok)
 (multiple-value-bind (int len) (parse-integer tok :junk-allowed t)
  (when (and int (= len (length tok))) int)))
```

———-

26.2.38 defun Handle parsed system commands

[dumbTokenize p717]
[parseSystemCmd p719]
[tokTran p718]
[systemCommand p700]

— **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))))
```

———-

26.2.39 defun Parse a system command

[tokTran p718]
[stripSpaces p721]
[parseFromString p307]
[dumbTokenize p717]

— **defun parseSystemCmd** —

```
(defun |parseSystemCmd| (opt)
 (let (spaceIndex)
  (if (setq spaceIndex (search " " opt))
   (list
    (|tokTran| (|stripSpaces| (subseq opt 0 spaceIndex)))
    (|parseFromString| (|stripSpaces| (subseq opt spaceIndex))))
   (mapcar #'|tokTran| (|dumbTokenize| opt)))))
```

———-

26.2.40 defun Get first word in a string

[subseq p??]
[stringSpaces p??]

— **defun getFirstWord** —

```
(defun |getFirstWord| (string)
 (let (spaceIndex)
  (setq spaceIndex (search " " string))
  (if spaceIndex
   (|stripSpaces| (subseq string 0 spaceIndex))
   string)))
```

———-

26.2.41 defun Unabbreviate keywords in commands

[selectOptionLC p728]
[selectOption p728]
[commandsForUserLevel p701]

[$systemCommands p696]
[$currentLine p??]
[$syscommands p697]
[line p1027]

— defun unAbbreviateKeyword —

```
(defun |unAbbreviateKeyword| (x)
 (let (xp)
 (declare (special |$systemCommands| |$currentLine| $syscommands line))
  (setq xp (|selectOptionLC| x $syscommands '|commandErrorIfAmbiguous|))
  (cond
   ((null xp)
    (setq xp '|system|)
    (setq line (concat ")system " (substring line 1 (1- (|#| line)))))
    (setq |$currentLine| line)))
  (|selectOption| xp (|commandsForUserLevel| |$systemCommands|)
     '|commandUserLevelError|)))
```

———-

26.2.42 defun The command is ambiguous error

[commandAmbiguityError p705]
[$oldline p702]
[line p1027]

— defun commandErrorIfAmbiguous —

```
(defun |commandErrorIfAmbiguous| (x u)
 (declare (special $oldline line))
 (when u
   (setq $oldline line)
   (|commandAmbiguityError| '|command| x u)))
```

———-

[stripSpaces p721]
[nplisp p722]
[stripLisp p721]
[sayKeyedMsg p39]
[npboot p722]
[npsystem p722]
[npsynonym p722]
[member p1108]
[concat p1107]

— defun handleNoParseCommands —

```
(defun |handleNoParseCommands| (unab string)
 (let (spaceindex funname)
  (setq string (|stripSpaces| string))
```

```
  (setq spaceindex (search " " string))
  (cond
   ((eq unab '|lisp|)
    (if spaceindex
     (|nplisp| (|stripLisp| string))
     (|sayKeyedMsg| "Your argument list is not valid." nil)))
   ((eq unab '|boot|)
    (if spaceindex
     (|npboot| (subseq string (1+ spaceindex)))
     (|sayKeyedMsg| "Your argument list is not valid." nil)))
   ((eq unab '|system|)
    (if spaceindex
      (|npsystem| unab string)
      (|sayKeyedMsg| "Your argument list is not valid." nil)))
   ((eq unab '|synonym|)
    (if spaceindex
     (|npsynonym| unab (subseq string (1+ spaceindex)))
     (|npsynonym| unab "")))
   ((null spaceindex)
    (funcall unab))
   ((|member| unab '(|quit| |fin| |pquit| |credits| |copyright| |trademark|))
    (|sayKeyedMsg| "Your argument list is not valid." nil))
   (t
    (setq funname (intern (concat "np" (string unab))))
    (funcall funname (subseq string (1+ spaceindex))))))
```

———-

26.2.43 defun Remove the spaces surrounding a string

TPDHERE: This should probably be a macro or eliminated

— defun stripSpaces 0 —

```
(defun |stripSpaces| (str)
 (string-trim '(#\space) str))
```

———-

26.2.44 defun Remove the lisp command prefix

— defun stripLisp 0 —

```
(defun |stripLisp| (str)
 (if (string= (subseq str 0 4) "lisp")
  (subseq str 4)
  str))
```

———-

26.2.45 defun Handle the)lisp command

[$ans p??]

— **defun nplisp 0** —

```
(defun |nplisp| (str)
 (declare (special |$ans|))
  (setq |$ans| (eval (read-from-string str)))
  (format t "~&Value = ~S~%" |$ans|))
```

———-

26.2.46 defun The)boot command is no longer supported

TPDHERE: Remove all boot references from top level

— **defun npboot 0** —

```
(defun |npboot| (str)
 (declare (ignore str))
 (format t "The )boot command is no longer supported~%"))
```

———-

26.2.47 defun Handle the)system command

Note that unAbbreviateKeyword returns the word "system" for unknown words so we have to search for this case. This complication may never arrive in practice.

[sayKeyedMsg p39]

— **defun npsystem** —

```
(defun |npsystem| (unab str)
 (let (spaceIndex sysPart)
  (setq spaceIndex (search " " str))
  (cond
   ((null spaceIndex) (|sayKeyedMsg| "Unknown system command: %1" (list str)))
   (t
    (setq sysPart (subseq str 0 spaceIndex))
    (if (search sysPart (string unab))
     (obey (subseq str (1+ spaceIndex)))
     (|sayKeyedMsg| "Unknown system command: %1" (list sysPart)))))))
```

———-

26.2.48 defun Handle the)synonym command

[npProcessSynonym p723]

— **defun npsynonym** —

```
(defun |npsynonym| (unab str)
 (declare (ignore unab))
 (|npProcessSynonym| str))
```

———-

26.2.49 defun Handle the synonym system command

[printSynonyms p723]
[processSynonymLine p986]
[putalist p??]
[terminateSystemCommand p704]
[$CommandSynonymAlist p727]

— **defun npProcessSynonym** —

```
(defun |npProcessSynonym| (str)
 (let (pair)
 (declare (special |$CommandSynonymAlist|))
  (if (= (length str) 0)
   (|printSynonyms| nil)
   (progn
    (setq pair (|processSynonymLine| str))
    (if |$CommandSynonymAlist|
     (putalist |$CommandSynonymAlist| (car pair) (cdr pair)))
     (setq |$CommandSynonymAlist| (cons pair nil))))
  (|terminateSystemCommand|)))
```

———-

26.2.50 defun printSynonyms

[specialChar p1043]
[filterListOfStringsWithFn p1011]
[synonymsForUserLevel p984]
[printLabelledList p724]
[$CommandSynonymAlist p727]
[$linelength p936]

— **defun printSynonyms** —

```
(defun |printSynonyms| (patterns)
 (let (ls t1)
  (declare (special |$CommandSynonymAlist| $linelength))
  (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " System Command Synonyms ")
  (setq ls
    (|filterListOfStringsWithFn| patterns
     (do ((t2 (|synonymsForUserLevel| |$CommandSynonymAlist|) (cdr t2)))
         ((atom t2) (nreverse0 t1))
        (push (cons (princ-to-string (caar t2)) (cdar t2)) t1))
     #'car))
```

```
  (|printLabelledList| ls "user" "synonyms" ")" patterns)))
```

———-

26.2.51 defun Print a list of each matching synonym

The prefix goes before each element on each side of the list, eg, ")"

[sayMessage p**??**]
[blankList p**??**]
[substring p293]
[entryWidth p**??**]
[sayBrightly p**??**]
[concat p1107]
[fillerSpaces p279]

— **defun printLabelledList** —

```
(defun |printLabelledList| (ls label1 label2 prefix patterns)
 (let (comm syn wid)
  (if (null ls)
   (if (null patterns)
    (|sayMessage| (list "   No " label1 "-defined " label2 " in effect."))
    (|sayMessage|
     `("   No " ,label1 "-defined " ,label2 " satisfying patterns:"
       |%l| "     " ,@(append (|blankList| patterns) (list nil)))))
   (progn
    (when patterns
      (|sayMessage|
       `(,label1 "-defined " ,label2 " satisfying patterns:" |%l| "   "
          ,@(append (|blankList| patterns) (list nil)))))
     (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 (|#| syn)) 1))
      (|sayBrightly|
       (|concat| prefix syn (|fillerSpaces| wid ".")
         " " prefix comm)))
     (|sayBrightly| ""))))))
```

———-

26.2.52 defvar $tokenCommands

This is a list of the commands that expect the interpreter to parse their arguments. Thus the history command expects that Axiom will have tokenized and validated the input before calling the history function.

— **initvars** —

```
(defvar |$tokenCommands| nil)
```

————-

— **postvars** —

```
(eval-when (eval load)
 (setq |$tokenCommands|
 '( |abbreviations|
    |cd|
    |clear|
    |close|
    |compiler|
    |depends|
    |display|
    |describe|
    |edit|
    |frame|
    |help|
    |history|
    |input|
    |library|
    |load|
    |ltrace|
    |read|
    |regress|
    |savesystem|
    |set|
    |spool|
    |tangle|
    |undo|
    |what|
    |with|
    |workfiles|
   )))
```

————-

26.2.53 defvar $InitialCommandSynonymAlist

Axiom can create "synonyms" for commands. We create an initial table of synonyms which are in common use.

— **initvars** —

```
(defvar |$InitialCommandSynonymAlist| nil)
```

————-

26.2.54 defun Print the current version information

[*yearweek* p??]
[*build-version* p??]

— **defun axiomVersion 0** —

```
(defun axiomVersion ()
 (declare (special *build-version* *yearweek*))
  (concatenate 'string "Axiom " *build-version* " built on " *yearweek*))
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq |$InitialCommandSynonymAlist|
   '(
       (|?|           . "what commands")
       (|ap|          . "what things")
       (|apr|         . "what things")
       (|apropos|     . "what things")
       (|cache|       . "set functions cache")
       (|cl|          . "clear")
       (|cms|         . "system")
       (|co|          . "compiler")
       (|d|           . "display")
       (|dep|         . "display dependents")
       (|dependents|  . "display dependents")
       (|e|           . "edit")
       (|expose|      . "set expose add constructor")
       (|fns|         . "exec spadfn")
       (|fortran|     . "set output fortran")
       (|h|           . "help")
       (|hd|          . "system hypertex &")
       (|kclam|       . "boot clearClams ( )")
       (|killcaches|  . "boot clearConstructorAndLisplibCaches ( )")
       (|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")
       (|version|     . "lisp (axiomVersion)")
       (|w|           . "what")
       (|wc|          . "what categories")
       (|wd|          . "what domains")
       (|who|         . "lisp (pprint credits)")
```

```
       (|wp|          . "what packages")
       (|ws|          . "what synonyms")
)))
```

———-

26.2.55 defvar $CommandSynonymAlist

The actual list of synonyms is initialized to be the same as the above initial list of synonyms. The user synonyms that are added during a session are pushed onto this list for later lookup.

— **initvars** —

```
(defvar |$CommandSynonymAlist| nil)
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq |$CommandSynonymAlist| (copy-alist |$InitialCommandSynonymAlist|)))
```

———-

26.2.56 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 `)read` of input files.

[ncloopPrefix? p728]
[ncloopInclude1 p826]
[$systemCommandFunction p**??**]
[$systemCommandFunction p**??**]

— **defun ncloopCommand** —

```
(defun |ncloopCommand| (line n)
 (let (a)
 (declare (special |$systemCommandFunction|))
  (if (setq a (|ncloopPrefix?| ")include" line))
   (|ncloopInclude1| a n)
   (progn
    (funcall |$systemCommandFunction| line)
    n))))
```

———-

26.2.57 defun ncloopPrefix?

If we find the prefix string in the whole string starting at position zero we return the remainder of the string without the leading prefix.

— **defun ncloopPrefix? 0** —

```
(defun |ncloopPrefix?| (prefix whole)
 (when (eql (search prefix whole) 0)
  (subseq whole (length prefix))))
```

———-

26.2.58 defun selectOptionLC

[selectOption p728]
[downcase p1140]
[object2Identifier p??]

— **defun selectOptionLC** —

```
(defun |selectOptionLC| (x l errorFunction)
 (|selectOption| (downcase (|object2Identifier| x)) l errorFunction))
```

———-

26.2.59 defun selectOption

[member p1108]
[identp p1107]
[stringPrefix? p1254]
[pname p1106]
[qcdr p??]
[qcar p??]

— **defun selectOption** —

```
(defun |selectOption| (x l errorfunction)
 (let (u y)
  (cond
   ((|member| x l) x)
   ((null (identp x))
    (cond
     (errorfunction (funcall errorfunction x u))
     (t nil)))
   (t
    (setq u
     (let (t0)
       (do ((t1 l (cdr t1)) (y nil))
           ((or (atom t1) (progn (setq y (car t1)) nil)) (nreverse0 t0))
          (if (|stringPrefix?| (pname x) (pname y))
             (setq t0 (cons y t0))))))
    (cond
```

```
((and (consp u) (eq (qcdr u) nil) (progn (setq y (qcar u)) t)) y)
(errorfunction (funcall errorfunction x u))
(t nil))))))
```

———-

26.3)abbreviations Command

26.3.1 abbreviations man page

— abbreviations.help —

```
==========================================================================
A.2.  )abbreviation
==========================================================================

User Level Required:  compiler

Command Syntax:

  -  )abbreviation query  [nameOrAbbrev]
  -  )abbreviation category  abbrev  fullname []quiet]
  -  )abbreviation domain  abbrev  fullname   []quiet]
  -  )abbreviation package  abbrev  fullname  []quiet]
  -  )abbreviation remove  nameOrAbbrev

Command Description:

This command is used to query, set and remove abbreviations for category,
domain and package constructors. Every constructor must have a unique
abbreviation. This abbreviation is part of the name of the subdirectory under
which the components of the compiled constructor are stored. Furthermore, by
issuing this command you let the system know what file to load automatically
if you use a new constructor. Abbreviations must start with a letter and then
be followed by up to seven letters or digits. Any letters appearing in the
abbreviation must be in uppercase.

When used with the query argument, this command may be used to list the name
associated with a particular abbreviation or the abbreviation for a
constructor. If no abbreviation or name is given, the names and corresponding
abbreviations for all constructors are listed.

The following shows the abbreviation for the constructor List:

)abbreviation query List

The following shows the constructor name corresponding to the abbreviation
NNI:

)abbreviation query NNI

The following lists all constructor names and their abbreviations.

)abbreviation query

To add an abbreviation for a constructor, use this command with category,
domain or package. The following add abbreviations to the system for a
category, domain and package, respectively:
```

```
)abbreviation domain   SET Set
)abbreviation category COMPCAT  ComplexCategory
)abbreviation package  LIST2MAP ListToMap

If the )quiet option is used, no output is displayed from this command. You
would normally only define an abbreviation in a library source file. If this
command is issued for a constructor that has already been loaded, the
constructor will be reloaded next time it is referenced. In particular, you
can use this command to force the automatic reloading of constructors.

To remove an abbreviation, the remove argument is used. This is usually only
used to correct a previous command that set an abbreviation for a constructor
name. If, in fact, the abbreviation does exist, you are prompted for
confirmation of the removal request. Either of the following commands will
remove the abbreviation VECTOR2 and the constructor name VectorFunctions2
from the system:

)abbreviation remove VECTOR2
)abbreviation remove VectorFunctions2

Also See:
o )compile
```

———-

26.3.2 defun abbreviations

[abbreviationsSpad2Cmd p731]

— **defun abbreviations** —

```
(defun |abbreviations| (l)
 (|abbreviationsSpad2Cmd| l))
```

———-

26.3.3 defun abbreviationsSpad2Cmd

[listConstructorAbbreviations p733]
[abbreviation? p**??**]
[abbQuery p770]
[deldatabase p1069]
[size p1106]
[sayKeyedMsg p39]
[mkUserConstructorAbbreviation p**??**]
[setdatabase p1069]
[seq p**??**]
[exit p**??**]
[opOf p**??**]
[helpSpad2Cmd p782]

[selectOptionLC p728]
[qcar p??]
[qcdr p??]
[$options p63]

— **defun abbreviationsSpad2Cmd** —

```
(defun |abbreviationsSpad2Cmd| (arg)
 (let (abopts quiet opt key type constructor t2 a b al)
  (declare (special |$options|))
  (if (null arg)
   (|helpSpad2Cmd| '(|abbreviations|))
   (progn
    (setq abopts '(|query| |domain| |category| |package| |remove|))
    (setq quiet nil)
    (do ((t0 |$options| (cdr t0)) (t1 nil))
        ((or (atom t0)
             (progn (setq t1 (car t0)) nil)
             (progn (progn (setq opt (car t1)) t1) nil))
         nil)
      (setq opt (|selectOptionLC| opt '(|quiet|) '|optionError|))
      (when (eq opt '|quiet|) (setq quiet t)))
    (when
     (and (consp arg)
          (progn
           (setq opt (qcar arg))
           (setq al (qcdr arg))
           t))
      (setq key (|opOf| (car al)))
      (setq type (|selectOptionLC| opt abopts '|optionError|))
      (cond
       ((eq type '|query|)
        (cond
         ((null al) (|listConstructorAbbreviations|))
         ((setq constructor (|abbreviation?| key))
           (|abbQuery| constructor))
         (t (|abbQuery| key))))
       ((eq type '|remove|)
        (deldatabase key 'abbreviation))
       ((oddp (size al))
        (|sayKeyedMsg|
         (format nil
          "%1 must be followed by an alternating list of abbreviation(s) ~
           and name(s). Issue )abbrev ? for more information.")
         (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| "%1 abbreviates % %2 %3 %"
           (list a type (|opOf| b))))))))))
```

———-

26.3.4 defun listConstructorAbbreviations

[upcase p1140]
[queryUserKeyedMsg p??]
[string2id-n p??]
[whatSpad2Cmd p1008]
[sayKeyedMsg p39]

— **defun listConstructorAbbreviations** —

```
(defun |listConstructorAbbreviations| ()
 (let (x)
  (setq x
   (upcase
     (|queryUserKeyedMsg|
      (format nil
       "You have requested that all abbreviations be displayed. As there are ~
        several hundred abbreviations, please confirm your request by ~
        typing y or yes and then pressing Enter :")
       nil)))
  (if (member (string2id-n x 1) '(Y YES))
   (progn
    (|whatSpad2Cmd| '(|categories|))
    (|whatSpad2Cmd| '(|domains|))
    (|whatSpad2Cmd| '(|packages|)))
   (|sayKeyedMsg|
    (format nil
     "Since you did not respond with y or yes the list of abbreviations ~
      will not be displayed.")
     nil))))
```

———-

26.4)boot Command

26.4.1 boot man page

— **boot.help** —

```
====================================================================
A.3.  )boot
====================================================================

User Level Required:  development

Command Syntax:

  -  )boot bootExpression

Command Description:

This command is used by AXIOM system developers to execute expressions
written in the BOOT language. For example,

)boot times3(x) == 3*x

creates and compiles the Lisp function ‘‘times3’’ obtained by translating the
BOOT code.

Also See:
o )fin
o )lisp
o )set
o )system
```

[1]

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

[1] “fin” (26.20.2 p 779) “lisp” (26.27 p 831) “set” (26.51.1 p 962) “system” (26.56 p 987)

26.5)browse Command

26.5.1 browse 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.
```

———-

26.6 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

26.7 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:

```
<ul>
 <li>
  <input type="submit" id="p3" class="subbut"
    onclick="makeRequest('p3');"
    value="sin(x)" />
  <div id="ansp3"><div></div></div>
 </li>
</ul>
```

which says that this is an html "input" field of type "submit". The CSS display class is "subbut" which is of a different color than the surrounding text to make it obvious that you can click on this field. Clickable fields that have no response text are of class "noresult".

The javascript call to "makeRequest" gives the "id" of this input field, which must be unique in the page, as an argument. In this case, the argument is 'p3'. The "value" field holds the display text which will be passed back to Axiom as a command.

When the result arrives the "showanswer" function will select out the mathml field of the response, construct the "id" of the html div to hold the response by concatenating the string "ans" (answer) to the "id" of the request resulting, in this case, as "ansp3". The "showanswer" function will find this div and replace it with a div containing the mathml result.

The "makeRequest" function is:

```
function makeRequest(arg) {
  http_request = new XMLHttpRequest();
  var command = commandline(arg);
  //alert(command);
  http_request.open('POST', '127.0.0.1:8085', true);
  http_request.onreadystatechange = handleResponse;
  http_request.setRequestHeader('Content-Type', 'text/plain');
  http_request.send("command="+command);
  return(false);
```

It contains a request to open a local server connection to Axiom, sets "handleResponse" as the function to call on reply, sets up the type of request, fills in the command field, and sends off the http request.

When a response is received, the "handleResponse" function checks for the correct reply state, strips out the important text, and calls "showanswer".

```
function handleResponse() {
 if (http_request.readyState == 4) {
  if (http_request.status == 200) {
   showanswer(http_request.responseText,'mathAns');
  } else
  {
    alert('There was a problem with the request.'+ http_request.statusText);
```

```
   }
  }
 }
```

See bookvol11.pamphlet for further details.

26.8 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.

26.9 The)browse command

In order to make the whole process cleaner the function ")browse" handles the details. This code creates the command-line function for)browse

The browse function does the internal equivalent of the following 3 command line statments:

```
)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV
```

which causes Axiom to start serving web pages on port 8085

For those unfamiliar with calling algebra from lisp there are a few points to mention.

The loadLib needs to be called to load the algebra code into the image. Normally this is automatic but we are not using the interpreter so we need to do this "by hand".

Each algebra file contains a "constructor function" which builds the domain, which is a vector, and then caches the vector so that every call to the contructor returns an EQ vector, that is, the same vector. In this case, we call the constructor |AxiomServer|

The axServer function was mangled internally to |AXSERV;axServer;IMV;2|. The multiServ function was mangled to |AXSERV;multiServ;SeV;3| Note well that if you change axserver.spad these names might change which will generate the error message along the lines of:

```
System error:
The function $\vert$AXSERV;axServer;IMV;2$\vert$ is undefined.
```

To fix this you need to look at int/algebra/AXSERV.nrlib/code.lsp and find the new mangled function name. A better solution would be to dynamically look up the surface names in the domain vector.

Each Axiom function expects the domain vector as the last argument. This is not obvious from the call as the interpreter supplies it. We must do that "by hand".

We don't call the multiServ function. We pass it as a parameter to the axServer function. When it does get called by the SPADCALL macro it needs to be a lisp pair whose car is the function and whose cdr is the domain vector. We construct that pair here as the second

argument to axServer. The third, hidden, argument to axServer is the domain vector which we supply "by hand".

The socket can be supplied on the command line but defaults to 8085. Axiom supplies the arguments as a list.

[set p962]
[loadLib p1093]
[AxiomServer p**??**]
[AXSERV;axServer;IMV;2 p**??**]

— **defun browse** —

```
(defun |browse| (socket)
 (let (axserv browser)
  (if socket
    (setq socket (car socket))
    (setq socket 8085))
  (|set| '(|mes| |auto| |off|))
  (|set| '(|out| |mathml| |on|))
  (|loadLib| '|AxiomServer|)
  (setq axserv (|AxiomServer|))
  (setq browser
   (|AXSERV;axServer;IMV;2| socket
    (cons #'|AXSERV;multiServ;SeV;3| axserv) axserv))))
```

————-

Now we have to bolt it into Axiom. This involves two lookups.

We create the lisp pair

```
(|browse| . |development|)
```

and cons it into the $systemCommands command table. This allows the command to be executed in development mode. This lookup decides if this command is allowed. It also has the side-effect of putting the command into the $SYSCOMMANDS variable which is used to determine if the token is a command.

26.10 The server support code

26.11)cd Command

26.11.1 cd man page

— cd.help —

```
====================================================================
A.4.  )cd
====================================================================

User Level Required:  interpreter

Command Syntax:

  -  )cd directory

Command Description:

This command sets the AXIOM working current directory. The current directory
is used for looking for input files (for )read), AXIOM library source files
(for )compile), saved history environment files (for )history )restore),
compiled AXIOM library files (for )library), and files to edit (for )edit).
It is also used for writing spool files (via )spool), writing history input
files (via )history )write) and history environment files (via )history
)save),and compiled AXIOM library files (via )compile).

If issued with no argument, this command sets the AXIOM current directory to
your home directory. If an argument is used, it must be a valid directory
name. Except for the ``)'' at the beginning of the command, this has the same
syntax as the operating system cd command.

Also See:
o )compile
o )edit
o )history
o )library
o )read
o )spool
```

———-

[2]

[2] "edit" (26.19.2 p 776) "history" (26.23.11 p 791) "library" (36.1.30 p 1073) "read" (26.31.2 p 838) "spool" (26.53 p 980)

26.12)clear Command

26.12.1 clear man page

— clear.help —

```
====================================================================
A.6.  )clear
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )clear all
  - )clear completely
  - )clear properties all
  - )clear properties  obj1 [obj2 ...]
  - )clear value      all
  - )clear value     obj1 [obj2 ...]
  - )clear mode       all
  - )clear mode      obj1 [obj2 ...]

Command Description:

This command is used to remove function and variable declarations,
definitions and values from the workspace. To empty the entire workspace and
reset the step counter to 1, issue

)clear all

To remove everything in the workspace but not reset the step counter, issue

)clear properties all

To remove everything about the object x, issue

)clear properties x

To remove everything about the objects x, y and f, issue

)clear properties x y f

The word properties may be abbreviated to the single letter ‘‘p’’.

)clear p all
)clear p x
)clear p x y f

All definitions of functions and values of variables may be removed by either

)clear value all
)clear v all
```

```
This retains whatever declarations the objects had. To remove definitions and
values for the specific objects x, y and f, issue

)clear value x y f
)clear v x y f

To remove the declarations of everything while leaving the definitions and
values, issue

)clear mode  all
)clear m all

To remove declarations for the specific objects x, y and f, issue

)clear mode x y f
)clear m x y f

The )display names and )display properties commands may be used to see what
is currently in the workspace.

The command

)clear completely

does everything that )clear all does, and also clears the internal system
function and constructor caches.

Also See:
o )display
o )history
o )undo
```

———-

[3]

26.12.2 defvar $clearOptions

— **initvars** —

```
(defvar |$clearOptions| '(|modes| |operations| |properties| |types| |values|))
```

———-

26.12.3 defun clear

[clearSpad2Cmd p742]

[3] “display” (26.18.3 p 768) “history” (26.23.11 p 791) “undo” (26.59 p 991)

— **defun clear** —

```
(defun |clear| (l)
 (|clearSpad2Cmd| l))
```

————-

26.12.4 defvar $clearExcept

— **initvars** —

```
(defvar |$clearExcept| nil)
```

————-

26.12.5 defun clearSpad2Cmd

TPDHERE: Note that this function also seems to parse out)except)completely and)scaches which don't seem to be documented. [selectOptionLC p728]
[sayKeyedMsg p39]
[clearCmdAll p745]
[clearCmdCompletely p744]
[clearCmdSortedCaches p743]
[clearCmdExcept p747]
[clearCmdParts p747]
[updateCurrentInterpreterFrame p27]
[$clearExcept p742]
[$options p63]
[$clearOptions p741]

— **defun clearSpad2Cmd** —

```
(defun |clearSpad2Cmd| (l)
  (let (|$clearExcept| opt optlist arg)
  (declare (special |$clearExcept| |$options| |$clearOptions|))
  (cond
   (|$options|
    (setq |$clearExcept|
     (prog (t0)
      (setq t0 t)
      (return
       (do ((t1 nil (null t0))
            (t2 |$options| (cdr t2))
            (t3 nil))
           ((or t1
                (atom t2)
                (progn (setq t3 (car t2)) nil)
                (progn (progn (setq opt (car t3)) t3) nil))
             t0)
        (setq t0
         (and t0
```

```
            (eq
             (|selectOptionLC| opt '(|except|) '|optionError|)
             '|except|))))))))))
  (cond
   ((null l)
     (setq optlist
      (prog (t4)
       (setq t4 nil)
       (return
        (do ((t5 |$clearOptions| (cdr t5)) (x nil))
            ((or (atom t5) (progn (setq x (car t5)) nil)) t4)
          (setq t4 (append t4 '(|%l| "       " ,x)))))))
     (|sayKeyedMsg|
      (format nil
       "Use )clear all to clear everything in the workspace.  Use )clear ~
        completely to clear everything in the workspace and internal ~
        tables.  Other )clear keyword arguments are %1 %l or abbreviations ~
        thereof.  Issue )clear ? for more information.")
       (list optlist)))
   (t
     (setq arg
      (|selectOptionLC| (car l) '(|all| |completely| |scaches|) nil))
     (cond
      ((eq arg '|all|)        (|clearCmdAll|))
      ((eq arg '|completely|) (|clearCmdCompletely|))
      ((eq arg '|scaches|)    (|clearCmdSortedCaches|))
      (|$clearExcept|         (|clearCmdExcept| l))
      (t
       (|clearCmdParts| l)
       (|updateCurrentInterpreterFrame|)))))))
```

———-

26.12.6 defun clearCmdSortedCaches

[compiledLookupCheck p744]
[spadcall p??]
[$lookupDefaults p??]
[$Void p634]
[$ConstructorCache p??]

— **defun clearCmdSortedCaches** —

```
(defun |clearCmdSortedCaches| ()
 (let (|$lookupDefaults| domain pair)
  (declare (special |$lookupDefaults| |$Void| |$ConstructorCache|))
  (do ((t0 (hget |$ConstructorCache| '|SortedCache|) (cdr t0))
       (t1 nil))
      ((or (atom t0)
           (progn
            (setq t1 (car t0))
            (setq domain (cddr t1))
            nil))
```

```
        nil)
    (setq pair (|compiledLookupCheck| '|clearCache| (list |$Void|) domain))
    (spadcall pair))))
```

———-

26.12.7 defun compiledLookupCheck

[compiledLookup p1097]
[keyedSystemError p??]
[formatSignature p??]

— **defun compiledLookupCheck** —

```
(defun |compiledLookupCheck| (op sig dollar)
 (let (fn)
  (setq fn (|compiledLookup| op sig dollar))
  (cond
    ((and (null fn) (eq op '^))
      (setq fn (|compiledLookup| '** sig dollar)))
    ((and (null fn) (eq op '**))
      (setq fn (|compiledLookup| '^ sig dollar)))
    (t nil))
  (cond
   ((null fn)
    (|keyedSystemError|
      "The function %1 with signature %2 is missing from domain %3"
      (list op (|formatSignature| sig) (elt dollar 0))))
   (t fn))))
```

———-

26.12.8 defvar $functionTable

— **initvars** —

```
(defvar |$functionTable| nil)
```

———-

26.12.9 defun clearCmdCompletely

[clearCmdAll p745]
[sayKeyedMsg p39]
[clearClams p??]
[clearConstructorCaches p??]
[reclaim p299]
[$localExposureData p147]
[$xdatabase p??]

[$CatOfCatDatabase p??]
[$DomOfCatDatabase p??]
[$JoinOfCatDatabase p??]
[$JoinOfDomDatabase p??]
[$attributeDb p??]
[$functionTable p744]
[$existingFiles p??]
[$localExposureDataDefault p148]

— **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 |$functionTable| nil)
  (|sayKeyedMsg| "All )browse facility databases have been cleared." nil)
  (|clearClams|)
  (|clearConstructorCaches|)
  (setq |$existingFiles| (make-hash-table :test #'equal))
  (|sayKeyedMsg|
    "Internally cached functions and constructors have been cleared." nil)
  (reclaim)
  (|sayKeyedMsg| ")clear completely is finished." nil))
```

———-

26.12.10 defun clearCmdAll

[clearCmdSortedCaches p743]
[untraceMapSubNames p92]
[resetInCoreHist p798]
[deleteFile p1104]
[histFileName p789]
[updateCurrentInterpreterFrame p27]
[clearMacroTable p746]
[sayKeyedMsg p39]
[$frameRecord p45]
[$previousBindings p45]
[$variableNumberAlist p??]
[$InteractiveFrame p34]
[$useInternalHistoryTable p788]

[$internalHistoryTable p42]
[$frameMessages p901]
[$interpreterFrameName p34]
[$currentLine p??]
[$traceNames p66]

— **defun clearCmdAll** —

```
(defun |clearCmdAll| ()
  (declare (special |$frameRecord| |$previousBindings| |$variableNumberAlist|
     |$InteractiveFrame| |$useInternalHistoryTable| |$internalHistoryTable|
     |$frameMessages| |$interpreterFrameName| |$currentLine| |$traceNames|))
  (|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|
     (format nil
       "All user variables and function definitions have been cleared in ~
        the current frame ( %1 ).")
      (list |$interpreterFrameName|))
    (|sayKeyedMsg|
      "All user variables and function definitions have been cleared." nil)))
```

———-

26.12.11 defun clearMacroTable

[$pfMacros p352]

— **defun clearMacroTable 0** —

```
(defun |clearMacroTable| ()
  (declare (special |$pfMacros|))
  (setq |$pfMacros| nil))
```

———-

26.12.12 defun clearCmdExcept

Clear all the options except the argument. [stringPrefix? p1254]
[object2String p??]
[clearCmdParts p747]
[$clearOptions p741]

— **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))))))
```

———-

26.12.13 defun clearCmdParts

[selectOptionLC p728]
[pname p1106]
[types p??]
[modes p??]
[values p??]
[boot-equal p??]
[assocleft p??]
[remdup p??]
[assoc p??]
[isMap p??]
[get p??]
[exit p??]
[untraceMapSubNames p92]
[seq p??]
[recordOldValue p801]
[recordNewValue p801]
[deleteAssoc p??]
[sayKeyedMsg p39]
[getParserMacroNames p705]
[getInterpMacroNames p??]
[clearDependencies p??]
[member p1108]
[clearParserMacro p705]
[sayMessage p??]
[fixObjectForPrinting p707]
[$e p285]
[$InteractiveFrame p34]
[$clearOptions p741]

— **defun clearCmdParts** —

```
(defun |clearCmdParts| (arg)
 (let (|$e| (opt (car arg)) option pmacs imacs (vl (cdr arg)) p1 lm prop p2)
 (declare (special |$e| |$InteractiveFrame| |$clearOptions|))
  (setq option (|selectOptionLC| opt |$clearOptions| '|optionError|))
  (setq option (intern (pname option)))
  (setq option
   (case option
    (|types| '|mode|)
    (|modes| '|mode|)
    (|values| '|value|)
    (t option)))
  (if (null vl)
   (|sayKeyedMsg|
    (format nil
     "After the property you wish to clear you must give one or more ~
      identifiers or specify all to clear that property from everything.")
     nil)
   (progn
    (setq pmacs (|getParserMacroNames|))
    (setq imacs (|getInterpMacroNames|))
    (cond
     ((boot-equal vl '(|all|))
      (setq vl (assocleft (caar |$InteractiveFrame|)))
      (setq vl (remdup (append vl pmacs)))))
    (setq |$e| |$InteractiveFrame|)
    (do ((t0 vl (cdr t0)) (x nil))
        ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
      (|clearDependencies| x t)
      (when (and (eq option '|properties|) (|member| x pmacs))
        (|clearParserMacro| x))
      (when (and (eq option '|properties|)
                 (|member| x imacs)
                 (null (|member| x pmacs)))
        (|sayMessage| (cons
         "   You cannot clear the definition of the system-defined macro "
          (cons (|fixObjectForPrinting| x)
                (cons (intern "." "BOOT") nil)))))
      (cond
       ((setq p1 (|assoc| x (caar |$InteractiveFrame|)))
        (cond
         ((eq option '|properties|)
          (cond
           ((|isMap| x)
            (seq
             (cond
              ((setq lm
                (|get| x '|localModemap| |$InteractiveFrame|))
               (cond
                ((consp lm)
                 (exit (|untraceMapSubNames| (cons (cadar lm) nil))))))
              (t nil)))))
          (dolist (p2 (cdr p1))
            (setq prop (car p2))
            (|recordOldValue| x prop (cdr p2))
```

```
          (|recordNewValue| x prop nil))
        (setf (caar |$InteractiveFrame|)
         (|deleteAssoc| x (caar |$InteractiveFrame|))))
       ((setq p2 (|assoc| option (cdr p1)))
        (|recordOldValue| x option (cdr p2))
        (|recordNewValue| x option nil)
        (rplacd p2 nil))))))
nil))))
```

———-

26.13)close Command

26.13.1 close man page

— close.help —

```
=======================================================================
A.5.  )close
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )close
  - )close )quietly

Command Description:

This command is used to close down interpreter client processes. Such
processes are started by HyperDoc to run AXIOM examples when you click on
their text. When you have finished examining or modifying the example and you
do not want the extra window around anymore, issue

)close

to the AXIOM prompt in the window.

If you try to close down the last remaining interpreter client process, AXIOM
will offer to close down the entire AXIOM session and return you to the
operating system by displaying something like

   This is the last AXIOM session. Do you want to kill AXIOM?

Type "y" (followed by the Return key) if this is what you had in mind. Type
"n" (followed by the Return key) to cancel the command.

You can use the )quietly option to force AXIOM to close down the interpreter
client process without closing down the entire AXIOM session.

Also See:
o )quit
o )pquit
```

————-

4

[4] "quit" (26.30.2 p 836) "pquit" (26.29.2 p 834)

26.13.2 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|))
```

———-

26.13.3 defun close

[throwKeyedMsg p??]
[sockSendInt p??]
[closeInterpreterFrame p33]
[selectOptionLC p728]
[upcase p1140]
[queryUserKeyedMsg p??]
[string2id-n p??]
[queryClients p751]
[$SpadServer p178]
[$SessionManager p??]
[$CloseClient p??]
[$currentFrameNum p302]
[$options p63]

— **defun close** —

```
(defun |close| (args)
 (declare (ignore args))
 (let (numClients opt fullopt quiet x)
 (declare (special |$SpadServer| |$SessionManager| |$CloseClient|
     |$currentFrameNum| |$options|))
  (if (null |$SpadServer|)
   (|throwKeyedMsg| "You cannot close this Axiom session." 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|
     "This is the last Axiom session.  Do you want to kill Axiom?"
     nil)))
  (when (member (string2id-n x 1) '(yes y)) (bye)))))))))
```

———-

26.14)compile Command

26.14.1 compile man page

— compile.help —

```
=======================================================================
A.7.  )compile
=======================================================================

User Level Required:  compiler

Command Syntax:

  -  )compile
  -  )compile fileName
  -  )compile fileName.spad
  -  )compile directory/fileName.spad
  -  )compile fileName )quiet
  -  )compile fileName )noquiet
  -  )compile fileName )break
  -  )compile fileName )nobreak
  -  )compile fileName )library
  -  )compile fileName )nolibrary
  -  )compile fileName )vartrace
  -  )compile fileName )constructor nameOrAbbrev

Command Description:

You use this command to invoke the AXIOM library compiler.  This
compiles files with file extension .spad with the AXIOM system
compiler. The command first looks in the standard system directories
for files with extension .spad.

Should you not want the )library command automatically invoked, call )compile
with the )nolibrary option. For example,

)compile mycode )nolibrary

By default, the )library system command exposes all domains and categories it
processes. This means that the AXIOM intepreter will consider those domains
and categories when it is trying to resolve a reference to a function.
Sometimes domains and categories should not be exposed. For example, a domain
may just be used privately by another domain and may not be meant for
top-level use. The )library command should still be used, though, so that the
code will be loaded on demand. In this case, you should use the )nolibrary
option on )compile and the )noexpose option in the )library command. For
example,

)compile mycode.spad )nolibrary
)library mycode )noexpose

Once you have established your own collection of compiled code, you may find
```

```
it handy to use the )dir option on the )library command. This causes )library
to process all compiled code in the specified directory. For example,

)library )dir /u/jones/as/quantum

You must give an explicit directory after )dir, even if you want all compiled
code in the current working directory processed.

)library )dir .

You can compile category, domain, and package constructors contained in files
with file extension .spad. You can compile individual constructors or every
constructor in a file.

The full filename is remembered between invocations of this command and )edit
commands. The sequence of commands

)compile matrix.spad
)edit
)compile

will call the compiler, edit, and then call the compiler again on the file
matrix.spad. If you do not specify a directory, the working current directory
(see description of command )cd ) is searched for the file. If the file is
not found, the standard system directories are searched.

If you do not give any options, all constructors within a file are compiled.
Each constructor should have an )abbreviation command in the file in which it
is defined. We suggest that you place the )abbreviation commands at the top
of the file in the order in which the constructors are defined. The list of
commands serves as a table of contents for the file.

The )library option causes directories containing the compiled code for each
constructor to be created in the working current directory. The name of such
a directory consists of the constructor abbreviation and the .NRLIB file
extension. For example, the directory containing the compiled code for the
MATRIX constructor is called MATRIX.NRLIB. The )nolibrary option says that
such files should not be created.

The )vartrace option causes the compiler to generate extra code for the
constructor to support conditional tracing of variable assignments. (see
description of command )trace ). Without this option, this code is suppressed
and one cannot use the )vars option for the trace command.

The )constructor option is used to specify a particular constructor to
compile. All other constructors in the file are ignored. The constructor name
or abbreviation follows )constructor. Thus either

)compile matrix.spad )constructor RectangularMatrix

or

)compile matrix.spad )constructor RMATRIX
```

```
compiles the RectangularMatrix constructor defined in matrix.spad.

The )break and )nobreak options determine what the compiler does
when it encounters an error. )break is the default and it indicates that
processing should stop at the first error. The value of the )set break
variable then controls what happens.

Also See:
o )abbreviation
o )edit
o )library
```

———-

[5]

26.14.2 defvar /editfile

— **initvars** —

```
(defvar /editfile nil)
```

———-

[5] "abbreviation" (**??** p **??**) "edit" (26.19.2 p 776) "library" (36.1.30 p 1073)

26.15)copyright Command

26.15.1 copyright man page

— copyright.help —

```
The term Axiom, in the field of computer algebra software,
along with AXIOM and associated images are common-law
trademarks. While the software license allows copies, the
trademarks may only be used when referring to this project.

Axiom is distributed under terms of the Modified BSD license.
Axiom was released under this license as of September 3, 2002.
Source code is freely available at:
http://savannah.nongnu.org/projects/axiom
Copyrights remain with the original copyright holders.
Use of this material is by permission and/or license.
Individual files contain reference to these applicable copyrights.
The copyright and license statements are collected here for reference.

Portions Copyright (c) 2003- The Axiom Team

The Axiom Team is the collective name for the people who have
contributed to this project. Where no other copyright statement
is noted in a file this copyright will apply.

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

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

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

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

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
```

```
      or redistribute your modifications without limitation;
  (c) you represent that you are legally entitled to grant these rights
      and that you are not providing Codemist with any code that violates
      any law or breaches any contract.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1995-1997 Eric Young (eay@mincom.oz.au)
All rights reserved.

This package is an SSL implementation written
by Eric Young (eay@mincom.oz.au).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as
the following conditions are aheared to.  The following conditions
apply to all code found in this distribution, be it the RC4, RSA,
lhash, DES, etc., code; not just the SSL code.  The SSL documentation
included with this distribution is covered by the same copyright terms
except that the holder is Tim Hudson (tjh@mincom.oz.au).

Copyright remains Eric Young's, and as such any Copyright notices in
the code are not to be removed.
If this package is used in a product, Eric Young should be given attribution
as the author of the parts of the library used.
This can be in the form of a textual message at program startup or
in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software
   must display the following acknowledgement:
   "This product includes cryptographic software written by
    Eric Young (eay@mincom.oz.au)"
   The word 'cryptographic' can be left out if the rouines from the library
   being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from
   the apps directory (application code) you must include an acknowledgement:
```

———-

26.15.2 defun copyright

[obey p??]
[concat p1107]
[getenviron p291]

— **defun copyright** —

```
(defun |copyright| ()
 (obey (concat "cat " (getenviron "AXIOM") "/doc/spadhelp/copyright.help")))
```

————-

26.15.3 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` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

26.16)credits Command

26.16.1 credits man page

26.16.2 defun credits

[credits p762]

— **defun credits 0** —

```
(defun |credits| ()
 (declare (special credits))
 (mapcar #'(lambda (x) (princ x) (terpri)) creditlist))
```

————-

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

26.17)describe Command

26.17.1 describe 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.
```

———-

26.17.2 defvar $describeOptions

The current value of $describeOptions is

— **initvars** —

```
(defvar $describeOptions '(|category| |domain| |package|))
```

———-

26.17.3 defun Print comment strings from algebra libraries

This trivial function satisfies the standard pattern of making a user command match the name of the function which implements the command. That command immediatly invokes a "Spad2Cmd" version. [describespad2cmd p**??**]

— **defun describe** —

```
(defun |describe| (l)
 (describeSpad2Cmd l))
```

———-

26.17.4 defun describeSpad2Cmd

The describe command prints cleaned-up comment strings from the algebra libraries. It can print strings associated with a category, domain, package, or by operation.

This implements command line options of the form:

```
)describe categoryName [internal]
)describe domainName   [internal]
)describe packageName  [internal]
```

The describeInternal function will either call the "dc" function to describe the internal representation of the argument or it will print a cleaned up version of the text for the "Description" keyword in the Category, Domain, or Package source code.

[selectOptionLC p728]
[flatten p766]
[cleanline p765]
[getdatabase p1070]
[sayMessage p**??**]
[$e p285]
[$EmptyEnvironment p**??**]
[$describeOptions p763]

— **defun describeSpad2Cmd** —

```
(defun describeSpad2Cmd (l)
 (labels (
  (fullname (arg)
   "Convert abbreviations to the full constructor name"
   (let ((abb (getdatabase arg 'abbreviation)))
    (if abb arg (getdatabase arg 'constructor))))
  (describeInternal (cdp internal?)
   (if internal?
    (progn
     (unless (eq (getdatabase cdp 'constructorkind) '|category|) (|dc| cdp))
     (showdatabase cdp))
    (mapcar #'(lambda (x) (if (stringp x) (cleanline x)))
     (flatten  (car (getdatabase (fullname cdp) 'documentation)))))))
 (let ((|$e| |$EmptyEnvironment|) (opt (second l)))
  (declare (special |$e| |$EmptyEnvironment| $describeOptions))
```

```
  (if (and (consp l) (not (eq opt '?)))
    (describeInternal (first l) (second l))
   (|sayMessage|
    (format nil "~&~a ~{~%    ~a~^ ~}~%    ~a"
     "  )describe keyword arguments are"
     $describeOptions
     "or abbreviations thereof"))))))
```

———-

26.17.5 defun cleanline

— **defun cleanline** —

```
(defun cleanline (line)
 (labels (
   (replaceInLine (thing other line)
    (do ((mark (search thing line) (search thing line)))
        ((null mark) line)
     (setq line
      (concatenate 'string (subseq line 0 mark) other
                           (subseq line (+ mark (length thing)))))))

   (removeFromLine (thing line) (replaceInLine thing "" line))

   (removeKeyword (str line)
    (do ((mark (search str line) (search str line)))
     ((null mark) line)
     (let (left point mid right)
      (setq left (subseq line 0 mark))
      (setq point (search "}" line :start2 mark))
      (setq mid (subseq line (+ mark (length str)) point))
      (setq right (subseq line (+ point 1)))
      (setq line (concatenate 'string left mid right)))))

   (addSpaces (str line)
    (do ((mark (search str line) (search str line)) (cnt))
     ((null mark) line)
     (let (left point mid right)
      (setq left (subseq line 0 mark))
      (setq point (search "}" line :start2 mark))
      (setq mid (subseq line (+ mark (length str)) point))
      (if (setq cnt (parse-integer mid :junk-allowed t))
        (setq mid (make-string cnt :initial-element #\ ))
        (setq mid ""))
      (setq right (subseq line (+ point 1)))
      (setq line (concatenate 'string left mid right)))))

   (splitAtNewline (line)
    (do ((mark (search "~%" line) (search "~%" line)) (lines))
        ((null mark)
          (push " " lines)
```

```
          (push line lines)
          (nreverse lines))
      (push (subseq line 0 mark) lines)
      (setq line (subseq line (+ mark 2)))))

   (wrapOneLine (line margin result)
    (if (null line)
      (nreverse result)
      (if (< (length line) margin)
        (wrapOneLine nil margin (append (list line) result))
        (let (oneline spill aspace)
         (setq aspace (position #\space (subseq line 0 margin) :from-end t))
         (setq oneline (string-trim '(#\space) (subseq line 0 aspace)))
         (setq spill (string-trim '(#\space) (subseq line aspace)))
         (wrapOneLine spill margin (append (list oneline) result))))))

   (reflowParagraph (line)
    (let (lst1)
     (setq lst1 (splitAtNewLine line))
     (dolist (x lst1)
      (mapcar #'(lambda(y) (format t "~a~%" y))
       (wrapOneLine x 70 nil))))))

 (setq line (removeFromLine "{}" line))
 (setq line (replaceInLine "\\blankline" "~%~%" line))
 (setq line (replaceInLine "\\br" "~%" line))
 (setq line (removeFromLine "\\" line))
 (dolist (str '("spad{" "spadtype{" "spadop{" "spadfun{" "spadatt{"
                "axiom{" "axiomType{" "spadignore{" "axiomFun{"
                "centerline{" "inputbitmap{" "axiomOp{" "spadgloss{"))
  (setq line (removeKeyword str line)))
 (setq line (replaceInLine "{e.g.}" "e.g." line))
 (dolist (str '("tab{" "indented{" ))
   (setq line (addSpaces str line)))
  (reflowParagraph line)))
```

———-

26.17.6 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)))
```

———-

26.18)display Command

26.18.1 display man page

— display.help —

```
======================================================================
A.8.  )display
======================================================================

User Level Required:  interpreter

Command Syntax:

  -  )display all
  -  )display properties
  -  )display properties all
  -  )display properties [obj1 [obj2 ...] ]
  -  )display value all
  -  )display value [obj1 [obj2 ...] ]
  -  )display mode all
  -  )display mode [obj1 [obj2 ...] ]
  -  )display names
  -  )display operations opName

Command Description:

This command is used to display the contents of the workspace and signatures
of functions with a given name. (A signature gives the argument and return
types of a function.)

The command

)display names

lists the names of all user-defined objects in the workspace. This is useful
if you do not wish to see everything about the objects and need only be
reminded of their names.

The commands

)display all
)display properties
)display properties all

all do the same thing: show the values and types and declared modes of all
variables in the workspace. If you have defined functions, their signatures
and definitions will also be displayed.

To show all information about a particular variable or user functions, for
example, something named d, issue

)display properties d
```

```
To just show the value (and the type) of d, issue

)display value d

To just show the declared mode of d, issue

)display mode d

All modemaps for a given operation may be displayed by using )display
operations. A modemap is a collection of information about a particular
reference to an operation. This includes the types of the arguments and the
return value, the location of the implementation and any conditions on the
types. The modemap may contain patterns. The following displays the modemaps
for the operation FromcomplexComplexCategory:

)d op complex

Also See:
o )clear
o )history
o )set
o )show
o )what
```

———-

[6]

26.18.2 defvar $displayOptions

The current value of $displayOptions is

— **initvars** —

```
(defvar |$displayOptions|
 '(|abbreviations| |all| |macros| |modes| |names| |operations|
   |properties| |types| |values|))
```

———-

26.18.3 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))
```

[6] "clear" (26.12.3 p 741) "history" (26.23.11 p 791) "set" (26.51.1 p 962) "show" (26.52.2 p 967) "what" (26.61.3 p 1007)

———-

26.18.4 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 $displayOptions list to construct a list of strings for the sayMessage function and tell the user what options are available. [abbQuery p770]
[opOf p**??**]
[listConstructorAbbreviations p733]
[displayOperations p770]
[displayMacros p771]
[displayWorkspaceNames p706]
[displayProperties p712]
[selectOptionLC p728]
[sayMessage p**??**]
[$e p285]
[$EmptyEnvironment p**??**]
[$displayOptions p768]

— **defun displaySpad2Cmd** —

```
(defun displaySpad2Cmd (l)
 (let ((|$e| |$EmptyEnvironment|) (opt (car l)) (vl (cdr l)) option)
  (declare (special |$e| |$EmptyEnvironment| |$displayOptions|))
  (if (and (consp l) (not (eq opt '?)))
   (progn
    (setq option (|selectOptionLC| opt |$displayOptions| '|optionError|))
    (cond
     ((eq option '|all|)
       (setq l (list '|properties|))
       (setq option '|properties|))
     ((or (eq option '|modes|) (eq option '|types|))
       (setq l (cons '|type| vl))
       (setq option '|type|))
     ((eq option '|values|)
       (setq l (cons '|value| vl))
       (setq option '|value|)))
    (cond
     ((eq option '|abbreviations|)
      (if (null vl)
        (|listConstructorAbbreviations|)
        (dolist (v vl) (|abbQuery| (|opOf| v)))))
     ((eq option '|operations|) (|displayOperations| vl))
     ((eq option '|macros|) (|displayMacros| vl))
     ((eq option '|names|) (|displayWorkspaceNames|))
     (t (|displayProperties| option l))))
```

```
   (|sayMessage|
    (append
      '("  )display keyword arguments are")
      (mapcar #'(lambda (x) (format nil "~%     ~a" x)) |$displayOptions|)
      (format nil "~%  or abbreviations thereof"))))))
```

———-

26.18.5 defun abbQuery

[getdatabase p1070]
[sayKeyedMsg p39]

— **defun abbQuery** —

```
(defun |abbQuery| (x)
 (let (abb)
  (cond
   ((setq abb (getdatabase x 'abbreviation))
    (|sayKeyedMsg| "%1 abbreviates %2 %3"
     (list abb (getdatabase x 'constructorkind) x)))
   ((setq abb (getdatabase x 'constructor))
    (|sayKeyedMsg| "%1 abbreviates %2 %3"
      (list x (getdatabase abb 'constructorkind) abb)))
   (t
     (|sayKeyedMsg|
      "%1 is neither a constructor name nor a constructor abbreviation."
      (list x))))))
```

———-

26.18.6 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 p771]
[sayKeyedMsg p39]

— **defun displayOperations** —

```
(defun |displayOperations| (l)
 (if l
  (dolist (op l) (|reportOpSymbol| op))
  (if (yesanswer)
   (dolist (op (|allOperations|)) (|reportOpSymbol| op))
   (|sayKeyedMsg|
    (format nil
     "Since you did not respond with y or yes the list of operations will ~
      not be displayed.")
```

```
    nil))))
```

———-

26.18.7 defun yesanswer

This is a trivial function to simplify the logic of displaySpad2Cmd. If the user didn't supply an argument to the)display op command we ask if they wish to have all information about all Axiom operations displayed. If the answer is either Y or YES we return true else nil.

[string2id-n p**??**]
[upcase p1140]
[queryUserKeyedMsg p**??**]

— **defun yesanswer** —

```
(defun yesanswer ()
 (member
  (string2id-n
   (upcase
    (|queryUserKeyedMsg|
     (format nil
      "You have requested that all information about all Axiom operations ~
       (functions) be displayed.  As there are several hundred operations, ~
       please confirm your request by typing y or yes and then pressing ~
       Enter :")
      nil)) 1) '(y yes)))
```

———-

26.18.8 defun displayMacros

[getInterpMacroNames p**??**]
[getParserMacroNames p705]
[remdup p**??**]
[sayBrightly p**??**]
[member p1108]
[displayParserMacro p715]
[seq p**??**]
[exit p**??**]
[displayMacro p706]

— **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 macro
            (cons " is not a known Axiom macro." nil)))))))))
   (setq first t)
   (do ((t1 macros (cdr t1)) (macro nil))
       ((or (atom t1) (progn (setq macro (car t1)) nil)) nil)
    (seq
     (exit
      (cond
       ((|member| macro imacs)
         (cond
          ((|member| macro pmacs) '|iterate|)
          (t
           (cond
            (first
             (|sayBrightly|
              (cons '|%l|
               (cons "System-defined macros:" nil))) (setq first nil)))
            (|displayMacro| macro))))
       ((|member| macro pmacs) '|iterate|)))))
    nil))))
```

———-

26.18.9 defun sayExample

This function expects 2 arguments, the documentation string and the name of the operation. It searches the documentation string for `++X` lines. These lines are examples lines for functions. They look like ordinary `++` comments and fit into the ordinary comment blocks. So, for example, in the plot.spad.pamphlet file we find the following function signature:

```
plot: (F -> F,R) -> %
  ++ plot(f,a..b) plots the function \spad{f(x)}
  ++ on the interval \spad{[a,b]}.
  ++
  ++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
```

```
++X plot(fp,-1.0..1.0)$PLOT
```

This function splits out and prints the lines that begin with `++X`.

A minor complication of printing the examples is that the lines have been processed into internal compiler format. Thus the lines that read:

```
++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
++X plot(fp,-1.0..1.0)$PLOT
```

are actually stored as one long line containing the example lines

```
"\\indented{1}{plot(\\spad{f},{}a..\\spad{b}) plots the function
 \\spad{f(x)}} \\indented{1}{on the interval \\spad{[a,{}b]}.}
 \\blankline
 \\spad{X} fp:=(t:DFLOAT):DFLOAT +-> sin(\\spad{t})
 \\spad{X} plot(\\spad{fp},{}\\spad{-1}.0..1.0)\\$PLOT"
```

So when we have an example line starting with ++X, it gets converted to the compiler to `\spad{X}`. So each example line is delimited by `\spad{X}`.

The compiler also removes the newlines so if there is a subsequent `\spad{X}` in the docstring then it implies multiple example lines and we loop over them, splitting them up at the delimiter.

If there is only one then we clean it up and print it.

[cleanupline p**??**]
[sayNewLine p**??**]

— **defun sayExample** —

```
(defun sayExample (docstring)
 (let (line point)
  (when (setq point (search "spad{X}" docstring))
   (setq line (subseq docstring (+ point 8)))
   (do ((mark (search "spad{X}" line) (search "spad{X}" line)))
     ((null mark))
    (princ (cleanupLine (subseq line 0 mark)))
    (terpri)
    (setq line (subseq line (+ mark 8))))
   (princ (cleanupLine line))
   (terpri)
   (terpri))))
```

————-

26.18.10 defun cleanupLine

This function expects example lines in internal format that has been partially processed to remove the prefix. Thus we get lines that look like:

```
fp:=(t:DFLOAT):DFLOAT +-> sin(\\spad{t})
plot(\\spad{fp},{}\\spad{-1}.0..1.0)\\$PLOT
```

It removes all instances of `{}`, and `\`, and unwraps the `spad{}` call, leaving only the argument.

We return lines that look like:

```
fp:=(t:DFLOAT):DFLOAT +-> sin(t)
plot(fp,-1.0..1.0)$PLOT
```

which is hopefully exactly what the user wrote.

The compiler inserts `{}` as a space so we remove it. We remove all of the `\` characters. We remove all of the `spad{...}` delimiters which will occur around other spad variables. Technically we should search recursively for the matching delimiter rather than the next brace but the problem does not arise in practice.

— defun cleanupLine 0 —

```
(defun cleanupLine (line)
 (do ((mark (search "{}" line) (search "{}" line)))
  ((null mark))
  (setq line
   (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 2)))))
 (do ((mark (search "\\" line) (search "\\" line)))
  ((null mark))
  (setq line
   (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 1)))))
 ; split out \spad{...}
 (do ((mark (search "spad{" line) (search "spad{" line)))
  ((null mark))
  (let (left point mid right)
   (setq left (subseq line 0 mark))
   (setq point (search "}" line :start2 mark))
   (setq mid (subseq line (+ mark 5) point))
   (setq right (subseq line (+ point 1)))
   (setq line (concatenate 'string left mid right))))
 ; split out \spadtype{...}
 (do ((mark (search "spadtype{" line) (search "spadtype{" line)))
  ((null mark))
  (let (left point mid right)
   (setq left (subseq line 0 mark))
   (setq point (search "}" line :start2 mark))
   (setq mid (subseq line (+ mark 9) point))
   (setq right (subseq line (+ point 1)))
   (setq line (concatenate 'string left mid right))))
 line)
```

———-

26.19)edit Command

26.19.1 edit man page

— edit.help —

```
=======================================================================
A.9.  )edit
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )edit [filename]

Command Description:

This command is used to edit files. It works in conjunction with the )read
and )compile commands to remember the name of the file on which you are
working. By specifying the name fully, you can edit any file you wish. Thus

)edit /u/julius/matrix.input

will place you in an editor looking at the file /u/julius/matrix.input. By
default, the editor is vi, but if you have an EDITOR shell environment
variable defined, that editor will be used. When AXIOM is running under the X
Window System, it will try to open a separate xterm running your editor if it
thinks one is necessary. For example, under the Korn shell, if you issue

export EDITOR=emacs

then the emacs editor will be used by )edit.

If you do not specify a file name, the last file you edited, read or compiled
will be used. If there is no ‘‘last file’’ you will be placed in the editor
editing an empty unnamed file.

It is possible to use the )system command to edit a file directly. For
example,

)system emacs /etc/rc.tcpip

calls emacs to edit the file.

Also See:
o )system
o )compile
o )read
```

———-

[7]

26.19.2 defun edit

[editSpad2Cmd p776]

— **defun edit** —

```
(defun |edit| (l) (|editSpad2Cmd| l))
```

————-

26.19.3 defun editSpad2Cmd

[pathname p1103]
[pathnameDirectory p1103]
[pathnameType p1102]
[$FINDFILE p??]
[pathnameName p1102]
[editFile p777]
[updateSourceFiles p778]
[/editfile p755]

— **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))
```

————-

[7] "system" (26.56 p 987) "read" (26.31.2 p 838)

26.19.4 defun Implement the)edit command

[concat p1107]
[namestring p1102]
[pathname p1103]
[obey p??]

— **defun editFile** —

```
(defun |editFile| (file)
 (cond
  ((member (intern "WIN32" (find-package 'keyword)) *features*)
   (obey (concat "notepad " (|namestring| (|pathname| file)))))
  (t
   (obey
    (concat "$AXIOM/lib/SPADEDIT " (|namestring| (|pathname| file)))))))
```

————-

The SPADEDIT command

Axiom execute a shell script called SPADEDIT to open a file using the user's chosen editor. That editor name is, by convention, in the EDITOR shell variable. If that variable is not set we default to the 'vi' editor.

— **spadedit** —

```
#!/bin/sh
# this script is invoked by the spad )edit command
# can be replaced by users favorite editor
# optional second argument should be character offset in file

thefile=$1
if [ ! -f $1 ] ; then
  thefile=$AXIOM/../../src/algebra/$1
else
  thefile=$1
fi


if [ $# = 2 ] ; then
START=`grep -n \^$2\( $thefile | awk -F: '{print $1}'`
else
START=1
fi

if [ ! "$EDITOR" ] ; then
EDITOR=vi
fi

if [ "$DISPLAY" ] ; then
if [ "$EDITOR" = "emacs" ] ; then
emacs  +$START $thefile &
elif [ "$EDITOR" = "vi" ] ; then
```

```
xterm -e vi +$START $thefile &
        else
xterm -e $EDITOR $thefile &
fi
else
$EDITOR $thefile
fi
```

———-

26.19.5 defun updateSourceFiles

[pathname p1103]
[pathnameName p1102]
[pathnameType p1102]
[makeInputFilename p1047]
[member p1108]
[pathnameTypeId p1102]
[insert p??]
[$sourceFiles p??]

— **defun updateSourceFiles** —

```
(defun |updateSourceFiles| (arg)
 (declare (special |$sourceFiles|))
  (setq arg (|pathname| arg))
  (setq arg (|pathname| (list (|pathnameName| arg) (|pathnameType| arg) "*")))
  (when (and (makeInputFilename arg)
             (|member| (|pathnameTypeId| arg) '(boot lisp meta)))
   (setq |$sourceFiles| (|insert| arg |$sourceFiles|)))
  arg)
```

———-

26.20)fin Command

26.20.1 fin 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
```

———-

[8]

26.20.2 defun Exit from the interpreter to lisp

[spad-reader p??]
[eof p??]

— **defun fin 0** —

```
(defun |fin| ()
 (setq *eof* t)
 (throw 'spad_reader nil))
```

———-

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

[8] “pquit” (26.29.2 p 834) “quit” (26.30.2 p 836)

26.21)help Command

26.21.1 help man page

— help.help —

```
=======================================================================
A.12.  )help
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )help
  - )help commandName
  - )help syntax

Command Description:

This command displays help information about system commands. If you issue

)help

then this very text will be shown. You can also give the name or abbreviation
of a system command to display information about it. For example,

)help clear

will display the description of the )clear system command.

The command

)help syntax

will give further information about the Axiom language syntax.

All this material is available in the AXIOM User Guide and in HyperDoc. In
HyperDoc, choose the Commands item from the Reference menu.

=======================================================================
A.1.  Introduction
=======================================================================


System commands are used to perform AXIOM environment management. Among the
commands are those that display what has been defined or computed, set up
multiple logical AXIOM environments (frames), clear definitions, read files
of expressions and commands, show what functions are available, and terminate
AXIOM.

Some commands are restricted: the commands
```

```
)set userlevel interpreter
)set userlevel compiler
)set userlevel development

set the user-access level to the three possible choices. All commands are
available at development level and the fewest are available at interpreter
level. The default user-level is interpreter. In addition to the )set command
(discussed in description of command )set ) you can use the HyperDoc settings
facility to change the user-level. Click on [Settings] here to immediately go
to the settings facility.

Each command listing begins with one or more syntax pattern descriptions plus
examples of related commands. The syntax descriptions are intended to be easy
to read and do not necessarily represent the most compact way of specifying
all possible arguments and options; the descriptions may occasionally be
redundant.

All system commands begin with a right parenthesis which should be in the
first available column of the input line (that is, immediately after the
input prompt, if any). System commands may be issued directly to AXIOM or be
included in .input files.

A system command argument is a word that directly follows the command name
and is not followed or preceded by a right parenthesis. A system command
option follows the system command and is directly preceded by a right
parenthesis. Options may have arguments: they directly follow the option.
This example may make it easier to remember what is an option and what is an
argument:

         )syscmd arg1 arg2 )opt1 opt1arg1 opt1arg2 )opt2 opt2arg1 ...

In the system command descriptions, optional arguments and options are
enclosed in brackets (``['' and ``]''). If an argument or option name is in
italics, it is meant to be a variable and must have some actual value
substituted for it when the system command call is made. For example, the
syntax pattern description

)read fileName []quietly]

would imply that you must provide an actual file name for fileName but need
not use the )quietly option. Thus

)read matrix.input

is a valid instance of the above pattern.

System command names and options may be abbreviated and may be in upper or
lower case. The case of actual arguments may be significant, depending on the
particular situation (such as in file names). System command names and
options may be abbreviated to the minimum number of starting letters so that
the name or option is unique. Thus

)s Integer
```

```
is not a valid abbreviation for the )set command, because both )set and )show
begin with the letter ‘‘s’’. Typically, two or three letters are sufficient
for disambiguating names. In our descriptions of the commands, we have used
no abbreviations for either command names or options.

In some syntax descriptions we use a vertical line ‘‘|’’ to indicate that you
must specify one of the listed choices. For example, in

)set output fortran on | off

only on and off are acceptable words for following boot. We also sometimes
use ‘‘...’’ to indicate that additional arguments or options of the listed
form are allowed. Finally, in the syntax descriptions we may also list the
syntax of related commands.

========================================================================
Other help topics
========================================================================
Available help topics are:

abbreviations assignment blocks     browse     boot       cd
clear         clef       close      collection compile    describe
display       edit       fin        for        frame      help
history       if         iterate    leave      library    lisp
load          ltrace     parallel   pquit      quit       read
repeat        savesystem set        show       spool      suchthat
synonym       system     syntax     trace      undo       what
while

Available algebra help topics are:
```

———-

26.21.2 The top level help command

[helpSpad2Cmd p782]

— **defun help** —

```
(defun |help| (l)
 "The top level help command"
 (|helpSpad2Cmd| l))
```

———-

26.21.3 The top level help command handler

[newHelpSpad2Cmd p783]
[sayKeyedMsg p39]

— **defun helpSpad2Cmd** —

```
(defun |helpSpad2Cmd| (args)
 "The top level help command handler"
 (unless (|newHelpSpad2Cmd| args)
  (|sayKeyedMsg|
   (format nil
    "If the system command or synonym %1 exists, help information is not ~
     available for it. Issue )what commands or )what synonyms to ~
     determine is %1 is a valid name.")
   (cons args nil))))
```

———-

26.21.4 defun newHelpSpad2Cmd

[makeInputFilename p1047]
[obey p??]
[concat p1107]
[namestring p1102]
[make-instream p1045]
[say p??]
[abbreviation? p??]
[poundsign p??]
[sayKeyedMsg p39]
[pname p1106]
[selectOptionLC p728]
[$syscommands p697]
[$useFullScreenHelp p894]

— **defun newHelpSpad2Cmd** —

```
(defun |newHelpSpad2Cmd| (args)
 (let (sarg arg narg helpfile filestream line unabbrev)
 (declare (special $syscommands |$useFullScreenHelp|))
  (when (null args) (setq args (list '?)))
  (if (> (|#| args) 1)
   (|sayKeyedMsg| "The )help system command supports at most one argument."
    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))
    ; expand abbreviations to full constructor names
    (when
     (setq unabbrev (|abbreviation?| (intern narg)))
      (setq narg (symbol-name unabbrev)))
    (setq narg (substitute #\q #\? narg))
```

```
(cond
  ; if the help file does not exist, exit
 ((null (setq helpfile (makeInputFilename (list narg "help"))))
   nil)
  ; if we expect to use full screen help, call SPADEDIT
 (|$useFullScreenHelp|
  (obey (concat "$AXIOM/lib/SPADEDIT " (|namestring| helpfile))) t)
  ; otherwise dump the help file to the console
 (t
  (setq filestream (make-instream helpfile))
  (do ((line (|read-line| filestream nil) (|read-line| filestream nil)))
      ((null line) (shut filestream))
   (say line))))))))
```

———-

26.22)history Command

26.22.1 history man page

— history.help —

```
====================================================================
A.13.  )history
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )history )on
  - )history )off
  - )history )write historyInputFileName
  - )history )show [n] [both]
  - )history )save savedHistoryName
  - )history )restore [savedHistoryName]
  - )history )reset
  - )history )change n
  - )history )memory
  - )history )file
  - %
  - %%(n)
  - )set history on | off

Command Description:

The history facility within AXIOM allows you to restore your environment to
that of another session and recall previous computational results. Additional
commands allow you to review previous input lines and to create an .input
file of the lines typed to AXIOM.

AXIOM saves your input and output if the history facility is turned on (which
is the default). This information is saved if either of

)set history on
)history )on

has been issued. Issuing either

)set history off
)history )off

will discontinue the recording of information.

Whether the facility is disabled or not, the value of % in AXIOM always
refers to the result of the last computation. If you have not yet entered
anything, % evaluates to an object of type Variable('%). The function %% may
be used to refer to other previous results if the history facility is
enabled. In that case, %%(n) is the output from step n if n > 0. If n < 0,
```

```
the step is computed relative to the current step. Thus %%(-1) is also the
previous step, %%(-2), is the step before that, and so on. If an invalid step
number is given, AXIOM will signal an error.

The environment information can either be saved in a file or entirely in
memory (the default). Each frame ( description of command )frame ) has its
own history database. When it is kept in a file, some of it may also be kept
in memory for efficiency. When the information is saved in a file, the name
of the file is of the form FRAME.axh where ``FRAME'' is the name of the
current frame. The history file is placed in the current working directory
(see description of command )cd ). Note that these history database files are
not text files (in fact, they are directories themselves), and so are not in
human-readable format.

The options to the )history command are as follows:

  )change n
    will set the number of steps that are saved in memory to n. This option
    only has effect when the history data is maintained in a file. If you
    have issued )history )memory (or not changed the default) there is no
    need to use )history )change.

  )on
    will start the recording of information. If the workspace is not empty,
    you will be asked to confirm this request. If you do so, the workspace
    will be cleared and history data will begin being saved. You can also
    turn the facility on by issuing )set history on.

  )off
    will stop the recording of information. The )history )show command will
    not work after issuing this command. Note that this command may be issued
    to save time, as there is some performance penalty paid for saving the
    environment data. You can also turn the facility off by issuing )set
    history off.

  )file
    indicates that history data should be saved in an external file on disk.

  )memory
    indicates that all history data should be kept in memory rather than
    saved in a file. Note that if you are computing with very large objects
    it may not be practical to kept this data in memory.

  )reset
    will flush the internal list of the most recent workspace calculations so
    that the data structures may be garbage collected by the underlying Lisp
    system. Like )history )change, this option only has real effect when
    history data is being saved in a file.

  )restore [savedHistoryName]
    completely clears the environment and restores it to a saved session, if
    possible. The )save option below allows you to save a session to a file
    with a given name. If you had issued )history )save jacobi the command
    )history )restore jacobi would clear the current workspace and load the
```

```
    contents of the named saved session. If no saved session name is
    specified, the system looks for a file called last.axh.

  )save savedHistoryName
    is used to save a snapshot of the environment in a file. This file is
    placed in the current working directory (see description of command )cd
    ). Use )history )restore to restore the environment to the state
    preserved in the file. This option also creates an input file containing
    all the lines of input since you created the workspace frame (for
    example, by starting your AXIOM session) or last did a )clear all or
    )clear completely.

  )show [n] [both]
    can show previous input lines and output results. )show will display up
    to twenty of the last input lines (fewer if you haven't typed in twenty
    lines). )show n will display up to n of the last input lines. )show both
    will display up to five of the last input lines and output results. )show
    n both will display up to n of the last input lines and output results.

  )write historyInputFile
    creates an .input file with the input lines typed since the start of the
    session/frame or the last )clear all or )clear completely. If
    historyInputFileName does not contain a period (``.'') in the filename,
    .input is appended to it. For example, )history )write chaos and )history
    )write chaos.input both write the input lines to a file called
    chaos.input in your current working directory. If you issued one or more
    )undo commands, )history )write eliminates all input lines backtracked
    over as a result of )undo. You can edit this file and then use )read to
    have AXIOM process the contents.

Also See:
o )frame
o )read
o )set
o )undo
```

[9]

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()`

26.23 Initialized history variables

The following global variables are used:

`$HistList`, `$HistListLen` and `$HistListAct` which is the actual number of "undoable"

[9] "frame" (3.5.1 p 22) "read" (26.31.2 p 838) "set" (26.51.1 p 962) "undo" (26.59 p 991)

steps)

`$HistRecord` collects the input line, all variable bindings and the output of a step, before it is written to the file histFileName().

`$HiFiAccess` is a flag, which is reset by)history)off

The result of step n can be accessed by `%n`, which is translated into a call of fetchOutput(n). The updateHist is called after every interpreter step. The putHist function records all changes in the environment to `$HistList` and `$HistRecord`.

26.23.1 defvar $oldHistoryFileName

— **initvars** —

```
(defvar |$oldHistoryFileName| '|last| "vm/370 filename name component")
```

———-

26.23.2 defvar $historyFileType

— **initvars** —

```
(defvar |$historyFileType| '|axh|     "vm/370 filename type component")
```

———-

26.23.3 defvar $historyDirectory

— **initvars** —

```
(defvar |$historyDirectory| 'A        "vm/370 filename disk component")
```

———-

26.23.4 defvar $useInternalHistoryTable

— **initvars** —

```
(defvar |$useInternalHistoryTable| t  "t means keep history in core")
```

———-

26.23.5 defun makeHistFileName

[makePathname p1104]

— **defun makeHistFileName** —

```
(defun |makeHistFileName| (fname)
 (|makePathname| fname |$historyFileType| |$historyDirectory|))
```

————-

26.23.6 defun oldHistFileName

[makeHistFileName p789]
[$oldHistoryFileName p788]

— **defun oldHistFileName** —

```
(defun |oldHistFileName| ()
 (declare (special |$oldHistoryFileName|))
 (|makeHistFileName| |$oldHistoryFileName|))
```

————-

26.23.7 defun histFileName

[makeHistFileName p789]
[$interpreterFrameName p34]

— **defun histFileName** —

```
(defun |histFileName| ()
 (declare (special |$interpreterFrameName|))
 (|makeHistFileName| |$interpreterFrameName|))
```

————-

26.23.8 defun histInputFileName

[makePathname p1104]
[$interpreterFrameName p34]
[$historyDirectory p788]

— **defun histInputFileName** —

```
(defun |histInputFileName| (fn)
 (declare (special |$interpreterFrameName| |$historyDirectory|))
 (if (null fn)
   (|makePathname| |$interpreterFrameName| 'input |$historyDirectory|)
   (|makePathname| fn 'input |$historyDirectory|)))
```

———-

26.23.9 defun initHist

[initHistList p790]
[oldHistFileName p789]
[histFileName p789]
[histFileErase p825]
[makeInputFilename p1047]
[$replace p??]
[$useInternalHistoryTable p788]
[$HiFiAccess p895]

— **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|)))))
```

———-

26.23.10 defun initHistList

[$HistListLen p41]
[$HistList p41]
[$HistListAct p42]
[$HistRecord p42]

— **defun initHistList** —

```
(defun |initHistList| ()
 (let (li)
 (declare (special |$HistListLen| |$HistList| |$HistListAct| |$HistRecord|))
   (setq |$HistListLen| 20)
   (setq |$HistList| (list nil))
   (setq li |$HistList|)
   (do ((i 1 (1+ i)))
       ((> i |$HistListLen|) nil)
     (setq li (cons nil li)))
   (rplacd |$HistList| li)
   (setq |$HistListAct| 0)
   (setq |$HistRecord| nil)))
```

———-

26.23.11 The top level history command

[sayKeyedMsg p39]
[historySpad2Cmd p791]
[$options p63]

— **defun history** —

```
(defun |history| (l)
 "The top level history command"
 (declare (special |$options|))
 (if (or l (null |$options|))
  (|sayKeyedMsg|
   (format nil
    "You have not used the correct syntax for the history command. ~
     Issue )help history for more information.")
    nil)
  (|historySpad2Cmd|)))
```

———-

26.23.12 The top level history command handler

[selectOptionLC p728]
[member p1108]
[sayKeyedMsg p39]
[initHistList p790]
[upcase p1140]
[queryUserKeyedMsg p??]
[string2id-n p??]
[histFileErase p825]
[histFileName p789]
[clearSpad2Cmd p742]
[disableHist p812]
[setHistoryCore p794]
[resetInCoreHist p798]
[saveHistory p805]
[showHistory p793]
[changeHistListLen p799]
[restoreHistory p806]
[writeInputLines p797]
[seq p??]
[exit p??]
[$options p63]
[$HiFiAccess p895]
[$IOindex p34]

— **defun historySpad2Cmd** —

```
(defun |historySpad2Cmd| ()
 "The top level history command handler"
 (let (histOptions opts opt optargs x)
 (declare (special |$options| |$HiFiAccess| |$IOindex|))
  (setq histOptions
    '(|on| |off| |yes| |no| |change| |reset| |restore| |write|
      |save| |show| |file| |memory|))
  (setq opts
   (prog (tmp1)
     (setq tmp1 nil)
     (return
      (do ((tmp2 |$options| (cdr tmp2)) (tmp3 nil))
          ((or (atom tmp2)
               (progn
                 (setq tmp3 (car tmp2))
                 nil)
               (progn
                (progn
                 (setq opt (car tmp3))
                 (setq optargs (cdr tmp3))
                 tmp3)
                nil))
             (nreverse0 tmp1))
        (setq tmp1
         (cons
           (cons
             (|selectOptionLC| opt histOptions '|optionError|)
             optargs)
           tmp1))))))
  (do ((tmp4 opts (cdr tmp4)) (tmp5 nil))
      ((or (atom tmp4)
           (progn
             (setq tmp5 (car tmp4))
             nil)
           (progn
            (progn
             (setq opt (car tmp5))
             (setq optargs (cdr tmp5))
             tmp5)
            nil))
       nil)
    (seq
     (exit
      (cond
       ((|member| opt '(|on| |yes|))
          (cond
           (|$HiFiAccess|
             (|sayKeyedMsg| "The history facility is already on." nil))
           ((eql |$IOindex| 1)
             (setq |$HiFiAccess| t)
             (|initHistList|)
             (|sayKeyedMsg| "The history facility is now on." nil))
           (t
             (setq x ; really want to turn history on?
```

```
              (upcase
               (|queryUserKeyedMsg|
                (format nil
                 "Turning on the history facility will clear the contents ~
                  of the workspace. Please enter y or yes if you really ~
                  want to do this:")
                 nil)))
             (cond
              ((member (string2id-n x 1) '(Y YES))
               (|histFileErase| (|histFileName|))
               (setq |$HiFiAccess| t)
               (setq |$options| nil)
               (|clearSpad2Cmd| '(|all|))
               (|sayKeyedMsg| "The history facility is now on." nil)
               (|initHistList|))
              (t
               (|sayKeyedMsg| "The history facility is still off." nil))))))
      ((|member| opt '(|off| |no|))
        (cond
         ((null |$HiFiAccess|)
           (|sayKeyedMsg| "The history facility is already off." nil))
         (t
           (setq |$HiFiAccess| nil)
           (|disableHist|)
           (|sayKeyedMsg| "The history facility is now off." nil))))
      ((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|))
```

———-

26.23.13 defun showHistory

[sayKeyedMsg p39]
[selectOptionLC p728]
[sayMSG p40]
[concat p1107]
[bright p??]
[showInOut p809]
[setIOindex p808]
[showInput p808]
[$printTimeSum p??]
[$evalTimePrint p??]

— **defun showHistory** —

```
(defun |showHistory| (arg)
 (let (|$printTimeSum| |$evalTimePrint| maxi mini arg2 arg1
           nset n showInputOrBoth)
  (declare (special |$printTimeSum| |$evalTimePrint| |$HiFiAccess|))
  (setq |$evalTimePrint| 0)
  (setq |$printTimeSum| 0)
  (cond
    ((null |$HiFiAccess|)
     (|sayKeyedMsg|
      (format nil
       "The history facility command %1 cannot be performed because the ~
        history facility is not on.")
       (list '|show|)))
    (t
      (setq showInputOrBoth '|input|)
      (setq n 20)
      (when arg
        (setq arg1 (car arg))
        (when (integerp arg1)
          (setq n arg1)
          (setq nset t)
          (cond
            ((ifcdr arg) (setq arg1 (cadr arg)))
            (t (setq arg1 nil))))
        (when arg1
          (setq arg2 (|selectOptionLC| arg1 '(|input| |both|) nil))
          (cond
            (arg2
              (cond
                ((and (eq (setq showInputOrBoth arg2) '|both|)
                      (null nset))
                  (setq n 5))))
            (t
              (|sayMSG|
               (|concat| "  " (|bright| arg1)  "is an invalid argument."))))))
      (cond ((not (< n |$IOindex|)) (setq n (- |$IOindex| 1))))
      (setq mini (- |$IOindex| n))
      (setq maxi (- |$IOindex| 1))
      (cond
        ((eq showInputOrBoth '|both|)
          (unwind-protect
            (|showInOut| mini maxi)
            (|setIOindex| (+ maxi 1))))
        (t (|showInput| mini maxi)))))))
```

26.23.14 defun setHistoryCore

We case on the inCore argument value

If history is already on and is kept in the same location as requested (file or memory) then complain.

If history is not in use then start using the file or memory as requested. This is done by simply setting the `$useInternalHistoryTable` to the requested value, where T means use memory and NIL means use a file. We tell the user.

If history should be in memory, that is inCore is not NIL, and the history file already contains information we read the information from the file, store it in memory, and erase the history file. We modify `$useInternalHistoryTable` to T to indicate that we're maintining the history in memory and tell the user.

Otherwise history must be on and in memory. We erase any old history file and then write the in-memory history to a new file

[boot-equal p**??**]
[sayKeyedMsg p39]
[rkeyids p**??**]
[histFileName p789]
[readHiFi p810]
[disableHist p812]
[histFileErase p825]
[rdefiostream p**??**]
[spadrwrite p813]
[object2Identifier p**??**]
[rshut p**??**]
[$useInternalHistoryTable p788]
[$internalHistoryTable p42]
[$HiFiAccess p895]
[$IOindex p34]

— **defun setHistoryCore** —

```
(defun |setHistoryCore| (inCore)
 (let (l vec str n rec)
 (declare (special |$useInternalHistoryTable| |$internalHistoryTable|
     |$HiFiAccess| |$IOindex|))
  (cond
   ((boot-equal inCore |$useInternalHistoryTable|)
     (if inCore
        (|sayKeyedMsg|
         (format nil
          "History information is already being maintained in memory (and ~
           not in an external file).")
          nil)
        (|sayKeyedMsg|
         (format nil
          "History information is already being maintained in an external ~
           file (and not in memory).")
         nil))) ; file history already in use
   ((null |$HiFiAccess|)
     (setq |$useInternalHistoryTable| inCore)
     (if inCore
        (|sayKeyedMsg|
         (format nil
          "When the history facility is active, history information will be ~
            maintained in memory (and not in an external file).")
```

```
        nil)
       (|sayKeyedMsg|
          (format nil
           "When the history facility is active, history information will be ~
            maintained in a file (and not in an internal table).")
        nil)))
  (inCore
    (setq |$internalHistoryTable| nil)
    (cond
     ((not (eql |$IOindex| 0))
       (setq l (length (rkeyids (|histFileName|))))
       (do ((i 1 (1+ i)))
           ((> i l) nil)
         (setq vec (unwind-protect (|readHiFi| i) (|disableHist|)))
         (setq |$internalHistoryTable|
           (cons (cons i vec) |$internalHistoryTable|)))
       (|histFileErase| (|histFileName|))))
    (setq |$useInternalHistoryTable| t)
    (|sayKeyedMsg|
     (format nil
      "When the history facility is active, history information will be ~
       maintained in memory (and not in an external file).")
     nil))
  (t
    (setq |$HiFiAccess| nil)
    (|histFileErase| (|histFileName|))
    (setq str
     (rdefiostream
      (cons
        '(mode . output)
        (cons
          (cons 'file (|histFileName|))
          nil))))
    (do ((tmp0 (reverse |$internalHistoryTable|) (cdr tmp0))
         (tmp1 nil))
        ((or (atom tmp0)
             (progn
               (setq tmp1 (car tmp0))
               nil)
             (progn
              (progn
               (setq n (car tmp1))
               (setq rec (cdr tmp1))
               tmp1)
              nil))
             nil)
      (spadrwrite (|object2Identifier| n) rec str))
    (rshut str)
    (setq |$HiFiAccess| t)
    (setq |$internalHistoryTable| nil)
    (setq |$useInternalHistoryTable| nil)
    (|sayKeyedMsg|
     (format nil
      "When the history facility is active, history information will be ~
```

```
      maintained in a file (and not in an internal table).")
    nil)))))
```

———-

26.23.15 defvar $underbar

Also used in the output routines.

— **initvars** —

```
(defvar underbar "_")
```

———-

26.23.16 defun writeInputLines

[sayKeyedMsg p39]
[throwKeyedMsg p??]
[size p1106]
[concat p1107]
[substring p293]
[readHiFi p810]
[histInputFileName p789]
[histFileErase p825]
[defiostream p1046]
[namestring p1102]
[shut p1046]
[underbar p797]
[$HiFiAccess p895]
[$IOindex p34]

— **defun writeInputLines** —

```
(defun |writeInputLines| (fn initial)
 (let (maxn breakChars vecl k svec done n lineList file inp)
 (declare (special underbar |$HiFiAccess| |$IOindex|))
  (cond
   ((null |$HiFiAccess|)
    (|sayKeyedMsg|
     (format nil
      "The history facility is not on, so the .input file containing your ~
       user input cannot be created.")
     nil))
   ((null fn)
    (|throwKeyedMsg|
      "You must specify a file name to the history write command" nil))
   (t
     (setq maxn 72)
     (setq breakChars (cons '| | (cons '+ nil)))
     (do ((tmp0 (- |$IOindex| 1))
          (i initial (+ i 1)))
```

```
        ((> i tmp0) nil)
      (setq vecl (car (|readHiFi| i)))
      (when (stringp vecl) (setq vecl (cons vecl nil)))
      (dolist (vec vecl)
        (setq n (size vec))
        (do ()
            ((null (> n maxn)) nil)
            (setq done nil)
            (do ((j 1 (1+ j)))
                ((or (> j maxn) (null (null done))) nil)
              (setq k (- (1+ maxn) j))
              (when (member (elt vec k) breakChars)
                (setq svec (concat (substring vec 0 (1+ k)) underbar))
                (setq lineList (cons svec lineList))
                (setq done t)
                (setq vec (substring vec (1+ k) nil))
                (setq n (size vec))))
            (when done (setq n 0)))
        (setq lineList (cons vec lineList))))
    (setq file (|histInputFileName| fn))
    (|histFileErase| file)
    (setq inp
     (defiostream
      (cons
       '(mode . output)
       (cons (cons 'file file) nil)) 255 0))
    (dolist (x (|removeUndoLines| (nreverse lineList)))
      (write-line x inp))
    (cond
     ((not (eq fn '|redo|))
       (|sayKeyedMsg| "Edit %1 to see the saved input lines."
         (list (|namestring| file)))))
    (shut inp)
    nil))))
```

———-

26.23.17 defun resetInCoreHist

[$HistListAct p42]
[$HistListLen p41]
[$HistList p41]

— **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)))
```

———-

26.23.18 defun changeHistListLen

[sayKeyedMsg p39]
[$HistListLen p41]
[$HistList p41]
[$HistListAct p42]

— **defun changeHistListLen** —

```
(defun |changeHistListLen| (n)
 (let (dif l)
 (declare (special |$HistListLen| |$HistList| |$HistListAct|))
 (if (null (integerp n))
   (|sayKeyedMsg|
    (format nil
     "The argument n for )history )change n must be a nonnegative integer ~
      and your argument, %1 , is not one.")
     (list n)) ; only positive integers
   (progn
     (setq dif (- n |$HistListLen|))
     (setq |$HistListLen| n)
     (setq l (cdr |$HistList|))
     (cond
      ((> dif 0)
        (do ((i 1 (1+ i)))
            ((> i dif) nil)
           (setq l (cons nil l))))
      ((minusp dif)
        (do ((tmp0 (- dif))
             (i 1 (1+ i)))
            ((> i tmp0) nil)
           (setq l (cdr l)))
        (cond
         ((> |$HistListAct| n) (setq |$HistListAct| n))
         (t nil))))
     (rplacd |$HistList| l)
     '|done|))))
```

———-

26.23.19 defun updateHist

[startTimingProcess p??]
[updateInCoreHist p800]
[writeHiFi p811]
[disableHist p812]
[updateCurrentInterpreterFrame p27]
[stopTimingProcess p??]
[$IOindex p34]

[$HiFiAccess p895]
[$HistRecord p42]
[$mkTestInputStack p??]
[$currentLine p??]

— **defun updateHist** —

```
(defun |updateHist| ()
 (declare (special |$IOindex| |$HiFiAccess| |$HistRecord| |$mkTestInputStack|
   |$currentLine|))
 (when |$IOindex|
   (|startTimingProcess| '|history|)
   (|updateInCoreHist|)
   (when |$HiFiAccess|
      (unwind-protect (|writeHiFi|) (|disableHist|))
      (setq |$HistRecord| nil))
   (incf |$IOindex|)
   (|updateCurrentInterpreterFrame|)
   (setq |$mkTestInputStack| nil)
   (setq |$currentLine| nil)
   (|stopTimingProcess| '|history|)))
```

———-

26.23.20 defun updateInCoreHist

[$HistList p41]
[$HistListLen p41]
[$HistListAct p42]

— **defun updateInCoreHist** —

```
(defun |updateInCoreHist| ()
 (declare (special |$HistList| |$HistListLen| |$HistListAct|))
  (setq |$HistList| (cdr |$HistList|))
  (rplaca |$HistList| nil)
  (when (> |$HistListLen| |$HistListAct|)
    (setq |$HistListAct| (1+ |$HistListAct|))))
```

———-

26.23.21 defun putHist

[recordOldValue p801]
[get p??]
[recordNewValue p801]
[putIntSymTab p??]
[$HiFiAccess p895]

— **defun putHist** —

```
(defun |putHist| (x prop val e)
```

```
  (declare (special |$HiFiAccess|))
  (when (null (eq x '%)) (|recordOldValue| x prop (|get| x prop e)))
  (when |$HiFiAccess| (|recordNewValue| x prop val))
  (|putIntSymTab| x prop val e))
```

———-

26.23.22 defun recordNewValue

[startTimingProcess p??]
[recordNewValue0 p801]
[stopTimingProcess p??]

— **defun recordNewValue** —

```
(defun |recordNewValue| (x prop val)
  (|startTimingProcess| '|history|)
  (|recordNewValue0| x prop val)
  (|stopTimingProcess| '|history|))
```

———-

26.23.23 defun recordNewValue0

[assq p1110]
[$HistRecord p42]

— **defun recordNewValue0** —

```
(defun |recordNewValue0| (x prop val)
 (let (p1 p2 p)
 (declare (special |$HistRecord|))
   (if (setq p1 (assq x |$HistRecord|))
    (if (setq p2 (assq prop (cdr p1)))
      (rplacd p2 val)
      (rplacd p1 (cons (cons prop val) (cdr p1))))
    (progn
      (setq p (cons x (list (cons prop val))))
      (setq |$HistRecord| (cons p |$HistRecord|))))))
```

———-

26.23.24 defun recordOldValue

[startTimingProcess p??]
[recordOldValue0 p802]
[stopTimingProcess p??]
[assq p1110]

— **defun recordOldValue** —

```
(defun |recordOldValue| (x prop val)
  (|startTimingProcess| '|history|)
  (|recordOldValue0| x prop val)
  (|stopTimingProcess| '|history|))
```

———-

26.23.25 defun recordOldValue0

[$HistList p41]

— **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|)))))
```

———-

26.23.26 defun undoInCore

[undoChanges p803]
[readHiFi p810]
[disableHist p812]
[assq p1110]
[sayKeyedMsg p39]
[putHist p800]
[updateHist p799]
[$HistList p41]
[$HistListLen p41]
[$IOindex p34]
[$HiFiAccess p895]
[$InteractiveFrame p34]

— **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 |$HistListLen|) nil)
   (setq li (cdr li)))
  (|undoChanges| li)
  (setq n (- (- |$IOindex| n) 1))
```

```
  (and
    (> n 0)
    (if |$HiFiAccess|
     (progn
       (setq vec (cdr (unwind-protect (|readHiFi| n) (|disableHist|))))
       (setq val
        (and
         (setq p (assq '% vec))
         (setq p1 (assq '|value| (cdr p)))
         (cdr p1))))
      (|sayKeyedMsg|
       "There is no history file, so value of step %1 is undefined."
       (cons n nil)))) ; no history file
  (setq |$InteractiveFrame| (|putHist| '% '|value| val |$InteractiveFrame|))
  (|updateHist|)))
```

———-

26.23.27 defun undoChanges

[boot-equal p??]
[undoChanges p803]
[putHist p800]
[$HistList p41]
[$InteractiveFrame p34]

— **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|)))))
```

———-

26.23.28 defun undoFromFile

[seq p??]
[exit p??]
[recordOldValue p801]
[recordNewValue p801]
[readHiFi p810]
[disableHist p812]
[putHist p800]
[assq p1110]
[updateHist p799]

[$InteractiveFrame p34]
[$HiFiAccess p895]

— **defun undoFromFile** —

```
(defun |undoFromFile| (n)
 (let (varl prop vec x p p1 val)
 (declare (special |$InteractiveFrame| |$HiFiAccess|))
  (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
           (progn (setq tmp1 (car tmp0)) nil)
           (progn
             (progn
              (setq x (car tmp1))
              (setq varl (cdr tmp1))
              tmp1)
              nil))
            nil)
   (seq
    (exit
     (do ((tmp2 varl (cdr tmp2)) (p nil))
         ((or (atom tmp2) (progn (setq p (car tmp2)) nil)) nil)
      (seq
       (exit
        (progn
         (setq prop (car p))
         (setq val (cdr p))
         (when val
           (progn
             (when (null (eq x '%))
               (|recordOldValue| x prop val))
             (when |$HiFiAccess|
               (|recordNewValue| x prop val))
             (rplacd p nil)))))))))
  (do ((i 1 (1+ i)))
      ((> i n) nil)
    (setq vec
      (unwind-protect (cdr (|readHiFi| i)) (|disableHist|)))
    (do ((tmp3 vec (cdr tmp3)) (p1 nil))
        ((or (atom tmp3) (progn (setq p1 (car tmp3)) nil)) nil)
     (setq x (car p1))
     (do ((tmp4 (cdr p1) (cdr tmp4)) (p2 nil))
         ((or (atom tmp4) (progn (setq p2 (car tmp4)) nil)) nil)
      (setq |$InteractiveFrame|
         (|putHist| x (car p2) (CDR p2) |$InteractiveFrame|)))))
  (setq val
   (and
    (setq p (assq '% vec))
    (setq p1 (assq '|value| (cdr p)))
    (cdr p1)))
  (setq |$InteractiveFrame| (|putHist| '% '|value| val |$InteractiveFrame|))
  (|updateHist|)))
```

————-

26.23.29 defun saveHistory

[sayKeyedMsg p39]
[makeInputFilename p1047]
[histFileName p789]
[throwKeyedMsg p??]
[makeHistFileName p789]
[histInputFileName p789]
[writeInputLines p797]
[histFileErase p825]
[rdefiostream p??]
[spadrwrite0 p813]
[object2Identifier p??]
[rshut p??]
[namestring p1102]
[$seen p??]
[$HiFiAccess p895]
[$useInternalHistoryTable p788]
[$internalHistoryTable p42]

— **defun saveHistory** —

```
(defun |saveHistory| (fn)
 (let (|$seen| savefile inputfile saveStr n rec val)
 (declare (special |$seen| |$HiFiAccess| |$useInternalHistoryTable|
     |$internalHistoryTable|))
 (setq |$seen| (make-hash-table :test #'eq))
 (cond
  ((null |$HiFiAccess|)
    (|sayKeyedMsg|
     "The history facility is not on, so no information can be saved."
     nil)) ; the history file is not on
  ((and (null |$useInternalHistoryTable|)
        (null (makeInputFilename (|histFileName|))))
    (|sayKeyedMsg| "No history information had been saved yet." nil))
  ((null fn)
    (|throwKeyedMsg|
      "You must specify a file name to the history save command"
      nil))
  (t
    (setq savefile (|makeHistFileName| fn))
    (setq inputfile (|histInputFileName| fn))
    (|writeInputLines| fn 1)
    (|histFileErase| savefile)
    (when |$useInternalHistoryTable|
      (setq saveStr
        (rdefiostream
         (cons '(mode . output)
           (cons (cons 'file savefile) nil))))
      (do ((tmp0 (reverse |$internalHistoryTable|) (cdr tmp0))
           (tmp1 nil))
          ((or (atom tmp0)
               (progn (setq tmp1 (car tmp0)) nil)
```

```
              (progn
               (progn
                (setq n (car tmp1))
                (setq rec (cdr tmp1))
                tmp1)
               nil))
              nil)
        (setq val (spadrwrite0 (|object2Identifier| n) rec saveStr))
        (when (eq val '|writifyFailed|)
         (|sayKeyedMsg|
          (format nil
           "Can't save the value of step number %1. You can re-generate ~
            this value by running the input file %2.")
          (list n inputfile))))
      (rshut saveStr))
    (|sayKeyedMsg| "The saved history file is %1 ."
      (cons (|namestring| savefile) nil))
    nil))))
```

———-

26.23.30 defun restoreHistory

[qcdr p??]
[qcar p??]
[identp p1107]
[throwKeyedMsg p??]
[makeHistFileName p789]
[putHist p800]
[makeInputFilename p1047]
[sayKeyedMsg p39]
[namestring p1102]
[clearSpad2Cmd p742]
[histFileName p789]
[histFileErase p825]
[$fcopy p??]
[rkeyids p??]
[readHiFi p810]
[disableHist p812]
[updateInCoreHist p800]
[get p??]
[rempropI p??]
[clearCmdSortedCaches p743]
[$options p63]
[$internalHistoryTable p42]
[$HiFiAccess p895]
[$e p285]
[$useInternalHistoryTable p788]
[$InteractiveFrame p34]
[$oldHistoryFileName p788]

— defun restoreHistory —

```
(defun |restoreHistory| (fn)
 (let (|$options| fnq restfile curfile l oldInternal vec line x a)
 (declare (special |$options| |$internalHistoryTable| |$HiFiAccess| |$e|
     |$useInternalHistoryTable| |$InteractiveFrame| |$oldHistoryFileName|))
 (cond
   ((null fn) (setq fnq |$oldHistoryFileName|))
   ((and (consp fn)
         (eq (qcdr fn) nil)
         (progn
          (setq fnq (qcar fn))
          t)
         (identp fnq))
     (setq fnq fnq))
   (t (|throwKeyedMsg| "%1 is not a valid filename for the history file."
       (cons fnq nil)))) ; invalid filename
 (setq restfile (|makeHistFileName| fnq))
 (if (null (makeInputFilename restfile))
  (|sayKeyedMsg|
   (format nil
    "History information cannot be restored from %1 because the file does ~
     not exist.")
    (cons (|namestring| restfile) nil))
  (progn
    (setq |$options| nil)
    (|clearSpad2Cmd| '(|all|))
    (setq curfile (|histFileName|))
    (|histFileErase| curfile)
    ($fcopy restfile curfile)
    (setq l (length (rkeyids curfile)))
    (setq |$HiFiAccess| t)
    (setq oldInternal |$useInternalHistoryTable|)
    (setq |$useInternalHistoryTable| nil)
    (when oldInternal (setq |$internalHistoryTable| nil))
    (do ((i 1 (1+ i)))
        ((> i l) nil)
      (setq vec (unwind-protect (|readHiFi| i) (|disableHist|)))
      (when oldInternal
        (setq |$internalHistoryTable|
         (cons (cons i vec) |$internalHistoryTable|)))
      (setq line (car vec))
      (dolist (p1 (cdr vec))
        (setq x (car p1))
        (do ((tmp1 (cdr p1) (cdr tmp1)) (p2 nil))
            ((or (atom tmp1) (progn (setq p2 (car tmp1)) nil)) nil)
         (setq |$InteractiveFrame|
           (|putHist| x
            (car p2) (cdr p2) |$InteractiveFrame|))))
      (|updateInCoreHist|))
    (setq |$e| |$InteractiveFrame|)
    (do ((tmp2 (caar |$InteractiveFrame|) (cdr tmp2)) (tmp3 nil))
        ((or (atom tmp2)
```

```
            (progn
              (setq tmp3 (car tmp2))
              nil)
            (progn
             (progn
               (setq a (car tmp3))
               tmp3)
               nil))
            nil)
       (when (|get| a '|localModemap| |$InteractiveFrame|)
          (|rempropI| a '|localModemap|)
          (|rempropI| a '|localVars|)
          (|rempropI| a '|mapBody|)))
    (setq |$IOindex| (1+ l))
    (setq |$useInternalHistoryTable| oldInternal)
    (|sayKeyedMsg|
     "The workspace has been successfully restored from the history file %1 ."
      (cons (|namestring| restfile) nil))
    (|clearCmdSortedCaches|)
     nil))))
```

———-

26.23.31 defun setIOindex

[$IOindex p34]

— **defun setIOindex** —

```
(defun |setIOindex| (n)
  (declare (special |$IOindex|))
  (setq |$IOindex| n))
```

———-

26.23.32 defun showInput

[tab p??]
[readHiFi p810]
[disableHist p812]
[sayMSG p40]

— **defun showInput** —

```
(defun |showInput| (mini maxi)
 (let (vec l)
  (do ((|ind| mini (+ |ind| 1)))
      ((> |ind| maxi) nil)
    (setq vec (unwind-protect (|readHiFi| |ind|) (|disableHist|)))
    (cond
     ((> 10 |ind|) (tab 2))
     ((> 100 |ind|) (tab 1))
```

```
     (t nil))
    (setq l (car vec))
    (if (stringp l)
      (|sayMSG| (list "   [" |ind| "] " (car vec)))
      (progn
       (|sayMSG| (list "   [" |ind| "] "))
       (do ((tmp0 l (cdr tmp0)) (ln nil))
           ((or (atom tmp0) (progn (setq ln (car tmp0)) nil)) nil)
         (|sayMSG| (list "      " ln))))))
```

———-

26.23.33 defun showInOut

[assq p1110]
[spadPrint p??]
[objValUnwrap p462]
[objMode p462]
[readHiFi p810]
[disableHist p812]
[sayMSG p40]

— **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)))))))
```

———-

26.23.34 defun fetchOutput

[boot-equal p??]
[getI p??]
[throwKeyedMsg p??]
[readHiFi p810]
[disableHist p812]
[assq p1110]

— **defun fetchOutput** —

```
(defun |fetchOutput| (n)
 (let (vec Alist val)
```

```
  (cond
   ((and (boot-equal n (- 1)) (setq val (|getI| '% '|value|)))
     val)
   (|$HiFiAccess|
    (setq n
     (cond
      ((minusp n) (+ |$IOindex| n))
      (t n)))
    (cond
     ((>= n |$IOindex|)
       (|throwKeyedMsg|
        "You have not reached step %1, and so its value cannot be supplied."
        (cons n nil)))
     ((> 1 n)
       (|throwKeyedMsg|
        "Cannot supply value for step %1b because 1 is the first step."
        (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| "Step %1 has no value." (cons n nil)))))
        (t (|throwKeyedMsg| "Step %1 has no value." (cons n nil)))))))
   (t (|throwKeyedMsg|
       "The history facility is not on, so you cannot use %%."
       nil))))) ; history not on
```

———-

26.23.35 Read the history file using index n

[assoc p??]
[keyedSystemError p??]
[qcdr p??]
[rdefiostream p??]
[histFileName p789]
[spadrread p814]
[object2Identifier p??]
[rshut p??]
[$useInternalHistoryTable p788]
[$internalHistoryTable p42]

— 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| "Missing element in internal history table." nil)
     (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))
```

———-

26.23.36 Write information of the current step to history file

[rdefiostream p??]
[histFileName p789]
[spadrwrite p813]
[object2Identifier p??]
[rshut p??]
[$useInternalHistoryTable p788]
[$internalHistoryTable p42]
[$IOindex p34]
[$HistRecord p42]
[$currentLine p??]

— **defun writeHiFi** —

```
(defun |writeHiFi| ()
 "Writes information of the current step to history file"
 (let (HiFi)
 (declare (special |$useInternalHistoryTable| |$internalHistoryTable|
     |$IOindex| |$HistRecord| |$currentLine|))
   (if |$useInternalHistoryTable|
     (setq |$internalHistoryTable|
      (cons
       (cons |$IOindex|
        (cons |$currentLine| |$HistRecord|))
        |$internalHistoryTable|))
    (progn
      (setq HiFi
        (rdefiostream
         (cons
           '(mode . output)
            (cons (cons 'file (|histFileName|)) nil))))
      (spadrwrite (|object2Identifier| |$IOindex|)
```

```
        (cons |$currentLine| |$HistRecord|) HiFi)
      (rshut HiFi)))))
```

———-

26.23.37 Disable history if an error occurred

[histFileErase p825]
[histFileName p789]
[$HiFiAccess p895]

— **defun disableHist** —

```
(defun |disableHist| ()
 "Disable history if an error occurred"
 (declare (special |$HiFiAccess|))
 (cond
  ((null |$HiFiAccess|)
    (|histFileErase| (|histFileName|)))
  (t nil)))
```

———-

26.23.38 defun writeHistModesAndValues

[get p??]
[putHist p800]
[$InteractiveFrame p34]

— **defun writeHistModesAndValues** —

```
(defun |writeHistModesAndValues| ()
 (let (a x)
 (declare (special |$InteractiveFrame|))
  (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
           (progn
             (setq tmp1 (car tmp0))
             nil)
           (progn
             (progn
               (setq a (car tmp1))
               tmp1)
           nil))
       nil)
     (cond
      ((setq x (|get| a '|value| |$InteractiveFrame|))
        (|putHist| a '|value| x |$InteractiveFrame|))
      ((setq x (|get| a '|mode| |$InteractiveFrame|))
        (|putHist| a '|mode| x |$InteractiveFrame|))))))
```

———-

Lisplib output transformations

Some types of objects cannot be saved by LISP/VM in lisplibs. These functions transform an object to a writable form and back.

26.23.39 defun spadrwrite0

[safeWritify p815]
[rwrite p813]

— **defun spadrwrite0** —

```
(defun spadrwrite0 (vec item stream)
 (let (val)
  (setq val (|safeWritify| item))
  (if (eq val '|writifyFailed|)
    val
    (progn
     (|rwrite| vec val stream)
     item))))
```

———-

26.23.40 defun Random write to a stream

[rwrite p813]
[pname p1106]
[identp p1107]

— **defun rwrite** —

```
(defun |rwrite| (key val stream)
  (when (identp key) (setq key (pname key)))
  (rwrite key val stream))
```

———-

26.23.41 defun spadrwrite

[spadrwrite0 p813]
[throwKeyedMsg p??]

— **defun spadrwrite** —

```
(defun spadrwrite (vec item stream)
 (let (val)
  (setq val (spadrwrite0 vec item stream))
  (if (eq val '|writifyFailed|)
   (|throwKeyedMsg| "The value specified cannot be saved to a file." nil)
   item)))
```

———-

26.23.42 defun spadrread

[dewritify p823]
[rread p814]

— **defun spadrread** —

```
(defun spadrread (vec stream)
 (|dewritify| (|rread| vec stream nil)))
```

———-

26.23.43 defun Random read a key from a stream

RREAD takes erroval to return if key is missing

[rread p814]
[identp p1107]
[pname p1106]

— **defun rread** —

```
(defun |rread| (key rstream errorval)
  (when (identp key) (setq key (pname key)))
  (rread key rstream errorval))
```

———-

26.23.44 defun unwritable?

[vecp p**??**]
[placep p1161]

— **defun unwritable?** —

```
(defun |unwritable?| (ob)
 (cond
  ((or (consp ob) (simple-vector-p ob)) nil)
  ((or (compiled-function-p ob) (hash-table-p ob)) t)
  ((or (placep ob) (readtablep ob)) t)
  ((floatp ob) t)
  (t nil)))
```

———-

26.23.45 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 p39]
[$writifyComplained p??]

— **defun writifyComplain** —

```
(defun |writifyComplain| (s)
 (declare (special |$writifyComplained|))
 (unless |$writifyComplained|
  (setq |$writifyComplained| t)
  (|sayKeyedMsg|
   (format nil
    "A value containing a %1 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 )history )off")
    (list s)))) ; cannot save value
```

———-

26.23.46 defun safeWritify

[writifyTag p??]
[writify p818]

— **defun safeWritify** —

```
(defun |safeWritify| (ob)
 (catch '|writifyTag| (|writify| ob)))
```

———-

26.23.47 defun writify,writifyInner

[writifyTag p??]
[seq p??]
[exit p??]
[hget p1105]
[qcar p??]
[qcdr p??]
[spadClosure? p819]
[writify,writifyInner p815]
[hput p1105]
[qrplaca p??]
[qrplacd p??]
[vecp p??]
[isDomainOrPackage p94]
[mkEvalable p994]

[devaluate p??]
[qvmaxindex p??]
[qsetvelt p??]
[qvelt p??]
[constructor? p??]
[hkeys p1105]
[hashtable-class p??]
[placep p1161]
[boot-equal p??]
[$seen p??]
[$NonNullStream p819]
[$NullStream p819]

— **defun writify,writifyInner** —

```
(defun |writify,writifyInner| (ob)
 (prog (e name tmp1 tmp2 tmp3 x qcar qcdr d n keys nob)
  (declare (special |$seen| |$NonNullStream| |$NullStream|))
  (return
   (seq
    (when (null ob) (exit nil))
    (when (setq e (hget |$seen| ob)) (exit e))
    (when (consp ob)
     (exit
      (seq
       (setq qcar (qcar ob))
       (setq qcdr (qcdr ob))
       (when (setq name (|spadClosure?| ob))
        (exit
         (seq
          (setq d (|writify,writifyInner| (qcdr ob)))
          (setq nob
           (cons 'writified!!
            (cons 'spadclosure
             (cons d (cons name nil)))))
          (hput |$seen| ob nob)
          (hput |$seen| nob nob)
          (exit nob))))
       (when
        (and
         (and (consp ob)
              (eq (qcar ob) 'lambda-closure)
              (progn
               (setq tmp1 (qcdr ob))
               (and (consp tmp1)
                    (progn
                     (setq tmp2 (qcdr tmp1))
                     (and
                      (consp tmp2)
                      (progn
                       (setq tmp3 (qcdr tmp2))
                       (and (consp tmp3)
                            (progn
```

```
                        (setq x (qcar tmp3))
                        t)))))))) x)
     (exit
      (throw '|writifyTag| '|writifyFailed|)))
    (setq nob (cons qcar qcdr))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (setq qcar (|writify,writifyInner| qcar))
    (setq qcdr (|writify,writifyInner| qcdr))
    (qrplaca nob qcar)
    (qrplacd nob qcdr)
    (exit nob))))
(when (simple-vector-p 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)))))
```

———-

26.23.48 defun writify

[ScanOrPairVec p824]
[function p??]
[writify,writifyInner p815]
[$seen p??]
[$writifyComplained p??]

— **defun writify** —

```
(defun |writify| (ob)
 (let (|$seen| |$writifyComplained|)
```

```
(declare (special |$seen| |$writifyComplained|))
 (if (null (|ScanOrPairVec| #'|unwritable?| ob))
   ob
   (progn
    (setq |$seen| (make-hash-table :test #'eq))
    (setq |$writifyComplained| nil)
    (|writify,writifyInner| ob)))))
```

———-

26.23.49 defun spadClosure?

[qcar p??]
[bpiname p??]
[qcdr p??]
[vecp p??]

— **defun spadClosure?** —

```
(defun |spadClosure?| (ob)
 (let (fun name vec)
  (setq fun (qcar ob))
  (if (null (setq name (bpiname fun)))
   nil
   (progn
     (setq vec (qcdr ob))
     (if (null (simple-vector-p vec))
      nil
      name)))))
```

———-

26.23.50 defvar $NonNullStream

— **initvars** —

```
(defvar |$NonNullStream| "NonNullStream")
```

———-

26.23.51 defvar $NullStream

— **initvars** —

```
(defvar |$NullStream| "NullStream")
```

———-

26.23.52 defun dewritify,dewritifyInner

[seq p??]
[exit p??]
[hget p1105]
[intp p??]
[gensymmer p??]
[error p??]
[poundsign p??]
[hput p1105]
[dewritify,dewritifyInner p820]
[concat p1107]
[vmread p1161]
[make-instream p1045]
[qcar p??]
[qcdr p??]
[qrplaca p??]
[qrplacd p??]
[vecp p??]
[qvmaxindex p??]
[qsetvelt p??]
[qvelt p??]
[$seen p??]
[$NullStream p819]
[$NonNullStream p819]

— **defun dewritify,dewritifyInner** —

```
(defun |dewritify,dewritifyInner| (ob)
 (prog (e type oname f vec name tmp1 signif expon sign fval qcar qcdr n nob)
  (declare (special |$seen| |$NullStream| |$NonNullStream|))
  (return
   (seq
    (when (null ob)
      (exit nil))
    (when (setq e (hget |$seen| ob))
      (exit e))
    (when (and (consp ob) (eq (car ob) 'writified!!))
     (exit
      (seq
       (setq type (elt ob 1))
       (when (eq type 'self)
        (exit 'writified!!))
       (when (eq type 'bpi)
        (exit
         (seq
          (setq oname (elt ob 2))
          (setq f
           (seq
            (when (integerp oname) (exit (eval (gensymmer oname))))
            (exit (symbol-function oname))))
          (when (null (compiled-function-p f))
```

```
     (exit (|error| "A required BPI does not exist.")))
    (when (and (> (|#| ob) 3) (not (equal (sxhash f) (elt ob 3))))
     (exit (|error| "A required BPI has been redefined.")))
    (hput |$seen| ob f)
    (exit f))))
 (when (eq type 'hashtable)
  (exit
   (seq
    (setq nob (make-hash-table :test #'equal))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (do ((tmp0 (elt ob 3) (cdr tmp0))
         (k nil)
         (tmp1 (elt ob 4) (cdr tmp1))
         (e nil))
        ((or (atom tmp0)
             (progn
              (setq k (car tmp0))
              nil)
             (atom tmp1)
             (progn
              (setq e (car tmp1))
              nil))
             nil)
     (seq
      (exit
       (hput nob (|dewritify,dewritifyInner| k)
        (|dewritify,dewritifyInner| e)))))
    (exit nob))))
 (when (eq type 'devaluated)
  (exit
   (seq
    (setq nob (eval (|dewritify,dewritifyInner| (elt ob 2))))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (exit nob))))
 (when (eq type 'spadclosure)
  (exit
   (seq
    (setq vec (|dewritify,dewritifyInner| (elt ob 2)))
    (setq name (ELT ob 3))
    (when (null (fboundp name))
     (exit
      (|error|
       (concat "undefined function: " (symbol-name name)))))
    (setq nob (cons (symbol-function name) vec))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (exit nob))))
 (when (eq type 'place)
  (exit
   (seq
    (setq nob (vmread (make-instream nil)))
    (hput |$seen| ob nob)
```

```
      (hput |$seen| nob nob)
      (exit nob))))
   (when (eq type 'readtable)
    (exit (|error| "Cannot de-writify a read table.")))
   (when (eq type 'nullstream)
    (exit |$NullStream|))
   (when (eq type 'nonnullstream)
    (exit |$NonNullStream|))
   (when (eq type 'float)
    (exit
     (seq
      (progn
       (setq tmp1 (cddr ob))
       (setq fval (car tmp1))
       (setq signif (cadr tmp1))
       (setq expon (caddr tmp1))
       (setq sign (cadddr tmp1))
       tmp1)
      (setq fval (scale-float (float signif fval) expon))
      (when (minusp sign)
       (exit (- fval)))
      (exit fval))))
   (exit (|error| "Unknown type to de-writify.")))))
(when (consp ob)
 (exit
  (seq
   (setq qcar (qcar ob))
   (setq qcdr (qcdr ob))
   (setq nob (cons qcar qcdr))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (qrplaca nob (|dewritify,dewritifyInner| qcar))
   (qrplacd nob (|dewritify,dewritifyInner| qcdr))
   (exit nob))))
(when (simple-vector-p ob)
 (exit
  (seq
   (setq n (qvmaxindex ob))
   (setq nob (make-array (1+ n)))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (do ((i 0 (1+ i)))
       ((> i n) nil)
    (seq
     (exit
      (qsetvelt nob i
        (|dewritify,dewritifyInner| (qvelt ob i))))))
   (exit nob))))
(exit ob)))))
```

26.23.53 defun dewritify

[ScanOrPairVec p824]
[function p??]
[dewritify,dewritifyInner p820]
[$seen p??]

— **defun dewritify** —

```
(defun |dewritify| (ob)
 (let (|$seen|)
 (declare (special |$seen|))
   (if (null (|ScanOrPairVec| #'(lambda (a) (eq a 'writified!!)) ob))
    ob
    (progn
      (setq |$seen| (make-hash-table :test #'eq))
      (|dewritify,dewritifyInner| ob)))))
```

———-

26.23.54 defun ScanOrPairVec,ScanOrInner

[ScanOrPairVecAnswer p??]
[hget p1105]
[hput p1105]
[ScanOrPairVec,ScanOrInner p823]
[qcar p??]
[qcdr p??]
[vecp p??]
[$seen p??]

— **defun ScanOrPairVec,ScanOrInner** —

```
(defun |ScanOrPairVec,ScanOrInner| (f ob)
  (declare (special |$seen|))
  (when (hget |$seen| ob) nil)
  (when (consp ob)
    (hput |$seen| ob t)
    (|ScanOrPairVec,ScanOrInner| f (qcar ob))
    (|ScanOrPairVec,ScanOrInner| f (qcdr ob)))
  (when (simple-vector-p ob)
     (hput |$seen| ob t)
     (do ((tmp0 (- (|#| ob) 1)) (i 0 (1+ i)))
         ((> i tmp0) nil)
       (|ScanOrPairVec,ScanOrInner| f (elt ob i))))
  (when (funcall f ob) (throw '|ScanOrPairVecAnswer| t))
  nil)
```

———-

26.23.55 defun ScanOrPairVec

[ScanOrPairVecAnswer p??]
[ScanOrPairVec,ScanOrInner p823]
[$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))))
```

———-

26.23.56 defun gensymInt

[gensymp p??]
[error p??]
[pname p1106]
[charDigitVal p824]

— **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 (- (|#| p) 1)) (i 2 (1+ i)))
         ((> i tmp0) nil)
       (setq n (+ (* 10 n) (|charDigitVal| (elt p i)))))
    n))))
```

———-

26.23.57 defun charDigitVal

[error p??]

— **defun charDigitVal** —

```
(defun |charDigitVal| (c)
 (let (digits n)
  (setq digits "0123456789")
  (setq n (- 1))
  (do ((tmp0 (- (|#| 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)))
```

———-

26.23.58 defun histFileErase

— **defun histFileErase** —

```
(defun |histFileErase| (file)
 (when (probe-file file) (delete-file file)))
```

———-

26.24)include Command

26.24.1 include 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.
```

———-

26.24.2 defun ncloopInclude1

[ncloopIncFileName p826]
[ncloopInclude p827]

— **defun ncloopInclude1** —

```
(defun |ncloopInclude1| (name n)
 (let (a)
  (if (setq a (|ncloopIncFileName| name))
    (|ncloopInclude| a n)
    n)))
```

———-

26.24.3 Returns the first non-blank substring of the given string

[incFileName p827]
[concat p1107]

— **defun ncloopIncFileName** —

```
(defun |ncloopIncFileName| (string)
 "Returns the first non-blank substring of the given string"
 (let (fn)
  (unless (setq fn (|incFileName| string))
   (write-line (concat string " not found")))
 fn))
```

———-

26.24.4 Open the include file and read it in

The ncloopInclude0 function is part of the parser and lives in int-top.boot.

[ncloopInclude0 p329]

— **defun ncloopInclude** —

```
(defun |ncloopInclude| (name n)
 "Open the include file and read it in"
 (with-open-file (st name) (|ncloopInclude0| st name n)))
```

———-

26.24.5 Return the include filename

Given a string we return the first token from the string which is the first non-blank substring.
[incBiteOff p827]

— **defun incFileName** —

```
(defun |incFileName| (x)
 "Return the include filename"
  (car (|incBiteOff| x)))
```

———-

26.24.6 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) "")))))
```

———-

26.25)library Command

26.25.1 library 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

10

[10] "cd" (26.11 p 739) "frame" (3.5.1 p 22) "set" (26.51.1 p 962)

26.26)license Command

26.26.1 license man page

— **license.help** —

```
====================================================================
A.15.  )license
====================================================================

Command Syntax:

  -  )license

Command Description:

This command displays the Axiom license.

Also See:
o )trademark
```

———-

26.26.2 defun license

[obey p**??**]
[concat p1107]
[getenviron p291]

— **defun license** —

```
(defun |license| (l)
 (declare (ignore l))
 (obey (concat "cat " (getenviron "AXIOM") "/doc/spadhelp/copyright.help")))
```

———-

26.27)lisp Command

26.27.1 lisp 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
```

———-

[11]

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

[11] "system" (26.56 p 987) "boot" (26.4 p 734) "fin" (26.20.2 p 779)

26.28)ltrace Command

26.28.1 ltrace 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 functions.
It is not supported for general use.

Also See:
o )lisp
o )trace
```

————-

12

26.28.2 defun The top level)ltrace function

[trace p67]

— defun ltrace —

```
(defun |ltrace| (arg) (|trace| arg))
```

————-

12 “lisp” (26.27 p 831) “trace” (7.4.1 p 67)

26.29)pquit Command

26.29.1 pquit man page

— pquit.help —

```
====================================================================
A.18.  )pquit
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )pquit

Command Description:

This command is used to terminate AXIOM and return to the operating system.
Other than by redoing all your computations or by using the )history )restore
command to try to restore your working environment, you cannot return to
AXIOM in the same state.

)pquit differs from the )quit in that it always asks for confirmation that
you want to terminate AXIOM (the ‘‘p’’ is for ‘‘protected’’). When you enter
the )pquit command, AXIOM responds

      Please enter y or yes if you really want to leave the interactive
                environment and return to the operating system:

If you respond with y or yes, you will see the message

            You are now leaving the AXIOM interactive environment.
    Issue the command axiom to the operating system to start a new session.

and AXIOM will terminate and return you to the operating system (or the
environment from which you invoked the system). If you responded with
something other than y or yes, then the message

        You have chosen to remain in the AXIOM interactive environment.

will be displayed and, indeed, AXIOM would still be running.

Also See:
o )fin
o )history
o )close
o )quit
o )system

                ————————-
```

13

26.29.2 The top level pquit command

[pquitSpad2Cmd p834]

— **defun pquit** —

```
(defun |pquit| ()
 "The top level pquit command"
 (|pquitSpad2Cmd|))
```

———-

26.29.3 The top level pquit command handler

[quitSpad2Cmd p836]
[$quitCommandType p955]

— **defun pquitSpad2Cmd** —

```
(defun |pquitSpad2Cmd| ()
 "The top level pquit command handler"
 (let ((|$quitCommandType| '|protected|))
  (declare (special |$quitCommandType|))
  (|quitSpad2Cmd|)))
```

———-

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

[13] "fin" (26.20.2 p 779) "history" (26.23.11 p 791) "close" (26.13.3 p 751) "quit" (26.30.2 p 836) "system" (26.56 p 987)

26.30)quit Command

26.30.1 quit man page

— quit.help —

```
========================================================================
A.19.  )quit
========================================================================

User Level Required:  interpreter

Command Syntax:

  - )quit
  - )set quit protected | unprotected

Command Description:

This command is used to terminate AXIOM and return to the operating system.
Other than by redoing all your computations or by using the )history )restore
command to try to restore your working environment, you cannot return to
AXIOM in the same state.

)quit differs from the )pquit in that it asks for confirmation only if the
command

)set quit protected

has been issued. Otherwise, )quit will make AXIOM terminate and return you to
the operating system (or the environment from which you invoked the system).

The default setting is )set quit protected so that )quit and )pquit behave in
the same way. If you do issue

)set quit unprotected

we suggest that you do not (somehow) assign )quit to be executed when you
press, say, a function key.

Also See:
o )fin
o )history
o )close
o )pquit
o )system
```

————-

14

[14] "fin" (26.20.2 p 779) "history" (26.23.11 p 791) "close" (26.13.3 p 751) "pquit" (26.29.2 p 834) "system" (26.56 p 987)

26.30.2 The top level quit command

[quitSpad2Cmd p836]

— defun quit —

```
(defun |quit| ()
 "The top level quit command"
 (|quitSpad2Cmd|))
```

———-

26.30.3 The top level quit command handler

[upcase p1140]
[queryUserKeyedMsg p??]
[string2id-n p??]
[leaveScratchpad p836]
[sayKeyedMsg p39]
[tersyscommand p704]
[$quitCommandType p955]

— defun quitSpad2Cmd —

```
(defun |quitSpad2Cmd| ()
 "The top level quit command handler"
 (declare (special |$quitCommandType|))
 (if (eq |$quitCommandType| '|protected|)
  (let (x)
   (setq x
    (upcase
     (|queryUserKeyedMsg|
      (format nil
       "Please enter y or yes if you really want to leave the interactive ~
        environment and return to the operating system:")
       nil)))
   (when (member (string2id-n x 1) '(y yes)) (|leaveScratchpad|))
   (|sayKeyedMsg|
    "You have chosen to remain in the Axiom interactive environment." nil)
   (tersyscommand))
  (|leaveScratchpad|)))
```

———-

26.30.4 Leave the Axiom interpreter

— defun leaveScratchpad —

```
(defun |leaveScratchpad| ()
 "Leave the Axiom interpreter"
 (bye))
```

———-

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

26.31)read Command

26.31.1 read man page

— read.help —

```
====================================================================
A.20.  )read
====================================================================

User Level Required:  interpreter

Command Syntax:

  -  )read [fileName]
  -  )read [fileName] []quiet] []ifthere]

Command Description:

This command is used to read .input files into AXIOM. The command

)read matrix.input

will read the contents of the file matrix.input into AXIOM. The ``.input''
file extension is optional. See the AXIOM User Guide index for more
information about .input files.

This command remembers the previous file you edited, read or compiled. If you
do not specify a file name, the previous file will be read.

The )ifthere option checks to see whether the .input file exists. If it does
not, the )read command does nothing. If you do not use this option and the
file does not exist, you are asked to give the name of an existing .input
file.

The )quiet option suppresses output while the file is being read.

Also See:
o )compile
o )edit
o )history
```

————-

15

26.31.2 defun The)read command

[readSpad2Cmd p839]

15 “edit” (26.19.2 p 776) “history” (26.23.11 p 791)

— defun read —

```
(defun |read| (arg) (|readSpad2Cmd| arg))
```

————-

26.31.3 defun Implement the)read command

[selectOptionLC p728]
[optionError p702]
[pathname p1103]
[pathnameTypeId p1102]
[makePathname p1104]
[pathnameName p1102]
[mergePathnames p1103]
[findfile p??]
[throwKeyedMsg p??]
[namestring p1102]
[upcase p1140]
[member p1108]
[/read p840]
[$InteractiveMode p284]
[$findfile p??]
[$UserLevel p961]
[$options p63]
[/editfile p755]

— 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 |$options|)
   (setq fullopt
    (|selectOptionLC| (car opt) '(|quiet| |test| |ifthere|) '|optionError|))
    (cond
     ((eq fullopt '|ifthere|) (setq ifthere t))
     ((eq fullopt '|quiet|)   (setq quiet t))))
  (setq ef (or (|pathname| /editfile) ""))
  (when (eq (|pathnameTypeId| ef) 'spad)
    (setq ef (|makePathname| (|pathnameName| ef) "*" "*")))
  (if arg
   (setq arg (|mergePathnames| (|pathname| arg) ef))
   (setq arg ef))
  (setq devFTs '("input" "INPUT" "boot" "BOOT" "lisp" "LISP"))
  (setq fileTypes
   (cond
    ((eq |$UserLevel| '|interpreter|) '("input" "INPUT"))
    ((eq |$UserLevel| '|compiler|)    '("input" "INPUT"))
```

```
    (t devFTs)))
  (setq ll ($findfile arg fileTypes))
  (unless ll
    (if ifthere
     (return nil)
     (|throwKeyedMsg| "The file %1 is needed but does not exist."
      (list (|namestring| arg)))))
  (setq ll (|pathname| ll))
  (setq ft (|pathnameType| ll))
  (setq upft (upcase ft))
  (cond
   ((null (|member| upft fileTypes))
     (setq fs (|namestring| arg))
     (if (|member| upft devFTs)
      (|throwKeyedMsg|
       (format nil
        "You cannot )read the file %1 because your user-level is not is ~
         not high enough. For more information about your user-level, issue ~
         )set userlevel.")
       (list fs))
      (|throwKeyedMsg|
       (format nil
        "You cannot )read the file %1 because it is not suitable for ~
         reading by Axiom. Note that files with file extension .spad ~
         can now only be compiled with the )compile system command.")
       (list fs))))
   (t
    (setq /editfile ll)
    (when (string= upft "BOOT") (setq |$InteractiveMode| nil))
    (/read ll quiet)))))
```

———-

26.31.4 defun /read

[/read /rf (vol9)]
[/read /rq (vol9)]
[/editfile p755]

— **defun /read** —

```
(defun /read (l q)
 (declare (special /editfile))
  (setq /editfile l)
  (cond
    (q (/rq))
    (t (/rf)) )
  (flag |boot-NewKEY| 'key)
  (|terminateSystemCommand|)
  (|spadPrompt|))
```

26.32)regress Command

26.32.1 regress 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:

  )set break resume
  )spool foo.output
  )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:

  Starts dribbling to foo.output (2012/2/28, 12:25:7).
  )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
  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 command** —

```
(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")))))
```

———-

26.32.2 The regress function details

This is the regression test mechanism. The input files have been rewritten to have a standard structure. This fixed format identifies the tests within a file. Each test is run and any mismatch between the actual and expected results is reported.

In order to regression test axiom results we created a standard file format. This format has 3 kinds of markers:

- "–S" marker which must have a integer test number
- "–R" marker lines, one per expected output from axiom
- "–E" marker which has an integer matching the preceeding "–S"
- "–I" marker ignores the line, useful for gensyms and random

Because these markers use Axiom's standard comment prefix they are valid lines in input files and are ignored by the ")read" command. They are simply copied to the output file. This allows us to include the expected output in the output file so we can compare what Axiom computes with what it should compute.

To create these regression files all you need to do is create an input file and run it through Axiom. Then, for each test case in the file you mark it up by putting a "–S number" **before** the Axiom input line. You put "–R" prefixes on each line of Axiom output, including the blank lines. Then you put a "–E number" line after the last output line, usually the `Type:` line. This newly marked-up input file is now a regression test.

To actually run the regression test you simply include the marked up the input file in the `src/input` subdirectory. This file will automatically be run at build time and any failing tests will be marked. This code will ignore any input that does not contain proper regression markups.

Ideally the regression test files should be pamphlet files that explain the content and purpose of each regression test case.

Thus you run the marked-up input file `foo.input` and spool the result to `foo.output` and then run the lisp function
`(regress ‘‘foo.output’’)`

If the file does not contain proper regression markups it is ignored. Comments or any other commands in the file that are not surrounded by "–S" and "–E" boundaries are ignored.

26.32.3 defvar *all-tests-ran*

This variable is used to check whether all of the tests actually ran. This is needed to see if the execution ended early.

— **initvars** —

```
(defvar *all-tests-ran* nil "true implies that all tests ran")
```

————-

26.32.4 defun Scan a spool output file for failures

This function takes an output file which has been created by the Axiom `)spool` command and looks for regression test markups. Each regression test is checked against the actual result and any failures are marked.

[getspoolname p847]
[findnexttest p847]
[testpassed p848]
[*all-tests-ran* p846]

— **defun regress** —

```
(defun regress (infile)
 (let (name comment test (count 0) (passed 0) (failed 0))
 (declare (special *all-tests-ran*))
  (setq *all-tests-ran* nil)
  (with-open-file (stream infile :direction :input)
   (setq name (getspoolname stream))
   (when name
    (format t "testing ~a~%" name)
    (loop
     (setq *ok* nil)
     (multiple-value-setq (comment test) (findnexttest stream))
     (unless comment (return))
     (setq count (+ count 1))
     (if (testpassed test)
      (progn
       (setq passed (+ passed 1))
       (format t "passed ~a ~a~%" name comment))
      (progn
       (setq failed (+ failed 1))
       (format t "FAILED ~a ~a~%" name comment))))
   (if (= failed 0)
    (format t "regression result passed ~a of ~a stanzas ~Tfile ~a~%"
     passed count name)
    (format t "regression result FAILED ~a of ~a stanzas ~Tfile ~a~%"
     failed count name))
   (unless *all-tests-ran*
```

```
    (format t "regression result FAILED early exit in file ~a?~%" name))))))
```

———-

26.32.5 defun Parse test name from the spool command

We need to parse out the name of the test. The ")spool" command writes a line into the output file containing the name of the test. We parse out the name of the test from this line.

— **defun getspoolname 0** —

```
(defun getspoolname (stream)
 (let (line point)
  (setq line (read-line stream))
  (setq point (position #\. line))
  (if (or (null point)
      (< (length line) 30)
      (not (string= (subseq line (+ point 1) (+ point 7)) "output")))
    nil
  (subseq line 20 point))))
```

———-

26.32.6 defun Find the next –S marker

We need to break the file into separate test cases. This routine looks for the "–S" line which indicates a test is starting. It collects up input lines until it encounters the "–E" line marking the end of the test case. These lines are returned as a list of strings.

[testnumberp p847]

— **defun findnexttest** —

```
(defun findnexttest (stream)
 (let (teststart result)
  (do ((line (read-line stream nil 'done) (read-line stream nil 'done)))
      ((or (eq line 'done) (endedp line))
        (values (if line teststart) result))
   (if teststart
    (push line result)
    (setq teststart (testnumberp line))))))
```

———-

26.32.7 defun Parse out the test number from –S lines

The "–S" line has a test number on the line. We parse out the test number for printing. [startp p850]

— **defun testnumberp** —

```
(defun testnumberp (oneline)
 (when (startp oneline) (subseq oneline 3)))
```

———-

26.32.8 defvar *ok*

We can mark a test as always ok by putting the word "ok" anywhere on the start line. The regress function resets this value. The startp function checks the –S line for the word "ok". If found, it sets this value to true which causes a failing test to be considered as passed.

— **initvars** —

```
(defvar *ok* nil "did we mark this test as always ok?")
```

———-

26.32.9 defun Compare the computed and expected results

This routine takes the test input, passes it to split to clean up and break into two lists, and then compares the resulting lists element by element, complaining about any mismatches. The result is either true if everything passes or false if a mismatch occurs.

A test line can also be considered at passing if the expected line is the string "ignore".

The ok variable allows us to mark failing tests as "ok" because we expect the test might fail due to random values or testing known bugs against expected output. We filter these tests marked "ok" so they do not count as "real" failures.

[split p849]
[*ok* p848]

— **defun testpassed** —

```
(defun testpassed (test)
 (let (answer expected (passed t) mismatchedLines)
  (declare (special *ok*))
  (multiple-value-setq (answer expected) (split test))
  (dotimes (i (length answer))
   (unless
    (or (string= (first expected) "ignore")
        (string= (first expected) (first answer)))
    (unless *ok* (setq passed nil))
    (push (cons (first expected) (first answer)) mismatchedLines))
   (pop answer)
   (pop expected))
  (when mismatchedLines
   (dolist (pair mismatchedLines)
    (format t "expected:~s~%     got:~s~%"  (car pair) (cdr pair))))
  passed))
```

———-

26.32.10 defun Split the calculated and expect results into lists

We have a list containing all of the lines in a test. The input is of the form:

```
("--R                                              Type: List Integer"
 "--R   (1)  [1,4,2,- 6,0,3,5,4,2,3]"
 "--R"
 "--R "
 "                                          Type: List Integer"
 "   (1)  [1,4,2,- 6,0,3,5,4,2,3]"
 ""
 " "
 "l := [1,4,2,-6,0,3,5,4,2,3]")
```

It removes the "–R" prefix from the result strings and generates two hopefully equal-length lists, thus:

```
("                                          Type: List Integer"
 "   (1)  [1,4,2,- 6,0,3,5,4,2,3]"
 ""
 " ")
("                                          Type: List Integer"
 "   (1)  [1,4,2,- 6,0,3,5,4,2,3]"
 ""
 " "))
```

Thus the first line is the start line, the second line is the Axiom input line, followed by the Axiom output. Then we have the lines marked "–R" which are the expected result. We split these into two separate lists and throw way the lines that are the start and end lines.

Once we have classified all of the lines we need to throw away the input lines. By assumption there will be more answer lines than expected lines because the input lines are included. And given the way we process the file these input lines are on the top of the answer stack. Since the number of answer lines should equal the number of expected lines we pop the stack until the numbers are equal.

Each element of the answer list should be `string=` to the corresponding element of the result list.

If the input line starts with "–I" we push the string "ignore". This is useful for handling random results or gensym symbols.

[startp p850]
[endedp p850]
[ignorep p851]
[resultp p851]

— **defun split** —

```
(defun split (test)
 (let (answer (acnt 0) expected (ecnt 0))
  (dolist (oneline test)
   (cond
    ((startp oneline))
    ((endedp oneline))
    ((ignorep oneline)
      (setq ecnt (+ ecnt 1))
```

```
      (push "ignore" expected))
    ((resultp oneline)
      (setq ecnt (+ ecnt 1))
      (push (subseq oneline 3) expected))
    (t
      (setq acnt (+ acnt 1))
      (push oneline answer))))
  (dotimes (i (- acnt ecnt)) (pop answer))
  (values (nreverse answer) (nreverse expected))))
```

———-

26.32.11 defun Returns true on –S lines

This test returns true if we have a “start” line. That is, a line with a “–S” prefix.

The *all-tests-ran* variable is true if the start line is of the form ”–S N of M” and N=M, that is, it checks that all tests were performed since this should only occur on the last start line. This will detect “premature exit” in processing.

If a test is failing because of random input values or we want the test to fail but not to count toward failing values then put the string “ok” somewhere on the “–S” line as in:

```
--S 29 of 42 fails due to random values but that is ok
```

[lastcount p851]
[*ok* p848]

— **defun startp** —

```
(defun startp (oneline)
 (let (result)
 (declare (special *ok*))
  (when
   (setq result
    (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--S")))
     (setq *ok* (search "ok" oneline))
     (setq *all-tests-ran* (lastcount oneline)))
  result))
```

———-

26.32.12 defun Returns true on –E lines

This test returns true if we have a “ended” line. That is, a line with a “–E” prefix.

— **defun endedp 0** —

```
(defun endedp (oneline)
 (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--E")))
```

———-

26.32.13 defun Returns true on –R lines

This test returns true if we have a "results" line. That is, a line with a "–R" prefix.

— **defun resultp 0** —

```
(defun resultp (oneline)
 (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--R")))
```

———-

26.32.14 defun Returns true on –I lines

This test returns true if we have an "ignore" line. That is, a line with a "–I" prefix.

— **defun ignorep 0** —

```
(defun ignorep (oneline)
 (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--I")))
```

———-

26.32.15 defun Check the last –S line ran

If the "–S" line has the format "–S n of m" we return true if n=m, false otherwise. Thus,

```
 "--S"              => nil
 "--S 1 of 4"       => nil
 "--S 10 of 40"     => nil
 "--S 4 of 4"       => t
 "--S 40 of 40"     => t
 "--S 1 of a"       => nil
```

This is used as a final end check to make sure that all of the tests actually ran rather than having the regression test exit early and quietly. This will be false on all but the last test and will be false if the "–S" line does not contain the optional count marker. It is not required but is highly recommended.

— **defun lastcount 0** —

```
(defun lastcount (oneline)
 (let ((n :done) (m :done) next somemore isof)
 (when (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--S"))
  (setq somemore (string-trim " " (subseq oneline 3)))
  (when somemore
   (multiple-value-setq (n next) (read-from-string somemore nil :done))
   (when (integerp n)
    (setq somemore (string-trim " " (subseq somemore next)))
    (multiple-value-setq (isof next) (read-from-string somemore nil :done))
    (when (string= isof "OF")
     (setq somemore (string-trim " " (subseq somemore next)))
     (multiple-value-setq (m next) (read-from-string somemore nil :done))))))
  (and (integerp m) (integerp n) (= m n))))
```

26.33)savesystem Command

26.33.1 savesystem 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.

   axiom
   (1) -> t1:=4
   (1) -> )savesystem foo

and Axiom exits. Then do

   ./foo
   (1) -> t1
   4
```

———-

26.33.2 defvar *ThisIsARunningSystem*

When a user does

```
)savesystem foo
```

we set this variable to true. This is tested in the restart function, which is called when the system starts, to prevent losing user information.

— **initvars** —

```
(defvar *ThisIsARunningSystem* nil "Are we restarting a running system?")
```

————-

26.33.3 defun The)savesystem command

[helpSpad2Cmd p782]
[spad-save p1051]

— **defun savesystem** —

```
(defun |savesystem| (arg)
  (if (or (not (eql (|#| arg) 1)) (null (symbolp (car arg))))
    (|helpSpad2Cmd| '(|savesystem|))
    (progn
     (setq *ThisIsARunningSystem* t)
     (spad-save (symbol-name (car arg))))))
```

————-

26.34)set Command

26.34.1 set 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
```

[16]

26.34.2 Overview

This section contains tree of information used to initialize the **)set** command in the interpreter. The current list is:

```
Variable     Description                              Current Value
-------------------------------------------------------------------
compile    Library compiler options                     ...
breakmode  execute break processing on error            break
expose     control interpreter constructor exposure     ...
functions  some interpreter function options            ...
fortran    view and set options for FORTRAN output      ...
kernel     library functions built into the kernel for
           efficiency                                   ...
hyperdoc   options in using HyperDoc                    ...
help       view and set some help options               ...
history    save workspace values in a history file      on
messages   show messages for various system features    ...
naglink    options for NAGLink                          ...
output     view and set some output options             ...
quit       protected or unprotected quit                unprotected
streams    set some options for working with streams    ...
system     set some system development variables        ...
userlevel  operation access level of system user        development

Variables with current values of ... have further sub-options.
For example, issue )set  system to see what the options are
for system. For more information, issue )help set .
```

26.34.3 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 26.34.9. [sayMSG p40]
[literals p**??**]
[translateYesNo2TrueFalse p861]
[tree p**??**]
[initializeSetVariables p856]

[16] "quit" (26.30.2 p 836)

— 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))))))
```

———-

26.34.4 Reset the workspace variables

[copy p??]
[initializeSetVariables p856]
[/editfile p755]
[/sourcefiles p??]
[/pretty p??]
[$spaceList p64]
[$countList p60]
[$timerList p65]
[$sourceFiles p??]
[$existingFiles p??]
[$functionTable p744]
[$boot p734]
[$compileMapFlag p??]
[$echoLineStack p??]
[$operationNameList p??]
[$slamFlag p??]
[$CommandSynonymAlist p727]
[$InitialCommandSynonymAlist p725]
[$UserAbbreviationsAlist p??]
[$msgAlist p38]
[$msgDatabase p??]
[$msgDatabaseName p177]
[$dependeeClosureAlist p??]
[$IOindex p34]
[$coerceIntByMapCounter p??]
[$e p285]
[$env p284]
[$setOptions p??]

— **defun resetWorkspaceVariables** —

```
(defun |resetWorkspaceVariables| ()
 "Reset the workspace variables"
 (declare (special |$countList| /editfile /sourcefiles |$sourceFiles| /pretty
    |$spaceList| |$timerList| |$existingFiles| |$functionTable| $boot
    |$compileMapFlag| |$echoLineStack| |$operationNameList| |$slamFlag|
    |$CommandSynonymAlist| |$InitialCommandSynonymAlist|
    |$UserAbbreviationsAlist| |$msgAlist| |$msgDatabase| |$msgDatabaseName|
    |$dependeeClosureAlist| |$IOindex| |$coerceIntByMapCounter| |$e| |$env|
    |$setOptions|))
  (setq |$countList| nil)
  (setq /editfile nil)
  (setq /sourcefiles nil)
  (setq |$sourceFiles| nil)
  (setq /pretty nil)
  (setq |$spaceList| nil)
  (setq |$timerList| nil)
  (setq |$existingFiles| (make-hash-table :test #'equal))
  (setq |$functionTable| nil)
  (setq $boot nil)
  (setq |$compileMapFlag| nil)
  (setq |$echoLineStack| nil)
  (setq |$operationNameList| nil)
  (setq |$slamFlag| nil)
  (setq |$CommandSynonymAlist| (copy |$InitialCommandSynonymAlist|))
  (setq |$UserAbbreviationsAlist| nil)
  (setq |$msgAlist| nil)
  (setq |$msgDatabase| nil)
  (setq |$msgDatabaseName| nil)
  (setq |$dependeeClosureAlist| nil)
  (setq |$IOindex| 1)
  (setq |$coerceIntByMapCounter| 0)
  (setq |$e| (cons (cons nil nil) nil))
  (setq |$env| (cons (cons nil nil) nil))
  (|initializeSetVariables| |$setOptions|))
```

———-

26.34.5 Display the set option information

[displaySetVariableSettings p860]
[concat p1107]
[object2String p**??**]
[specialChar p1043]
[sayBrightly p**??**]
[bright p**??**]
[sayMSG p40]
[boot-equal p**??**]
[sayMessage p**??**]
[eval p**??**]

[literals p??]
[translateTrueFalse2YesNo p861]
[$linelength p936]

— 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
     (format t "~v,,,'-:@<~a~>~%" (- $linelength 2)
      (concat " The " (|object2String| arg) " Option "))
    (|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"
          |%l| ,@(|bright| (elt (sixth setdata) 1)) "inclusive."
          " The current setting is" ,@(|bright| (|eval| (fifth setdata))))))
     (string
      (|sayMessage|
        `(" The" ,@(|bright| arg) "option"
          " is followed by a string enclosed in double quote marks."
          '|%l| " The current setting is"
          ,@(|bright| (list '|"| (|eval| (fifth setdata)) '|"|)))))
     (literals
       (|sayMessage|
        `(" The" ,@(|bright| arg) "option"
              " may be followed by any one of the following:"))
       (setq current
        (|translateTrueFalse2YesNo| (|eval| (fifth setdata))))
       (dolist (name (sixth setdata))
         (if (boot-equal name current)
           (|sayBrightly| `( " ->" ,@(|bright| (|object2String| name))))
           (|sayBrightly| (list "    " (|object2String| name)))))
       (|sayMessage| " The current setting is indicated."))))))
```

———-

26.34.6 Display the set variable settings

[concat p1107]
[object2String p??]
[sayBrightly p??]
[say p??]
[fillerSpaces p279]
[specialChar p1043]
[concat p1107]
[satisfiesUserLevel p703]
[poundsign p??]
[eval p??]
[bright p??]
[literals p??]
[translateTrueFalse2YesNo p861]
[tree p??]
[$linelength p936]

— **defun displaySetVariableSettings** —

```
(defun |displaySetVariableSettings| (settree label)
 "Display the set variable settings"
 (let (setoption opt subtree subname)
 (declare (special $linelength))
  (if (eq label '||)
    (setq label ")set")
    (setq label (concat "  " (|object2String| label) " ")))
  (format t "~v:@<~a~>~%" (- $linelength 2)
    (concat " Current Values of" label " Variables "))
  (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| (- 13 (|#| setoption)) " ")
       (second setdata)))
     (setq setoption
      (concat setoption
       (|fillerSpaces| (- 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 opt)))
      (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") ".")))))
```

———-

26.34.7 Translate options values to t or nil

[member p1108]

— **defun translateYesNo2TrueFalse** —

```
(defun |translateYesNo2TrueFalse| (x)
 "Translate options values to t or nil"
 (cond
  ((|member| x '(|yes| |on|)) t)
  ((|member| x '(|no| |off|)) nil)
  (t x)))
```

———-

26.34.8 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)))
```

———-

26.34.9 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**. 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

26.35 set 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.
```

26.35.1 defvar $BreakMode

— **initvars** —

```
(defvar |$BreakMode| '|nobreak| "execute break processing on error")
```

————-

— **breakmode** —

```
  (|breakmode|
   "execute break processing on error"
   |interpreter|
   LITERALS
   |$BreakMode|
   (|nobreak| |break| |query| |resume| |fastlinks| |quit|)
   |nobreak|)         ; needed to avoid possible startup looping
```

————-

26.36 set debug

```
                Current Values of debug Variables

Variable     Description                            Current Value
-----------------------------------------------------------------
lambdatype   Show type information for #1 syntax     off
dalymode     Interpret leading open paren as lisp    off
```

— **debug** —

```
   (|debug|
    "debug options"
    |interpreter|
    TREE
    |novar|
    (
\getchunk{debuglambdatype}
\getchunk{debugdalymode}
    ))
```

———-

26.36.1 set debug lambdatype

```
---------------------- The lambdatype Option ----------------------

 Description: Show type information for #1 syntax
```

26.36.2 defvar $lambdatype

— **initvars** —

```
(defvar $lambdatype nil "show type information for #1 syntax")
```

———-

— **debuglambdatype** —

```
      (|lambdatype|
       "show type information for #1 syntax"
       |interpreter|
       LITERALS
       $lambdatype
       (|on| |off|)
       |off|)
```

———-

26.37 set compiler

```
                Current Values of  compiler  Variables

Variable     Description                              Current Value
-----------------------------------------------------------------
output       library in which to place compiled code
input        controls libraries from which to load compiled code
```

— **compile** —

```
   (|compiler|
    "Library compiler options"
    |interpreter|
    TREE
    |novar|
    (
\getchunk{compileoutput}
\getchunk{compileinput}
    ))
```

————-

26.37.1 set compiler output

```
---------------------- The output Option ----------------------

 Description: library in which to place compiled code
```

— **compileoutput** —

```
      (|output|
       "library in which to place compiled code "
       |interpreter|
       FUNCTION
       |setOutputLibrary|
       NIL
       |htSetOutputLibrary|
       )
```

————-

26.37.2 The set output command handler

[poundsign p??]
[describeOutputLibraryArgs p866]
[filep p??]
[openOutputLibrary p866]
[$outputLibraryName p??]

— **defun setOutputLibrary** —

```
(defun |setOutputLibrary| (arg)
 "The set output command handler"
 (let (fn)
 (declare (special |$outputLibraryName|))
   (cond
    ((eq arg '|%initialize%|) (setq |$outputLibraryName| nil))
    ((eq arg '|%display%|) (or |$outputLibraryName| "user.lib"))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?) (/= (|#| arg) 1))
     (|describeOutputLibraryArgs|))
    (t
     (when (probe-file (setq fn (princ-to-string (car arg))))
       (setq fn (truename fn)))
     (|openOutputLibrary| (setq |$outputLibraryName| fn))))))
```

———-

26.37.3 Describe the set output library arguments

[sayBrightly p??]

— **defun describeOutputLibraryArgs** —

```
(defun |describeOutputLibraryArgs| ()
 "Describe the set output library arguments"
 (|sayBrightly| (list
  " )set compile output library 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 user.lib in the current directory.")))
```

———-

26.37.4 defvar output-library

— **initvars** —

```
(defvar output-library nil)
```

———-

26.37.5 Open the output library

The input-libraries and output-library are now truename based.

[dropInputLibrary p869]
[output-library p866]
[input-libraries p869]

— **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))
```

———-

26.37.6 set compiler input

```
---------------------- The input Option -----------------------

 Description: controls libraries from which to load compiled code

 )set compile input add library is used to tell AXIOM to add
  library to the front of the path which determines where
  compiled code is loaded from.
 )set compile input drop library is used to tell AXIOM to remove
  library from this path.
```

— **compileinput** —

```
(|input|
 "controls libraries from which to load compiled code"
 |interpreter|
 FUNCTION
 |setInputLibrary|
 NIL
 |htSetInputLibrary|)
```

———-

26.37.7 The set input library command handler

The input-libraries is now maintained as a list of truenames.

[describeInputLibraryArgs p868]
[qcar p??]
[qcdr p??]
[selectOptionLC p728]
[addInputLibrary p868]
[dropInputLibrary p869]
[setInputLibrary p867]
[input-libraries p869]

— **defun setInputLibrary** —

```
(defun |setInputLibrary| (arg)
 "The set input library command handler"
 (declare (special input-libraries))
 (let (tmp1 filename act)
```

```
  (cond
   ((eq arg '|%initialize%|) t)
   ((eq arg '|%display%|) (mapcar #'namestring input-libraries))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeInputLibraryArgs|))
   ((and (consp arg)
         (progn
          (setq act (qcar arg))
          (setq tmp1 (qcdr arg))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq filename (qcar tmp1)) t)))
         (setq act (|selectOptionLC| act '(|add| |drop|) nil)))
    (cond
     ((eq act '|add|)
      (|addInputLibrary| (truename (princ-to-string filename))))
     ((eq act '|drop|)
      (|dropInputLibrary| (truename (princ-to-string filename))))))
   (t (|setInputLibrary| nil))))
```

———-

26.37.8 Describe the set input library arguments

[sayBrightly p??]

— **defun describeInputLibraryArgs** —

```
(defun |describeInputLibraryArgs| ()
 "Describe the set input library arguments"
 (|sayBrightly| (list
  " )set compile input add library "
  "is used to tell AXIOM to add library to"
  '|%l| " the front of the path used to find compile code."
  '|%l|
  " )set compile input drop library is used to tell AXIOM to remove library"
  '|%l| " from this path.")))
```

———-

26.37.9 Add the input library to the list

The input-libraries variable is now maintained as a list of truenames. [dropInputLibrary p869]
[input-libraries p869]

— **defun addInputLibrary** —

```
(defun |addInputLibrary| (lib)
 "Add the input library to the list"
 (declare (special input-libraries))
  (|dropInputLibrary| lib)
```

```
  (push (truename lib) input-libraries))
```

———-

26.37.10 defvar input-libraries

— **initvars** —

```
(defvar input-libraries nil)
```

———-

26.37.11 Drop an input library from the list

[input-libraries p869]

— **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)))
```

———-

26.38 set 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
```

26.38.1 defvar dalymode

— **initvars** —

```
(defvar $dalymode nil "Interpret leading open paren as lisp")
```

———-

— **debugdalymode** —

```
(|dalymode|
 "Interpret leading open paren as lisp"
 |interpreter|
 LITERALS
 $dalymode
 (|on| |off|)
 |off|)
```

———-

26.39 set expose

```
---------------------- The expose Option ----------------------

 Description: control interpreter constructor exposure

   The following groups are explicitly exposed in the current
   frame (called initial ):
                                    basic
                                categories
                                  naglink
                                   anna

   The following constructors are explicitly exposed in the
   current frame:
               there are no explicitly exposed constructors

   The following constructors are explicitly hidden in the
   current frame:
                there are no explicitly hidden constructors

   When )set expose is followed by no arguments, the information
   you now see is displayed. When followed by the initialize
   argument, the exposure group data in the file interp.exposed
   is read and is then available. The arguments add and drop are
   used to add or drop exposure groups or explicit constructors
   from the local frame exposure data. Issue
                  )set expose add    or    )set expose drop
   for more information.
```

— **expose** —

```
(|expose|
 "control interpreter constructor exposure"
 |interpreter|
 FUNCTION
 |setExpose|
 NIL
```

```
|htSetExpose|)
```

———-

26.39.1 functions

```
            Current Values of  functions  Variables

Variable     Description                            Current Value
-------------------------------------------------------------------
cache        number of function results to cache        0
compile      compile, don't just define function bodies off
recurrence   specially compile recurrence relations     on
```

— functions —

```
  (|functions|
   "some interpreter function options"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{functionscache}
\getchunk{functionscompile}
\getchunk{functionsrecurrence}
  ))
```

———-

26.39.2 functions cache

```
---------------------- The cache Option -----------------------

 Description: number of function results to cache

 )set functions cache is used to tell AXIOM how many  values
  computed by interpreter functions should be saved.  This can
  save quite a bit of time in recursive functions, though one
  must consider that the cached values will take up (perhaps
  valuable) room in the workspace.

 The value given  after cache must either be the word all or
 a positive integer. This may be followed by any number of
 function names whose cache sizes you wish to so set.  If no
 functions are given, the default cache  size is set.

 Examples:   )set fun cache all
             )set fun cache 10 f g Legendre

 In general, functions will cache no returned values.
```

— **functionscache** —

```
(|cache|
 "number of function results to cache"
 |interpreter|
 FUNCTION
 |setFunctionsCache|
 NIL
 |htSetCache|)
```

———-

26.39.3 defvar $cacheAlist

— **initvars** —

```
(defvar |$cacheAlist| nil)
```

———-

26.39.4 The top level set functions cache handler

[object2String p??]
[describeSetFunctionsCache p874]
[sayAllCacheCounts p875]
[sayMessage p??]
[bright p??]
[terminateSystemCommand p704]
[countCache p873]
[$options p63]
[$cacheCount p??]
[$cacheAlist p872]

— **defun setFunctionsCache** —

```
(defun |setFunctionsCache| (arg)
 "The top level set functions cache handler"
 (let (|$options| n)
  (declare (special |$options| |$cacheCount| |$cacheAlist|))
  (cond
    ((eq arg '|%initialize%|)
      (setq |$cacheCount| 0)
      (setq |$cacheAlist| nil))
    ((eq arg '|%display%|)
      (if (null |$cacheAlist|)
       (|object2String| |$cacheCount|)
       "..."))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (|describeSetFunctionsCache|)
      (terpri)
      (|sayAllCacheCounts|))
```

```
    (t
      (setq n (car arg))
      (cond
       ((and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
        (|sayMessage|
         `("Your value of" ,@(|bright| n) "is invalid because ..."))
        (|describeSetFunctionsCache|)
        (|terminateSystemCommand|))
       (t
        (when (cdr arg) (list (cons '|vars| (cdr arg))))
        (|countCache| n)))))))
```

———-

26.39.5 Display a particular cache count

[qcdr p??]
[qcar p??]
[identp p1107]
[sayKeyedMsg p39]
[insertAlist p874]
[internl p??]
[sayCacheCount p875]
[optionError p702]
[$options p63]
[$cacheAlist p872]
[$cacheCount p??]

— **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 |$options|))
           (and (consp tmp1)
                (eq (qcar tmp1) '|vars|)
                (progn (setq l (qcdr tmp1)) t))))
     (dolist (x l)
       (if (null (identp x))
         (|sayKeyedMsg| "%1 is not a valid function name." (list x))
         (progn
          (setq |$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))))))
```

———-

26.39.6 defun insertAlist

[rplac p**??**]
[?order p**??**]

— **defun insertAlist** —

```
(defun |insertAlist| (a b z)
 (labels (
  (fn (a b z)
   (cond
    ((null (cdr z)) (rplac (cdr z) (list (cons a b))))
    ((equal a (elt (elt z 1) 0)) (rplac (cdr (elt z 1)) b))
    ((?order (elt (elt z 1) 0) a) (rplac (cdr z) (cons (cons a b) (cdr z))))
    (t (fn a b (cdr z))))))
  (cond
    ((null z) (list (cons a b)))
    ((equal a (elt (elt z 0) 0)) (rplac (cdar z) b) z)
    ((?order (elt (elt z 0) 0) a) (cons (cons a b) z))
    (t (fn a b z) z))))
```

———-

26.39.7 Describe the set functions cache

[sayBrightly p**??**]

— **defun describeSetFunctionsCache** —

```
(defun |describeSetFunctionsCache| ()
 "Describe the set functions cache"
 (|sayBrightly| (list
  " )set functions cache "
  "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 "
   "cache must either be the word all 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")))
```

———-

26.39.8 Display all cache counts

[sayCacheCount p875]
[$cacheCount p??]
[$cacheAlist p872]

— defun sayAllCacheCounts —

```
(defun |sayAllCacheCounts| ()
 "Display all cache counts"
 (let (x n)
 (declare (special |$cacheCount| |$cacheAlist|))
   (|sayCacheCount| nil |$cacheCount|)
   (when |$cacheAlist|
     (do ((t0 |$cacheAlist| (cdr t0)) (t1 nil))
         ((or (atom t0)
              (progn (setq t1 (car t0)) nil)
              (progn
               (progn (setq x (car t1)) (setq n (cdr t1)) t1)
               nil))
             nil)
       (when (not (equal n |$cacheCount|)) (|sayCacheCount| x n))))))
```

———-

26.39.9 Describe the cache counts

[bright p??]
[linearFormatName p??]
[sayBrightly p??]

— 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))))))
```

———-

26.39.10 functions compile

```
--------------------- The compile Option ----------------------

 Description: compile, don't just define function bodies

 The compile option may be followed by any one of the following:

 -> on
    off

 The current setting is indicated.

\defdollar{compileDontDefineFunctions}
\begin{chunk}{initvars}
(defvar |$compileDontDefineFunctions| t
   "compile, don't just define function bodies")

\end{chunk}
\begin{chunk}{functionscompile}
     (|compile|
      "compile, don't just define function bodies"
      |interpreter|
      LITERALS
      |$compileDontDefineFunctions|
      (|on| |off|)
      |on|)
\end{chunk}
\subsection{functions recurrence}
\begin{verbatim}
-------------------- 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.
```

26.39.11 defvar $compileRecurrence

— **initvars** —

```
(defvar |$compileRecurrence| t "specially compile recurrence relations")
```

———-

— **functionsrecurrence** —

```
(|recurrence|
 "specially compile recurrence relations"
 |interpreter|
 LITERALS
 |$compileRecurrence|
 (|on| |off|)
 |on|)
```

———-

26.40 set fortran

```
              Current Values of  fortran  Variables

Variable     Description                              Current Value
-----------------------------------------------------------------
ints2floats  where sensible, coerce integers to reals   on
fortindent   the number of characters indented          6
fortlength   the number of characters on a line         72
typedecs     print type and dimension lines             on
defaulttype  default generic type for FORTRAN object    REAL
precision    precision of generated FORTRAN objects     double
intrinsic    whether to use INTRINSIC FORTRAN functions off
explength    character limit for FORTRAN expressions    1320
segment      split long FORTRAN expressions             on
optlevel     FORTRAN optimisation level                 0
startindex   starting index for FORTRAN arrays          1
calling      options for external FORTRAN calls         ...

Variables with current values of ... have further sub-options.
For example, issue )set calling to see what the options are for
calling.
For more information, issue )help set .
```

— **fortran** —

```
  (|fortran|
   "view and set options for FORTRAN output"
```

```
   |interpreter|
   TREE
   |novar|
    (
\getchunk{fortranints2floats}
\getchunk{fortranfortindent}
\getchunk{fortranfortlength}
\getchunk{fortrantypedecs}
\getchunk{fortrandefaulttype}
\getchunk{fortranprecision}
\getchunk{fortranintrinsic}
\getchunk{fortranexplength}
\getchunk{fortransegment}
\getchunk{fortranoptlevel}
\getchunk{fortranstartindex}
\getchunk{fortrancalling}
  ))
```

———-

26.40.1 set 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.
```

26.40.2 defvar $fortInts2Floats

— **initvars** —

```
(defvar |$fortInts2Floats| t "where sensible, coerce integers to reals")
```

———-

— **fortranints2floats** —

```
     (|ints2floats|
      "where sensible, coerce integers to reals"
      |interpreter|
      LITERALS
      |$fortInts2Floats|
      (|on| |off|)
```

```
      |on|)
```

———-

26.40.3 set 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
```

26.40.4 defvar $fortIndent

— **initvars** —

```
(defvar |$fortIndent| 6 "the number of characters indented")
```

———-

— **fortranfortindent** —

```
     (|fortindent|
      "the number of characters indented"
      |interpreter|
      INTEGER
      |$fortIndent|
      (0 NIL)
      6)
```

———-

26.40.5 set fortlength

```
-------------------- The fortlength Option --------------------

 Description: the number of characters on a line

 The fortlength option may be followed by an integer in the range
 1 to inclusive. The current setting is 72
```

26.40.6 defvar $fortLength

— **initvars** —

```
(defvar |$fortLength| 72 "the number of characters on a line")
```

———-

— **fortranfortlength** —

```
(|fortlength|
 "the number of characters on a line"
 |interpreter|
 INTEGER
 |$fortLength|
 (1 NIL)
 72)
```

———-

26.40.7 set typedecs

```
--------------------- The typedecs Option ---------------------

 Description: print type and dimension lines

 The typedecs option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

26.40.8 defvar $printFortranDecs

— **initvars** —

```
(defvar |$printFortranDecs| t "print type and dimension lines")
```

———-

— **fortrantypedecs** —

```
(|typedecs|
 "print type and dimension lines"
 |interpreter|
 LITERALS
 |$printFortranDecs|
 (|on| |off|)
 |on|)
```

———-

26.40.9 set 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.
```

26.40.10 defvar $defaultFortranType

— **initvars** —

```
(defvar |$defaultFortranType| 'real "default generic type for FORTRAN object")
```

————-

— **fortrandefaulttype** —

```
    (|defaulttype|
     "default generic type for FORTRAN object"
     |interpreter|
     LITERALS
     |$defaultFortranType|
     (REAL INTEGER COMPLEX LOGICAL CHARACTER)
     REAL)
```

————-

26.40.11 set 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.
```

26.40.12 defvar $fortranPrecision

— **initvars** —

```
(defvar |$fortranPrecision| '|double| "precision of generated FORTRAN objects")
```

————-

— **fortranprecision** —

```
     (|precision|
     "precision of generated FORTRAN objects"
      |interpreter|
      LITERALS
      |$fortranPrecision|
      (|single| |double|)
      |double|)
```

————-

26.40.13 set 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.
```

26.40.14 defvar $useIntrinsicFunctions

— **initvars** —

```
(defvar |$useIntrinsicFunctions| nil
   "whether to use INTRINSIC FORTRAN functions")
```

————-

— **fortranintrinsic** —

```
     (|intrinsic|
      "whether to use INTRINSIC FORTRAN functions"
      |interpreter|
      LITERALS
      |$useIntrinsicFunctions|
```

```
(|on| |off|)
|off|)
```

———-

26.40.15 set 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
```

26.40.16 defvar $maximumFortranExpressionLength

— **initvars** —

```
(defvar |$maximumFortranExpressionLength| 1320
  "character limit for FORTRAN expressions")
```

———-

— **fortranexplength** —

```
(|explength|
 "character limit for FORTRAN expressions"
 |interpreter|
 INTEGER
 |$maximumFortranExpressionLength|
 (0 NIL)
 1320)
```

———-

26.40.17 set 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.
```

26.40.18 defvar $fortranSegment

— **initvars** —

```
(defvar |$fortranSegment| t "split long FORTRAN expressions")
```

————-

— **fortransegment** —

```
(|segment|
 "split long FORTRAN expressions"
 |interpreter|
 LITERALS
 |$fortranSegment|
 (|on| |off|)
 |on|)
```

————-

26.40.19 set 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
```

26.40.20 defvar $fortranOptimizationLevel

— **initvars** —

```
(defvar |$fortranOptimizationLevel| 0 "FORTRAN optimisation level")
```

————-

— **fortranoptlevel** —

```
(|optlevel|
 "FORTRAN optimisation level"
 |interpreter|
 INTEGER
 |$fortranOptimizationLevel|
 (0 2)
 0)
```

————-

26.40.21 set startindex

```
-------------------- The startindex Option --------------------

 Description: starting index for FORTRAN arrays

 The startindex option may be followed by an integer in the range
 0 to 1 inclusive. The current setting is 1
```

26.40.22 defvar $fortranArrayStartingIndex

— **initvars** —

```
(defvar |$fortranArrayStartingIndex| 1 "starting index for FORTRAN arrays")
```

————-

— **fortranstartindex** —

```
      (|startindex|
       "starting index for FORTRAN arrays"
       |interpreter|
       INTEGER
       |$fortranArrayStartingIndex|
       (0 1)
       1)
```

————-

26.40.23 set calling

```
              Current Values of  calling  Variables

Variable     Description                            Current Value
-----------------------------------------------------------------
tempfile     set location of temporary data files       /tmp/
directory    set location of generated FORTRAN files    ./
linker       linker arguments (e.g. libraries to search) -lxlf
```

— **fortrancalling** —

```
      (|calling|
      "options for external FORTRAN calls"
      |interpreter|
      TREE
      |novar|
      (
\getchunk{callingtempfile}
\getchunk{callingdirectory}
\getchunk{callinglinker}
```

```
      )
     )
```

———-

set tempfile

```
--------------------- The tempfile Option ---------------------

 Description: set location of temporary data files

 )set fortran calling tempfile  is used to tell AXIOM where
 to place intermediate FORTRAN data files . This must be the
 name of a valid existing directory to which you have permission
 to write (including the final slash).

 Syntax:
   )set fortran calling tempfile DIRECTORYNAME

 The current setting is /tmp/
```

26.40.24 defvar $fortranTmpDir

— **initvars** —

```
(defvar |$fortranTmpDir| "/tmp/" "set location of temporary data files ")
```

———-

— **callingtempfile** —

```
(|tempfile|
 "set location of temporary data files "
 |interpreter|
 FUNCTION
 |setFortTmpDir|
 (("enter directory name for which you have write-permission "
   DIRECTORY
   |$fortranTmpDir|
   |chkDirectory|
   "/tmp/"))
 NIL)
```

———-

26.40.25 The top level set fortran calling tempfile handler

[pname p1106]
[describeSetFortTmpDir p887]
[validateOutputDirectory p887]

[sayBrightly p??]
[bright p??]
[$fortranTmpDir p886]

— defun setFortTmpDir —

```
(defun |setFortTmpDir| (arg)
 "The top level set fortran calling tempfile handler"
 (let (mode)
   (declare (special |$fortranTmpDir|))
   (cond
    ((eq arg '|%initialize%|) (setq |$fortranTmpDir| "/tmp/"))
    ((eq arg '|%display%|)
     (if (stringp |$fortranTmpDir|)
      |$fortranTmpDir|
      (pname |$fortranTmpDir|)))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetFortTmpDir|))
    ((null (setq mode (|validateOutputDirectory| arg)))
     (|sayBrightly|
      `(" Sorry, but your argument(s)" ,@(|bright| arg)
            "is(are) not valid." |%l|))
      (|describeSetFortTmpDir|))
    (t (setq |$fortranTmpDir| mode)))))
```

———-

26.40.26 Validate the output directory

— defun validateOutputDirectory —

```
(defun |validateOutputDirectory| (x)
 "Validate the output directory"
 (let ((dirname (car x)))
  (when (and (pathname-directory (string dirname)) (null (probe-file dirname)))
   dirname)))
```

———-

26.40.27 Describe the set fortran calling tempfile

[sayBrightly p??]
[$fortranTmpDir p886]

— defun describeSetFortTmpDir —

```
(defun |describeSetFortTmpDir| ()
 "Describe the set fortran calling tempfile"
 (declare (special |$fortranTmpDir|))
 (|sayBrightly| (list
  " )set fortran calling tempfile"
```

```
  " 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 "
  |$fortranTmpDir|
  )))
```

———-

directory

```
-------------------- The directory Option ---------------------

 Description: set location of generated FORTRAN files

 )set fortran calling directory  is used to tell AXIOM where
 to place generated FORTRAN files. This must be the name
 of a valid existing directory to which you have permission
 to write (including the final slash).

 Syntax:
   )set fortran calling directory DIRECTORYNAME

 The current setting is ./
```

26.40.28 defvar $fortranDirectory

— **initvars** —

```
(defvar |$fortranDirectory| "./" "set location of generated FORTRAN files ")
```

———-

— **callingdirectory** —

```
       (|directory|
        "set location of generated FORTRAN files "
        |interpreter|
        FUNCTION
        |setFortDir|
        (("enter directory name for which you have write-permission "
          DIRECTORY
          |$fortranDirectory|
          |chkDirectory|
          "./"))
        NIL)
```

———-

26.40.29 defun setFortDir

[pname p1106]
[describeSetFortDir p889]
[validateOutputDirectory p887]
[sayBrightly p??]
[bright p??]
[$fortranDirectory p888]

— defun setFortDir —

```
(defun |setFortDir| (arg)
 (declare (special |$fortranDirectory|))
 (let (mode)
   (COND
    ((eq arg '|%initialize%|) (setq |$fortranDirectory| "./"))
    ((eq arg '|%display%|)
     (if (stringp |$fortranDirectory|)
      |$fortranDirectory|
      (pname |$fortranDirectory|)))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (|describeSetFortDir|))
    ((null (setq mode (|validateOutputDirectory| arg)))
     (|sayBrightly|
      `(" Sorry, but your argument(s)" ,@(|bright| arg)
        "is(are) not valid." |%l|))
     (|describeSetFortDir|))
    (t (setq |$fortranDirectory| mode)))))
```

———-

26.40.30 defun describeSetFortDir

[sayBrightly p??]
[$fortranDirectory p888]

— defun describeSetFortDir —

```
(defun |describeSetFortDir| ()
 (declare (special |$fortranDirectory|))
 (|sayBrightly| (list
  " )set fortran calling directory"
  " is used to tell AXIOM where"
  '|%l| " to place generated FORTRAN files. This must be the name "
  '|%l| " of a valid existing directory to which you have permission "
  '|%l| " to write (including the final slash)."
  '|%l|
  '|%l| " Syntax:"
  '|%l| "   )set fortran calling directory DIRECTORYNAME"
  '|%l|
```

```
  '|%l| " The current setting is "
  |$fortranDirectory|
  )))
```

———-

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
```

26.40.31 defvar $fortranLibraries

— **initvars** —

```
(defvar |$fortranLibraries| "-lxlf"
  "linker arguments (e.g. libraries to search)")
```

———-

— **callinglinker** —

```
        (|linker|
         "linker arguments (e.g. libraries to search) "
         |interpreter|
         FUNCTION
         |setLinkerArgs|
         (("enter linker arguments "
           STRING
           |$fortranLibraries|
           |chkDirectory|
           "-lxlf"))
         NIL
         )
```

———-

26.40.32 defun setLinkerArgs

[object2String p??]
[describeSetLinkerArgs p891]
[$fortranLibraries p890]

— **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|))))
```

———-

26.40.33 defun describeSetLinkerArgs

[sayBrightly p??]
[$fortranLibraries p890]

— **defun describeSetLinkerArgs** —

```
(defun |describeSetLinkerArgs| ()
 (declare (special |$fortranLibraries|))
 (|sayBrightly| (list
  " )set fortran calling linkerargs"
  " is used to pass arguments to the linker"
  '|%l| " when using "
  "mkFort to create functions which call Fortran code."
  '|%l| " For example, it might give a list of libraries to be searched,"
  '|%l| " and their locations."
  '|%l| " The string is passed verbatim, so must be the correct syntax for"
  '|%l| " the particular linker being used."
  '|%l|
  '|%l| " Example: )set fortran calling linker \"-lxlf\""
  '|%l|
  '|%l| " The current setting is "
  |$fortranLibraries|
  )))
```

———-

26.41 set hyperdoc

```
Current Values of  hyperdoc  Variables
```

```
Variable     Description                              Current Value
-----------------------------------------------------------------
fullscreen   use full screen for this facility          off
mathwidth    screen width for history output            120
```

— **hyperdoc** —

```
  (|hyperdoc|
   "options in using HyperDoc"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{hyperdocfullscreen}
\getchunk{hyperdocmathwidth}
   ))
```

————-

26.41.1 fullscreen

```
-------------------- The fullscreen Option --------------------

 Description: use full screen for this facility

 The fullscreen option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

26.41.2 defvar $fullScreenSysVars

— **initvars** —

```
(defvar |$fullScreenSysVars| nil "use full screen for this facility")
```

————-

— **hyperdocfullscreen** —

```
     (|fullscreen|
      "use full screen for this facility"
      |interpreter|
      LITERALS
      |$fullScreenSysVars|
      (|on| |off|)
```

```
      |off|)
```

———-

26.41.3 mathwidth

```
-------------------- The mathwidth Option ---------------------

 Description: screen width for history output

 The mathwidth option may be followed by an integer in the range
 0 to inclusive. The current setting is 120
```

26.41.4 defvar $historyDisplayWidth

— **initvars** —

```
(defvar |$historyDisplayWidth| 120 "screen width for history output")
```

———-

— **hyperdocmathwidth** —

```
    (|mathwidth|
     "screen width for history output"
     |interpreter|
     INTEGER
     |$historyDisplayWidth|
     (0 NIL)
     120)
```

———-

26.42 set help

```
              Current Values of  help  Variables

Variable     Description                           Current Value
-----------------------------------------------------------------
fullscreen   use fullscreen facility, if possible       on
```

— **help** —

```
  (|help|
   "view and set some help options"
   |interpreter|
   TREE
   |novar|
```

```
   (
\getchunk{helpfullscreen}
   ))
```

———-

26.42.1 fullscreen

```
-------------------- The fullscreen Option --------------------

 Description: use fullscreen facility, if possible

 The fullscreen option may be followed by any one of the
 following:

 -> on
   off

 The current setting is indicated.
```

26.42.2 defvar $useFullScreenHelp

— **initvars** —

```
(defvar |$useFullScreenHelp| t "use fullscreen facility, if possible")
```

———-

— **helpfullscreen** —

```
    (|fullscreen|
     "use fullscreen facility, if possible"
     |interpreter|
     LITERALS
     |$useFullScreenHelp|
     (|on| |off|)
     |on|)
```

———-

26.43 set 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.
```

26.43.1 defvar $HiFiAccess

— **initvars** —

```
(defvar |$HiFiAccess| t "save workspace values in a history file")
```

———-

— **history** —

```
  (|history|
   "save workspace values in a history file"
   |interpreter|
   LITERALS
   |$HiFiAccess|
   (|on| |off|)
   |on|)
```

———-

26.44 set messages

```
            Current Values of  messages  Variables

Variable     Description                              Current Value
-----------------------------------------------------------------
autoload     print file auto-load messages              off
bottomup     display bottom up modemap selection        off
coercion     display datatype coercion messages         off
dropmap      display old map defn when replaced         off
expose       warning for unexposed functions            off
file         print msgs also to SPADMSG LISTING         off
frame        display messages about frames              off
highlighting use highlighting in system messages        off
instant      present instantiation summary              off
insteach     present instantiation info                 off
interponly   say when function code is interpreted      on
number       display message number with message        off
prompt       set type of input prompt to display        step
selection    display function selection msgs            off
set          show )set setting after assignment         off
startup      display messages on start-up               off
summary      print statistics after computation         off
testing      print system testing header                off
time         print timings after computation            off
```

```
type         print type after computation               on
void         print Void value when it occurs            off
any          print the internal type of objects of domain Any on
naglink      show NAGLink messages                      on
```

— **messages** —

```
  (|messages|
   "show messages for various system features"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{messagesany}
\getchunk{messagesautoload}
\getchunk{messagesbottomup}
\getchunk{messagescoercion}
\getchunk{messagesdropmap}
\getchunk{messagesexpose}
\getchunk{messagesfile}
\getchunk{messagesframe}
\getchunk{messageshighlighting}
\getchunk{messagesinstant}
\getchunk{messagesinsteach}
\getchunk{messagesinterponly}
\getchunk{messagesnaglink}
\getchunk{messagesnumber}
\getchunk{messagesprompt}
\getchunk{messagesselection}
\getchunk{messagesset}
\getchunk{messagesstartup}
\getchunk{messagessummary}
\getchunk{messagestesting}
\getchunk{messagestime}
\getchunk{messagestype}
\getchunk{messagesvoid}
   ))
```

———-

26.44.1 set message 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.
```

26.44.2 defvar $printAnyIfTrue

— **initvars** —

```
(defvar |$printAnyIfTrue| t
   "print the internal type of objects of domain Any")
```

————-

— **messagesany** —

```
     (|any|
      "print the internal type of objects of domain Any"
      |interpreter|
      LITERALS
      |$printAnyIfTrue|
      (|on| |off|)
      |on|)
```

————-

26.44.3 set message autoload

```
---------------------- The autoload Option ----------------------

 Description: print file auto-load messages
```

26.44.4 defvar $printLoadMsgs

— **initvars** —

```
(defvar |$printLoadMsgs| nil "print file auto-load messages")
```

————-

— **messagesautoload** —

```
     (|autoload|
      "print file auto-load messages"
      |interpreter|
      LITERALS
      |$printLoadMsgs|
      (|on| |off|)
      |on|)
```

————-

26.44.5 set message bottomup

```
--------------------- The bottomup Option ---------------------

 Description: display bottom up modemap selection

 The bottomup option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

26.44.6 defvar $reportBottomUpFlag

— **initvars** —

```
(defvar |$reportBottomUpFlag| nil "display bottom up modemap selection")
```

———-

— **messagesbottomup** —

```
     (|bottomup|
      "display bottom up modemap selection"
      |development|
      LITERALS
      |$reportBottomUpFlag|
      (|on| |off|)
      |off|)
```

———-

26.44.7 set message 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.
```

26.44.8 defvar $reportCoerceIfTrue

— **initvars** —

```
(defvar |$reportCoerceIfTrue| nil "display datatype coercion messages")
```

————-

— **messagescoercion** —

```
    (|coercion|
     "display datatype coercion messages"
     |development|
     LITERALS
     |$reportCoerceIfTrue|
     (|on| |off|)
     |off|)
```

————-

26.44.9 set message dropmap

```
---------------------- The dropmap Option -----------------------

 Description: display old map defn when replaced

 The dropmap option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

26.44.10 defvar $displayDroppedMap

— **initvars** —

```
(defvar |$displayDroppedMap| nil "display old map defn when replaced")
```

————-

— **messagesdropmap** —

```
    (|dropmap|
     "display old map defn when replaced"
     |interpreter|
     LITERALS
     |$displayDroppedMap|
```

```
      (|on| |off|)
      |off|)
```

———-

26.44.11 set message 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.
```

26.44.12 defvar $giveExposureWarning

— **initvars** —

```
(defvar |$giveExposureWarning| nil "warning for unexposed functions")
```

———-

— **messagesexpose** —

```
     (|expose|
      "warning for unexposed functions"
      |interpreter|
      LITERALS
      |$giveExposureWarning|
      (|on| |off|)
      |off|)
```

———-

26.44.13 set message file

```
---------------------- The file Option ----------------------

 Description: print msgs also to SPADMSG LISTING

 The file option may be followed by any one of the
 following:
```

```
   on
-> off

The current setting is indicated.
```

26.44.14 defvar $printMsgsToFile

— **initvars** —

```
(defvar |$printMsgsToFile| nil "print msgs also to SPADMSG LISTING")
```

————-

— **messagesfile** —

```
    (|file|
     "print msgs also to SPADMSG LISTING"
     |development|
     LITERALS
     |$printMsgsToFile|
     (|on| |off|)
     |off|)
```

————-

26.44.15 set message frame

```
---------------------- The frame Option -----------------------

 Description: display messages about frames

 The frame option may be followed by any one of the following:

   on
-> off

 The current setting is indicated.
```

26.44.16 defvar $frameMessages

— **initvars** —

```
(defvar |$frameMessages| nil "display messages about frames")
```

————-

— **messagesframe** —

```
(|frame|
 "display messages about frames"
 |interpreter|
 LITERALS
 |$frameMessages|
 (|on| |off|)
 |off|)
```

———-

26.44.17 set message highlighting

```
------------------- The highlighting Option -------------------

 Description: use highlighting in system messages

 The highlighting option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

26.44.18 defvar $highlightAllowed

— **initvars** —

```
(defvar |$highlightAllowed| nil "use highlighting in system messages")
```

———-

— **messageshighlighting** —

```
(|highlighting|
 "use highlighting in system messages"
 |interpreter|
 LITERALS
 |$highlightAllowed|
 (|on| |off|)
 |off|)
```

———-

26.44.19 set message 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.
```

26.44.20 defvar $reportInstantiations

— **initvars** —

```
(defvar |$reportInstantiations| nil "present instantiation summary")
```

————-

— **messagesinstant** —

```
     (|instant|
      "present instantiation summary"
      |development|
      LITERALS
      |$reportInstantiations|
      (|on| |off|)
      |off|)
```

————-

26.44.21 set message 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.
```

26.44.22 defvar $reportEachInstantiation—

— **initvars** —

```
(defvar |$reportEachInstantiation| nil "present instantiation info")
```

———-

— **messagesinsteach** —

```
    (|insteach|
     "present instantiation info"
     |development|
     LITERALS
     |$reportEachInstantiation|
     (|on| |off|)
     |off|)
```

———-

26.44.23 set message 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.
```

26.44.24 defvar $reportInterpOnly

— **initvars** —

```
(defvar |$reportInterpOnly| t "say when function code is interpreted")
```

———-

— **messagesinterponly** —

```
    (|interponly|
     "say when function code is interpreted"
     |interpreter|
     LITERALS
     |$reportInterpOnly|
```

```
      (|on| |off|)
      |on|)
```

———-

26.44.25 set message 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.
```

26.44.26 defvar $nagMessages

— **initvars** —

```
(defvar |$nagMessages| t "show NAGLink messages")
```

———-

— **messagesnaglink** —

```
     (|naglink|
      "show NAGLink messages"
      |interpreter|
      LITERALS
      |$nagMessages|
      (|on| |off|)
      |on|)
```

———-

26.44.27 set message 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.
```

26.44.28 defvar $displayMsgNumber

— **initvars** —

```
(defvar |$displayMsgNumber| nil "display message number with message")
```

———-

— **messagesnumber** —

```
    (|number|
     "display message number with message"
     |interpreter|
     LITERALS
     |$displayMsgNumber|
     (|on| |off|)
     |off|)
```

———-

26.44.29 set message prompt

```
---------------------- The prompt Option ----------------------

 Description: set type of input prompt to display

 The prompt option may be followed by any one of the following:

    none
    frame
    plain
 -> step
    verbose

 The current setting is indicated.
```

26.44.30 defvar $inputPromptType

— **initvars** —

```
(defvar |$inputPromptType| '|step| "set type of input prompt to display")
```

———-

— **messagesprompt** —

```
(|prompt|
 "set type of input prompt to display"
 |interpreter|
 LITERALS
 |$inputPromptType|
 (|none| |frame| |plain| |step| |verbose|)
 |step|)
```

———-

26.44.31 set message 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|)
```

———-

26.44.32 set

```
----------------------- The set Option -----------------------

 Description: show )set setting after assignment

 The set option may be followed by any one of the following:

    on
```

```
 -> off

 The current setting is indicated.
```

26.44.33 defvar $displaySetValue

— **initvars** —

```
(defvar |$displaySetValue| nil "show )set setting after assignment")
```

———-

— **messagesset** —

```
     (|set|
      "show )set setting after assignment"
      |interpreter|
      LITERALS
      |$displaySetValue|
      (|on| |off|)
      |off|)
```

———-

26.44.34 set message startup

```
--------------------- The startup Option ----------------------

 Description: display messages on start-up

 The startup option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

26.44.35 defvar $displayStartMsgs

— **initvars** —

```
(defvar |$displayStartMsgs| t "display messages on start-up")
```

———-

— **messagesstartup** —

```
(|startup|
 "display messages on start-up"
 |interpreter|
 LITERALS
 |$displayStartMsgs|
 (|on| |off|)
 |on|)
```

———-

26.44.36 set message 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.
```

26.44.37 defvar $printStatisticsSummaryIfTrue

— **initvars** —

```
(defvar |$printStatisticsSummaryIfTrue| nil
     "print statistics after computation")
```

———-

— **messagessummary** —

```
(|summary|
 "print statistics after computation"
 |interpreter|
 LITERALS
 |$printStatisticsSummaryIfTrue|
 (|on| |off|)
 |off|)
```

———-

26.44.38 set message 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.
```

26.44.39 defvar $testingSystem

— **initvars** —

```
(defvar |$testingSystem| nil "print system testing header")
```

————-

— **messagestesting** —

```
     (|testing|
      "print system testing header"
      |development|
      LITERALS
      |$testingSystem|
      (|on| |off|)
      |off|)
```

————-

26.44.40 set message 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.
```

26.44.41 defvar $printTimeIfTrue

— **initvars** —

```
(defvar |$printTimeIfTrue| nil "print timings after computation")
```

————-

— **messagestime** —

```
    (|time|
     "print timings after computation"
     |interpreter|
     LITERALS
     |$printTimeIfTrue|
     (|on| |off| |long|)
     |off|)
```

————-

26.44.42 set message 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.
```

26.44.43 defvar $printTypeIfTrue

— **initvars** —

```
(defvar |$printTypeIfTrue| t "print type after computation")
```

————-

— **messagestype** —

```
    (|type|
     "print type after computation"
     |interpreter|
     LITERALS
     |$printTypeIfTrue|
     (|on| |off|)
```

```
      |on|)
```

———-

26.44.44 set message 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.
```

26.44.45 defvar $printVoidIfTrue

— **initvars** —

```
(defvar |$printVoidIfTrue| nil "print Void value when it occurs")
```

———-

— **messagesvoid** —

```
     (|void|
      "print Void value when it occurs"
      |interpreter|
      LITERALS
      |$printVoidIfTrue|
      (|on| |off|)
      |off|)
```

———-

26.45 set naglink

```
              Current Values of  naglink  Variables

Variable     Description                            Current Value
-----------------------------------------------------------------
host         internet address of host for NAGLink       localhost
persistence  number of (fortran) functions to remember  1
messages     show NAGLink messages                      on
double       enforce DOUBLE PRECISION ASPs              on
```

— **naglink** —

```
  (|naglink|
   "options for NAGLink"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{naglinkhost}
\getchunk{naglinkpersistence}
\getchunk{naglinkmessages}
\getchunk{naglinkdouble}
   ))
```

————-

26.45.1 set naglink 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
```

26.45.2 defvar $nagHost

— **initvars** —

```
(defvar |$nagHost| "localhost" "internet address of host for NAGLink")
```

————-

— **naglinkhost** —

```
     (|host|
      "internet address of host for NAGLink "
      |interpreter|
      FUNCTION
      |setNagHost|
      (("enter host name"
        DIRECTORY
        |$nagHost|
        |chkDirectory|
        "localhost"))
      NIL)
```

———-

26.45.3 defun setNagHost

[object2String p??]
[describeSetNagHost p914]
[$nagHost p913]

— 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)))))
```

———-

26.45.4 defun describeSetNagHost

[sayBrightly p??]
[$nagHost p913]

— defun describeSetNagHost —

```
(defun |describeSetNagHost| ()
 (declare (special |$nagHost|))
 (|sayBrightly| (list
  " )set naglink host "
  "is used to tell  AXIOM which  host to contact for"
  '|%l| " a NAGLink request. An Internet address should be supplied. The host"
  '|%l| " specified must be running the NAGLink daemon."
  '|%l|
  '|%l| " The current setting is "
  |$nagHost|
  )))
```

———-

26.45.5 set naglink 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
```

26.45.6 defvar $fortPersistence

— **initvars** —

```
(defvar |$fortPersistence| 1 "number of (fortran) functions to remember")
```

————-

— **naglinkpersistence** —

```
     (|persistence|
      "number of (fortran) functions to remember "
      |interpreter|
      FUNCTION
      |setFortPers|
      (("Requested remote storage (for asps):"
       INTEGER
       |$fortPersistence|
       (0 NIL)
       10))
      NIL)
```

————-

26.45.7 defun setFortPers

[describeFortPersistence p916]
[sayMessage p**??**]
[bright p**??**]
[terminateSystemCommand p704]
[$fortPersistence p915]

— **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))))))))
```

———-

26.45.8 defun describeFortPersistence

[sayBrightly p??]
[$fortPersistence p915]

— **defun describeFortPersistence** —

```
(defun |describeFortPersistence| ()
 (declare (special |$fortPersistence|))
 (|sayBrightly| (list
 " )set naglink persistence "
 "is used to tell the "
 '|nagd|
 '| daemon how  many ASP|
 '|%l|
 " source and object files to keep around in case you reuse them. This helps"
 '|%l| " to avoid needless recompilations. The number specified should be a "
 '|%l| " non-negative integer."
 '|%l|
 '|%l| " The current setting is "
 |$fortPersistence|
 )))
```

———-

26.45.9 set naglink messages

```
--------------------- The messages Option ---------------------

 Description: show NAGLink messages

 The messages option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

TPDHERE: this is the same as)set nag mes on

— **naglinkmessages** —

```
    (|messages|
     "show NAGLink messages"
```

```
      |interpreter|
      LITERALS
      |$nagMessages|
      (|on| |off|)
       |on|)
```

———-

26.45.10 set naglink 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.
```

26.45.11 defvar $nagEnforceDouble

— **initvars** —

```
(defvar |$nagEnforceDouble| t "enforce DOUBLE PRECISION ASPs")
```

———-

— **naglinkdouble** —

```
     (|double|
      "enforce DOUBLE PRECISION ASPs"
      |interpreter|
      LITERALS
      |$nagEnforceDouble|
      (|on| |off|)
      |on|)
```

———-

26.46 set output

The result of the)**set output** command is:

```
Variable     Description                          Current Value
------------------------------------------------------------------
abbreviate   abbreviate type names                  off
```

```
algebra     display output in algebraic form         On:CONSOLE
characters  choose special output character set      plain
fortran     create output in FORTRAN format          Off:CONSOLE
fraction    how fractions are formatted              vertical
html        create output in HTML style              Off:CONSOLE
length      line length of output displays           77
mathml      create output in MathML style            Off:CONSOLE
openmath    create output in OpenMath style          Off:CONSOLE
script      display output in SCRIPT formula format  Off:CONSOLE
scripts     show subscripts,... linearly             off
showeditor  view output of )show in editor           off
tex         create output in TeX style               Off:CONSOLE
```

Since the output option has a bunch of sub-options each suboption is defined within the output structure.

— **output** —

```
  (|output|
   "view and set some output options"
   |interpreter|
   TREE
   |novar|
    (
\getchunk{outputabbreviate}
\getchunk{outputalgebra}
\getchunk{outputcharacters}
\getchunk{outputfortran}
\getchunk{outputfraction}
\getchunk{outputhtml}
\getchunk{outputlength}
\getchunk{outputmathml}
\getchunk{outputopenmath}
\getchunk{outputscript}
\getchunk{outputscripts}
\getchunk{outputshoweditor}
\getchunk{outputtex}
 ))
```

———-

26.46.1 set output 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.
```

26.46.2 defvar $abbreviateTypes

— **initvars** —

```
(defvar |$abbreviateTypes| nil "abbreviate type names")
```

———-

— **outputabbreviate** —

```
    (|abbreviate|
     "abbreviate type names"
     |interpreter|
     LITERALS
     |$abbreviateTypes|
     (|on| |off|)
     |off|)
```

———-

26.46.3 set output 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
```

26.46.4 defvar $algebraFormat

— **initvars** —

```
(defvar |$algebraFormat| t "display output in algebraic form ")
```

————-

26.46.5 defvar $algebraOutputFile

— **initvars** —

```
(defvar |$algebraOutputFile| "CONSOLE"
  "where algebra printing goes (enter {\em console} or a pathname)?")
```

————-

— **outputalgebra** —

```
    (|algebra|
     "display output in algebraic form "
     |interpreter|
     FUNCTION
     |setOutputAlgebra|
     (("display output in algebraic form "
       LITERALS
       |$algebraFormat|
       (|off| |on|)
       |on|)
      (break $algebraFormat)
      ("where algebra printing goes (enter {\em console} or a pathname)?"
       FILENAME
       |$algebraOutputFile|
       |chkOutputFileName|
       "console"))
     NIL)
```

————-

26.46.6 defvar $algebraOutputStream

— **initvars** —

```
(defvar |$algebraOutputStream| *standard-output*)
```

————-

26.46.7 defun setOutputAlgebra

[defiostream p1046]
[concat p1107]
[describeSetOutputAlgebra p923]
[qcdr p??]
[qcar p??]
[member p1108]
[upcase p1140]
[sayKeyedMsg p39]
[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p??]
[make-outstream p1045]
[object2String p??]
[$algebraOutputStream p920]
[$algebraOutputFile p920]
[$filep p??]
[$algebraFormat p920]

— defun setOutputAlgebra —

```
(defun |setOutputAlgebra| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$algebraOutputStream| |$algebraOutputFile| $filep
  |$algebraFormat|))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$algebraOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$algebraOutputFile| "CONSOLE")
     (setq |$algebraFormat| t))
    ((eq arg '|%display%|)
     (if |$algebraFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$algebraOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputAlgebra|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (|member| fn '(y n ye yes no o on of off console
                    |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
       '|ok|)
      (t (setq arg (list fn '|spout|))))
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
```

```
      (progn (setq fn (qcar arg)) t))
 (cond
  ((|member| (upcase fn) '(y n ye o of))
   (|sayKeyedMsg|
      (format nil
        "To toggle %1 printing on and off, specify %1 )set output %2 ~
         yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
    '(|algebra| |algebra|)))
  ((|member| (upcase fn) '(no off)) (setq |$algebraFormat| nil))
  ((|member| (upcase fn) '(yes on)) (setq |$algebraFormat| t))
  ((eq (upcase fn) 'console)
   (shut |$algebraOutputStream|)
   (setq |$algebraOutputStream|
    (defiostream '((mode . output) (device . console)) 255 0))
   (setq |$algebraOutputFile| "CONSOLE"))))
((or
  (and (consp arg)
       (progn
        (setq fn (qcar arg))
        (setq tmp1 (qcdr arg))
        (and (consp tmp1)
             (eq (qcdr tmp1) nil)
             (progn (setq ft (qcar tmp1)) t))))
  (and (consp arg)
       (progn (setq fn (qcar arg))
              (setq tmp1 (qcdr arg))
              (and (consp tmp1)
                   (progn (setq ft (qcar tmp1))
                          (setq tmp2 (qcdr tmp1))
                          (and (consp tmp2)
                               (eq (qcdr tmp2) nil)
                               (progn
                                (setq fm (qcar tmp2))
                                t)))))))
 (when (setq ptype (|pathnameType| fn))
  (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
  (setq ft ptype))
 (unless fm (setq fm 'a))
 (setq filename ($filep fn ft fm))
 (cond
  ((null filename)
   (|sayKeyedMsg|
    "It is not possible to open or create a file called %1 %2 %3 ."
    (list fn ft fm)))
  ((setq teststream (make-outstream filename 255 0))
   (shut |$algebraOutputStream|)
   (setq |$algebraOutputStream| teststream)
   (setq |$algebraOutputFile| (|object2String| filename))
   (|sayKeyedMsg|
    "%1 output will be written to file %2 ."
    (list "Algebra" |$algebraOutputFile|)))
  (t (|sayKeyedMsg|
      "It is not possible to open or create a file called %1 %2 %3 ."
      (list fn ft fm)))))
```

```
      (t
       (|sayKeyedMsg| "Your argument list is not valid." nil)
       (|describeSetOutputAlgebra|)))))))
```

———-

26.46.8 defun describeSetOutputAlgebra

[sayBrightly p??]
[setOutputAlgebra p921]

— defun describeSetOutputAlgebra —

```
(defun |describeSetOutputAlgebra| ()
 (|sayBrightly| (list
 " )set output algebra "
 "is used to tell AXIOM to turn algebra-style output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output algebra <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn algebra printing on (default state)"
 '|%l| "  off         turn algebra printing off"
 '|%l| "  console     send algebra output to screen (default state)"
 '|%l| "  fp<.fe>     send algebra output to file with file prefix fp"
 '|%l|
  "              and file extension .fe. If not given, .fe defaults to .spout."
 '|%l|
 '|%l|
 "If you wish to send the output to a file, you may need to issue this command"
 '|%l| "twice: once with"
 " on and once with the file name. For example, to send"
 '|%l| "algebra output to the file polymer.spout, issue the two commands"
 '|%l|
 '|%l| "  )set output algebra on"
 '|%l| "  )set output algebra polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 (|setOutputAlgebra| '|%display%|)
 )))
```

———-

26.46.9 set output characters

```
-------------------- The characters Option --------------------

 Description: choose special output character set
```

```
The characters option may be followed by any one of the
following:

   default
-> plain

The current setting is indicated.  This
option  determines the special characters used for algebraic
output. This is what the current choice of special characters
looks like:
  ulc is shown as +          urc is shown as +
  llc is shown as +          lrc is shown as +
  vbar is shown as |         hbar is shown as -
  quad is shown as ?         lbrk is shown as [
  rbrk is shown as ]         lbrc is shown as {
  rbrc is shown as }         ttee is shown as +
  btee is shown as +         rtee is shown as +
  ltee is shown as +         ctee is shown as +
  bslash is shown as \
```

— **outputcharacters** —

```
(|characters|
 "choose special output character set "
 |interpreter|
 FUNCTION
 |setOutputCharacters|
 NIL
 |htSetOutputCharacters|)
```

———-

26.46.10 defun setOutputCharacters

[sayMessage p**??**]
[bright p**??**]
[sayBrightly p**??**]
[concat p1107]
[pname p1106]
[specialChar p1043]
[sayAsManyPerLineAsPossible p**??**]
[qcdr p**??**]
[qcar p**??**]
[downcase p1140]
[setOutputCharacters p924]
[$specialCharacters p1042]
[$plainRTspecialCharacters p1041]
[$RTspecialCharacters p1042]
[$specialCharacterAlist p1043]

— defun setOutputCharacters —

```
(defun |setOutputCharacters| (arg)
 (let (current char s l fn)
 (declare (special |$specialCharacters| |$plainRTspecialCharacters|
   |$RTspecialCharacters| |$specialCharacterAlist|))
    (if (eq arg '|%initialize%|)
     (setq |$specialCharacters| |$plainRTspecialCharacters|)
     (progn
      (setq current
       (cond
        ((eq |$specialCharacters| |$RTspecialCharacters|) "default")
        ((eq |$specialCharacters| |$plainRTspecialCharacters|) "plain")
        (t "unknown")))
      (cond
       ((eq arg '|%display%|) current)
       ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
        (format t
         " The characters option may be followed by any one ~
           of the following:~%~%")
        (dolist (name '("default" "plain"))
         (if (string= (string current) name)
          (|sayBrightly| `(" ->" ,@(|bright| name)))
          (|sayBrightly|  (list "    " name))))
        (terpri)
        (format t
          " The current setting is indicated within the list.  ~
            This option determines ~% the special characters used ~
            for algebraic output.  This is what the~% current choice of ~
            special characters looks like:~%")
        (do ((t1 |$specialCharacterAlist| (CDR t1)) (t2 nil))
           ((or (atom t1)
                (progn (setq t2 (car t1)) nil)
                (progn (progn (setq char (car t2)) t2) nil)) nil)
         (setq s
          (concat "   " (pname char) " is shown as "
           (pname (|specialChar| char))))
          (setq l (cons s l)))
        (|sayAsManyPerLineAsPossible| (reverse l)))
       ((and (consp arg)
             (eq (qcdr arg) NIL)
             (progn (setq fn (qcar arg)) t)
             (setq fn (downcase fn)))
        (cond
         ((eq fn '|default|)
          (setq |$specialCharacters| |$RTspecialCharacters|))
         ((eq fn '|plain|)
          (setq |$specialCharacters| |$plainRTspecialCharacters|))
         (t (|setOutputCharacters| nil))))
       (t (|setOutputCharacters| nil)))))))
```

———-

26.46.11 set output fortran

```
--------------------- The fortran Option ----------------------

 Description: create output in FORTRAN format

 )set output fortran is used to tell AXIOM to turn FORTRAN-style
  output printing on and off, and where to place the output.  By
  default, the destination for the output is the screen but
  printing is turned off.

Also See: )set fortran

Syntax:   )set output fortran <arg>
    where arg can be one of
  on          turn FORTRAN printing on
  off         turn FORTRAN printing off (default state)
  console     send FORTRAN output to screen (default state)
  fp<.fe>     send FORTRAN output to file with file prefix
              fp and file extension .fe. If not given,
              .fe defaults to .sfort.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send FORTRAN output to the file polymer.sfort,
 issue the two commands

  )set output fortran on
  )set output fortran polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

26.46.12 defvar $fortranFormat

— **initvars** —

```
(defvar |$fortranFormat| nil "create output in FORTRAN format ")
```

———-

26.46.13 defvar $fortranOutputFile

— **initvars** —

```
(defvar |$fortranOutputFile| "CONSOLE"
   "where FORTRAN output goes (enter {\em console} or a a pathname)")
```

———-

— **outputfortran** —

```
(|fortran|
 "create output in FORTRAN format "
 |interpreter|
 FUNCTION
 |setOutputFortran|
 (("create output in FORTRAN format "
   LITERALS
   |$fortranFormat|
   (|off| |on|)
   |off|)
  (|break| |$fortranFormat|)
  ("where FORTRAN output goes (enter {\em console} or a a pathname)"
   FILENAME
   |$fortranOutputFile|
   |chkOutputFileName|
   "console"))
 NIL)
```

———-

26.46.14 defun setOutputFortran

[defiostream p1046]
[concat p1107]
[describeSetOutputFortran p929]
[upcase p1140]
[qcdr p**??**]
[qcar p**??**]
[member p1108]
[sayKeyedMsg p39]
[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p**??**]
[makeStream p1047]
[object2String p**??**]
[$fortranOutputStream p**??**]
[$fortranOutputFile p926]
[$filep p**??**]
[$fortranFormat p926]

— **defun setOutputFortran** —

```
(defun |setOutputFortran| (arg)
 (let (label APPEND quiet tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$fortranOutputStream| |$fortranOutputFile| $filep
    |$fortranFormat|))
  (cond
```

```
((eq arg '|%initialize%|)
 (setq |$fortranOutputStream|
  (defiostream '((mode . output) (device . console)) 255 0))
 (setq |$fortranOutputFile| "CONSOLE")
 (setq |$fortranFormat| nil))
((eq arg '|%display%|)
 (if |$fortranFormat|
  (setq label "On:")
  (setq label "Off:"))
 (concat label |$fortranOutputFile|))
((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
 (|describeSetOutputFortran|))
(t
 (DO ()
     ((null (and (listp arg)
                 (|member| (upcase (car arg)) '(append quiet))))
    nil)
  (cond
   ((eq (upcase (car arg)) 'append) (setq append t))
   ((eq (upcase (car arg)) 'quiet) (setq quiet t))
   (t nil))
  (setq arg (cdr arg)))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(Y N YE YES NO O ON OF OFF CONSOLE
          |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|sfort|))))
 (cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg|
      (format nil
       "To toggle %1 printing on and off, specify %l )set output %2 ~
        yes/no/on/off %l Yes, no, on and off cannot be abbreviated.")
      '(fortran |fortran|)))
    ((|member| (upcase fn) '(no off)) (setq |$fortranFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$fortranFormat| t))
    ((eq (upcase fn) 'console)
      (shut |$fortranOutputStream|)
      (setq |$fortranOutputStream|
       (defiostream '((mode . output) (device . console)) 255 0))
      (setq |$fortranOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn
          (setq fn (qcar arg))
          (setq tmp1 (qcdr arg))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq ft (qcar tmp1)) t))))
```

```
       (and (consp arg)
            (progn
             (setq fn (qcar arg))
             (setq tmp1 (qcdr arg))
             (and (consp tmp1)
                  (progn
                   (setq ft (qcar tmp1))
                   (setq tmp2 (qcdr tmp1))
                   (and (consp tmp2)
                        (eq (qcdr tmp2) nil)
                        (progn (setq fm (qcar tmp2)) t)))))))
      (when (setq ptype (|pathnameType| fn))
        (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
        (setq ft ptype))
      (unless fm (setq fm 'a))
      (setq filename ($filep fn ft fm))
      (cond
       ((null filename)
        (|sayKeyedMsg|
         "It is not possible to open or create a file called %1 %2 %3 ."
         (list fn ft fm)))
       ((setq teststream (|makeStream| append filename 255 0))
        (SHUT |$fortranOutputStream|)
        (setq |$fortranOutputStream| teststream)
        (setq |$fortranOutputFile| (|object2String| filename))
        (unless quiet
          (|sayKeyedMsg|
           "%1 output will be written to file %2 ."
           (list 'fortran |$fortranOutputFile|))))
       ((null quiet)
        (|sayKeyedMsg|
         "It is not possible to open or create a file called %1 %2 %3 ."
         (list fn ft fm)))
       (t nil)))
      (t
       (unless quiet (|sayKeyedMsg| "Your argument list is not valid." nil))
       (|describeSetOutputFortran|)))))))
```

———-

26.46.15 defun describeSetOutputFortran

[sayBrightly p??]
[setOutputFortran p927]

— **defun describeSetOutputFortran** —

```
(defun |describeSetOutputFortran| ()
 (|sayBrightly| (list
  " )set output fortran"
  " 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"
  " on and once with the file name. For example, to send "
  '|%l| "FORTRAN output to the file polymer.sfort, issue the two commands"
  '|%l|
  '|%l| "  )set output fortran on"
  '|%l| "  )set output fortran polymer"
  '|%l|
  '|%l| "The output is placed in the directory from which you invoked AXIOM or"
  '|%l| "the one you set with the )cd system command."
  '|%l| "The current setting is: "
  (|setOutputFortran| '|%display%|)
  )))
```

———-

26.46.16 set output 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.
```

26.46.17 defvar $fractionDisplayType

— **initvars** —

```
(defvar |$fractionDisplayType| '|vertical| "how fractions are formatted")
```

———-

— **outputfraction** —

```
(|fraction|
 "how fractions are formatted"
 |interpreter|
 LITERALS
 |$fractionDisplayType|
 (|vertical| |horizontal|)
 |vertical|)
```

———-

26.46.18 set output 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
```

26.46.19 defvar $htmlFormat

— **initvars** —

```
(defvar |$htmlFormat| nil "create output in HTML format ")
```

———-

26.46.20 defvar $htmlOutputFile

— **initvars** —

```
(defvar |$htmlOutputFile| "CONSOLE"
   "where HTML output goes (enter {\em console} or a pathname)")
```

———-

— **outputhtml** —

```
     (|html|
      "create output in HTML style "
      |interpreter|
      FUNCTION
      |setOutputHtml|
      (("create output in HTML format "
        LITERALS
        |$htmlFormat|
        (|off| |on|)
        |off|)
       (|break| |$htmlFormat|)
       ("where HTML output goes (enter {\em console} or a pathname)"
        FILENAME
        |$htmlOutputFile|
        |chkOutputFileName|
        "console"))
      NIL)
```

———-

26.46.21 defun setOutputHtml

[defiostream p1046]
[concat p1107]
[describeSetOutputHtml p934]
[qcdr p??]
[qcar p??]
[member p1108]
[upcase p1140]
[sayKeyedMsg p39]
[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p??]
[make-outstream p1045]

[object2String p??]
[$htmlOutputStream p??]
[$htmlOutputFile p932]
[$htmlFormat p931]
[$filep p??]

— **defun setOutputHtml** —

```
(defun |setOutputHtml| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$htmlOutputStream| |$htmlOutputFile| |$htmlFormat|
          $filep))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$htmlOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$htmlOutputFile| "CONSOLE")
     (setq |$htmlFormat| nil))
    ((eq arg '|%display%|)
     (if |$htmlFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$htmlOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputHtml|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (|member| fn '(y n ye yes no o on of off console
                   |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
       '|ok|)
      (t (setq arg (list fn '|smml|))))
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t))
       (cond
        ((|member| (upcase fn) '(y n ye o of))
         (|sayKeyedMsg|
            (format nil
              "To toggle %1 printing on and off, specify %1 )set output %2 ~
               yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
            '(|HTML| |html|)))
        ((|member| (upcase fn) '(no off)) (setq |$htmlFormat| nil))
        ((|member| (upcase fn) '(yes on)) (setq |$htmlFormat| t))
        ((eq (upcase fn) 'console)
         (shut |$htmlOutputStream|)
         (setq |$htmlOutputStream|
          (defiostream '((mode . output) (device . console)) 255 0))
         (setq |$htmlOutputFile| "CONSOLE"))))
      ((or
        (and (consp arg)
```

```
            (progn
             (setq fn (qcar arg))
             (setq tmp1 (qcdr arg))
             (and (consp tmp1)
                  (eq (qcdr tmp1) nil)
                  (progn (setq ft (qcar tmp1)) t))))
       (and (consp arg)
            (progn (setq fn (qcar arg))
                   (setq tmp1 (qcdr arg))
                   (and (consp tmp1)
                        (progn
                         (setq ft (qcar tmp1))
                         (setq tmp2 (qcdr tmp1))
                         (and (consp tmp2)
                              (eq (qcdr tmp2) nil)
                              (progn
                               (setq fm (qcar tmp2))
                               t)))))))
         (when (setq ptype (|pathnameType| fn))
          (setq fn
           (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
          (setq ft ptype))
         (unless fm (setq fm 'a))
         (setq filename ($filep fn ft fm))
         (cond
          ((null filename)
           (|sayKeyedMsg|
            "It is not possible to open or create a file called %1 %2 %3 ."
            (list fn ft fm)))
          ((setq teststream (make-outstream filename 255 0))
           (shut |$htmlOutputStream|)
           (setq |$htmlOutputStream| teststream)
           (setq |$htmlOutputFile| (|object2String| filename))
           (|sayKeyedMsg|
            "%1 output will be written to file %2 ."
            (list "HTML" |$htmlOutputFile|)))
          (t (|sayKeyedMsg|
              "It is not possible to open or create a file called %1 %2 %3 ."
              (list fn ft fm)))))
      (t
       (|sayKeyedMsg| "Your argument list is not valid." nil)
       (|describeSetOutputHtml|)))))))
```

———-

26.46.22 defun describeSetOutputHtml

[sayBrightly p??]
[setOutputHtml p932]

— **defun describeSetOutputHtml** —

```
(defun |describeSetOutputHtml| ()
```

```
(|sayBrightly| (LIST
" )set output html "
"is used to tell AXIOM to turn HTML-style output"
'|%l| "printing on and off, and where to place the output.  By default, the"
'|%l| "destination for the output is the screen but printing is turned off."
'|%l|
'|%l| "Syntax:   )set output html <arg>"
'|%l| "    where arg can be one of"
'|%l| "  on          turn HTML printing on"
'|%l| "  off         turn HTML printing off (default state)"
'|%l| "  console     send HTML output to screen (default state)"
'|%l| "  fp<.fe>     send HTML output to file with file prefix fp and file"
'|%l| "              extension .fe. If not given, .fe defaults to .stex."
'|%l|
'|%l| "If you wish to send the output to a file, you must issue this command"
'|%l| "twice: once with "
"on and once with the file name. For example, to send"
'|%l| "HTML output to the file polymer.smml, issue the two commands"
'|%l|
'|%l| "  )set output html on"
'|%l| "  )set output html polymer"
'|%l|
'|%l| "The output is placed in the directory from which you invoked AXIOM or"
'|%l| "the one you set with the )cd system command."
'|%l| "The current setting is: "
(|setOutputHtml| '|%display%|)
)))
```

———-

26.46.23 set output 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
```

26.46.24 defvar $margin

— **initvars** —

```
(defvar $margin 3)
```

———-

26.46.25 defvar $linelength

— **initvars** —

```
(defvar $linelength 77 "line length of output displays")
```

————-

— **outputlength** —

```
    (|length|
     "line length of output displays"
     |interpreter|
     INTEGER
     $LINELENGTH
     (10 245)
     77)
```

————-

26.46.26 set output mathml

```
----------------------- The mathml Option ------------------------

 Description: create output in MathML style

 )set output mathml is used to tell AXIOM to turn MathML-style output
printing on and off, and where to place the output.  By default,
the destination for the output is the screen but printing is
turned off.

Syntax:   )set output mathml <arg>
    where arg can be one of
  on          turn MathML printing on
  off         turn MathML printing off (default state)
  console     send MathML output to screen (default state)
  fp<.fe>     send MathML output to file with file prefix fp
              and file extension .fe. If not given,
              .fe defaults to .smml.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send MathML output to the file polymer.smml,
issue the two commands

  )set output mathml on
  )set output mathml polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

26.46.27 defvar $mathmlFormat

— **initvars** —

```
(defvar |$mathmlFormat| nil "create output in MathML format ")
```

————-

26.46.28 defvar $mathmlOutputFile

— **initvars** —

```
(defvar |$mathmlOutputFile| "CONSOLE"
   "where MathML output goes (enter {\em console} or a pathname)")
```

————-

— **outputmathml** —

```
    (|mathml|
     "create output in MathML style "
     |interpreter|
     FUNCTION
     |setOutputMathml|
     (("create output in MathML format "
       LITERALS
       |$mathmlFormat|
       (|off| |on|)
       |off|)
      (|break| |$mathmlFormat|)
      ("where MathML output goes (enter {\em console} or a pathname)"
       FILENAME
       |$mathmlOutputFile|
       |chkOutputFileName|
       "console"))
     NIL)
```

————-

26.46.29 defun setOutputMathml

[defiostream p1046]
[concat p1107]
[describeSetOutputMathml p940]
[qcdr p??]
[qcar p??]
[member p1108]
[upcase p1140]
[sayKeyedMsg p39]

[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p??]
[make-outstream p1045]
[object2String p??]
[$mathmlOutputStream p??]
[$mathmlOutputFile p937]
[$mathmlFormat p937]
[$filep p??]

— **defun setOutputMathml** —

```
(defun |setOutputMathml| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$mathmlOutputStream| |$mathmlOutputFile| |$mathmlFormat|
           $filep))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$mathmlOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$mathmlOutputFile| "CONSOLE")
     (setq |$mathmlFormat| nil))
    ((eq arg '|%display%|)
     (if |$mathmlFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$mathmlOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputMathml|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (|member| fn '(y n ye yes no o on of off console
                   |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
        '|ok|)
      (t (setq arg (list fn '|smml|))))
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t))
       (cond
        ((|member| (upcase fn) '(y n ye o of))
         (|sayKeyedMsg|
            (format nil
              "To toggle %1 printing on and off, specify %1 )set output %2 ~
               yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
           '(|MathML| |mathml|)))
        ((|member| (upcase fn) '(no off)) (setq |$mathmlFormat| nil))
        ((|member| (upcase fn) '(yes on)) (setq |$mathmlFormat| t))
```

```
  ((eq (upcase fn) 'console)
   (shut |$mathmlOutputStream|)
   (setq |$mathmlOutputStream|
    (defiostream '((mode . output) (device . console)) 255 0))
   (setq |$mathmlOutputFile| "CONSOLE"))))
((or
  (and (consp arg)
       (progn
        (setq fn (qcar arg))
        (setq tmp1 (qcdr arg))
        (and (consp tmp1)
             (eq (qcdr tmp1) nil)
             (progn (setq ft (qcar tmp1)) t))))
  (and (consp arg)
       (progn (setq fn (qcar arg))
              (setq tmp1 (qcdr arg))
              (and (consp tmp1)
                   (progn
                    (setq ft (qcar tmp1))
                    (setq tmp2 (qcdr tmp1))
                    (and (consp tmp2)
                         (eq (qcdr tmp2) nil)
                         (progn
                          (setq fm (qcar tmp2))
                          t)))))))
    (when (setq ptype (|pathnameType| fn))
     (setq fn
      (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
     (setq ft ptype))
    (unless fm (setq fm 'a))
    (setq filename ($filep fn ft fm))
    (cond
     ((null filename)
       (|sayKeyedMsg|
         "It is not possible to open or create a file called %1 %2 %3 ."
         (list fn ft fm)))
     ((setq teststream (make-outstream filename 255 0))
      (shut |$mathmlOutputStream|)
      (setq |$mathmlOutputStream| teststream)
      (setq |$mathmlOutputFile| (|object2String| filename))
      (|sayKeyedMsg|
       "%1 output will be written to file %2 ."
       (list "MathML" |$mathmlOutputFile|)))
     (t (|sayKeyedMsg|
         "It is not possible to open or create a file called %1 %2 %3 ."
         (list fn ft fm)))))
(t
 (|sayKeyedMsg| "Your argument list is not valid." nil)
 (|describeSetOutputMathml|)))))))
```

———-

26.46.30 defun describeSetOutputMathml

[sayBrightly p??]
[setOutputMathml p937]

— defun describeSetOutputMathml —

```
(defun |describeSetOutputMathml| ()
 (|sayBrightly| (LIST
 " )set output mathml "
 "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"
 " on and once with the file name. For example, to send"
 '|%l| "MathML output to the file polymer.smml, 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: "
 (|setOutputMathml| '|%display%|)
 )))
```

———-

26.46.31 set output 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
```

26.46.32 defvar $openMathFormat

— **initvars** —

```
(defvar |$openMathFormat| nil "create output in OpenMath format ")
```

———-

26.46.33 defvar $openMathOutputFile

— **initvars** —

```
(defvar |$openMathOutputFile| "CONSOLE"
  "where TeX output goes (enter {\em console} or a pathname)")
```

———-

— **outputopenmath** —

```
    (|openmath|
     "create output in OpenMath style "
     |interpreter|
     FUNCTION
     |setOutputOpenMath|
     (("create output in OpenMath format "
       LITERALS
       |$openMathFormat|
       (|off| |on|)
       |off|)
      (|break| |$openMathFormat|)
      ("where TeX output goes (enter {\em console} or a pathname)"
       FILENAME
       |$openMathOutputFile|
```

```
        |chkOutputFileName|
        "console"))
      NIL)
```

———-

26.46.34 defun setOutputOpenMath

[defiostream p1046]
[concat p1107]
[describeSetOutputOpenMath p944]
[qcdr p??]
[qcar p??]
[member p1108]
[upcase p1140]
[sayKeyedMsg p39]
[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p??]
[make-outstream p1045]
[object2String p??]
[$openMathOutputStream p??]
[$openMathFormat p941]
[$filep p??]
[$openMathOutputFile p941]

— **defun setOutputOpenMath** —

```
(defun |setOutputOpenMath| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$openMathOutputStream| |$openMathFormat| $filep
     |$openMathOutputFile|))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$openMathOutputStream|
      (defiostream  '((mode . output) (device . console)) 255 0))
     (setq |$openMathOutputFile| "CONSOLE")
     (setq |$openMathFormat| NIL))
    ((eq arg '|%display%|)
     (if |$openMathFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$openMathOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputOpenMath|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
```

```
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(y n ye yes no o on of off console
               |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|som|))))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg|
        (format nil
          "To toggle %1 printing on and off, specify %1 )set output %2 ~
           yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
       '(|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|
       "It is not possible to open or create a file called %1 %2 %3 ."
       (list fn ft fm)))
     ((setq teststream (make-outstream filename 255 0))
      (shut |$openMathOutputStream|)
      (setq |$openMathOutputStream| teststream)
      (setq |$openMathOutputFile| (|object2String| filename))
```

```
          (|sayKeyedMsg|
           "%1 output will be written to file %2 ."
           (list "OpenMath" |$openMathOutputFile|)))
         (t
          (|sayKeyedMsg|
           "It is not possible to open or create a file called %1 %2 %3 ."
           (list fn ft fm)))))
      (t
       (|sayKeyedMsg| "Your argument list is not valid." nil)
       (|describeSetOutputOpenMath|)))))))
```

———-

26.46.35 defun describeSetOutputOpenMath

[sayBrightly p??]
[setOutputOpenMath p942]

— **defun describeSetOutputOpenMath** —

```
(defun |describeSetOutputOpenMath| ()
 (|sayBrightly| (list
 " )set output openmath "
 "is used to tell AXIOM to turn OpenMath output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output openmath <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn OpenMath printing on"
 '|%l| "  off         turn OpenMath printing off (default state)"
 '|%l| "  console     send OpenMath output to screen (default state)"
 '|%l|
 "  fp<.fe>     send OpenMath output to file with file prefix fp and file"
 '|%l| "              extension .fe. If not given, .fe defaults to .som."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 " on and once with the file name. For example, to send"
 '|%l| "OpenMath output to the file polymer.som, issue the two commands"
 '|%l|
 '|%l| "  )set output openmath on"
 '|%l| "  )set output openmath polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 (|setOutputOpenMath| '|%display%|)
 )))
```

———-

26.46.36 set output 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
```

26.46.37 defvar $formulaFormat

— **initvars** —

```
(defvar |$formulaFormat| nil "display output in SCRIPT format")
```

———-

26.46.38 defvar $formulaOutputFile

— **initvars** —

```
(defvar |$formulaOutputFile| "CONSOLE"
    "where script output goes (enter {\em console} or a a pathname)")
```

———-

— **outputscript** —

```
(|script|
 "display output in SCRIPT formula format "
 |interpreter|
 FUNCTION
 |setOutputFormula|
 (("display output in SCRIPT format "
   LITERALS
   |$formulaFormat|
   (|off| |on|)
   |off|)
  (|break| |$formulaFormat|)
  ("where script output goes (enter {\em console} or a a pathname)"
   FILENAME
   |$formulaOutputFile|
   |chkOutputFileName|
   "console"))
 NIL)
```

———-

26.46.39 defun setOutputFormula

[defiostream p1046]
[concat p1107]
[describeSetOutputFormula p948]
[qcdr p??]
[qcar p??]
[member p1108]
[upcase p1140]
[sayKeyedMsg p39]
[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p??]
[make-outstream p1045]
[object2String p??]
[$formulaOutputStream p??]
[$formulaOutputFile p945]
[$filep p??]
[$formulaFormat p945]

— **defun setOutputFormula** —

```
(defun |setOutputFormula| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$formulaOutputStream| |$formulaOutputFile| $filep
    |$formulaFormat|))
   (cond
```

```
((eq arg '|%initialize%|)
 (setq |$formulaOutputStream|
  (defiostream '((mode . output) (device . console)) 255 0))
 (setq |$formulaOutputFile| "CONSOLE")
 (setq |$formulaFormat| nil))
((eq arg '|%display%|)
 (if |$formulaFormat|
  (setq label "On:")
  (setq label "Off:"))
 (concat label |$formulaOutputFile|))
((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
 (|describeSetOutputFormula|))
(t
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(y n ye yes no o on of off console
                  |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|sform|))))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg|
        (format nil
          "To toggle %1 printing on and off, specify %1 )set output %2 ~
           yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
        '(|script| |script|)))
    ((|member| (upcase fn) '(no off)) (setq |$formulaFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$formulaFormat| t))
    ((eq (upcase fn) 'console)
     (SHUT |$formulaOutputStream|)
     (setq |$formulaOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$formulaOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn (setq fn (qcar arg))
                (setq tmp1 (qcdr arg))
                (and (consp tmp1)
                     (eq (qcdr tmp1) nil)
                     (progn (setq ft (qcar tmp1)) t))))
    (and (consp arg)
         (progn (setq fn (qcar arg))
                (setq tmp1 (qcdr arg))
                (and (consp tmp1)
                     (progn (setq ft (qcar tmp1))
                            (setq tmp2 (qcdr tmp1))
                            (and (consp tmp2)
                                 (eq (qcdr tmp2) nil)
```

```
                                (progn
                                 (setq fm (qcar tmp2)) t)))))))
      (if (setq ptype (|pathnameType| fn))
        (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
        (setq ft ptype))
      (unless fm (setq fm 'a))
      (setq filename ($filep fn ft fm))
      (cond
       ((null filename)
        (|sayKeyedMsg|
         "It is not possible to open or create a file called %1 %2 %3 ."
         (list fn ft fm)))
       ((setq teststream (make-outstream filename 255 0))
        (shut |$formulaOutputStream|)
        (setq |$formulaOutputStream| teststream)
        (setq |$formulaOutputFile| (|object2String| filename))
        (|sayKeyedMsg|
         "%1 output will be written to file %2 ."
         (list "IBM Script formula" |$formulaOutputFile| )))
       (t
        (|sayKeyedMsg|
         "It is not possible to open or create a file called %1 %2 %3 ."
         (list fn ft fm)))))
    (t
     (|sayKeyedMsg| '"Your argument list is not valid." nil)
     (|describeSetOutputFormula|)))))
```

———-

26.46.40 defun describeSetOutputFormula

[sayBrightly p**??**]
[setOutputFormula p946]

— **defun describeSetOutputFormula** —

```
(defun |describeSetOutputFormula| ()
 (|sayBrightly| (list
 " )set output script "
 "is used to tell AXIOM to turn IBM Script formula-style"
 '|%l|
 "output printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output script <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn IBM Script formula printing on"
 '|%l| "  off         turn IBM Script formula printing off (default state)"
 '|%l| "  console     send IBM Script formula output to screen (default state)"
 '|%l|
 "  fp<.fe>     send IBM Script formula output to file with file prefix fp"
 '|%l|
 "              and file extension .fe. If not given, .fe defaults to .sform."
```

```
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 " on and once with the file name. For example, to send "
 '|%l| "IBM Script formula output to the file polymer.sform,"
 " issue the two commands"
 '|%l|
 '|%l| "  )set output script on"
 '|%l| "  )set output script polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 (|setOutputFormula| '|%display%|)
 )))
```

———-

26.46.41 set output scripts

```
---------------------- The scripts Option ----------------------

 Description: show subscripts,... linearly

 The scripts option may be followed by any one of the following:

    yes
    no

 The current setting is indicated.
```

26.46.42 defvar $linearFormatScripts

— **initvars** —

```
(defvar |$linearFormatScripts| nil "show subscripts,... linearly")
```

———-

— **outputscripts** —

```
    (|scripts|
     "show subscripts,... linearly"
     |interpreter|
     LITERALS
     |$linearFormatScripts|
     (|on| |off|)
     |off|)
```

———-

26.46.43 set output 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.
```

26.46.44 defvar $useEditorForShowOutput

— **initvars** —

```
(defvar |$useEditorForShowOutput| nil "view output of )show in editor")
```

———-

— **outputshoweditor** —

```
     (|showeditor|
      "view output of )show in editor"
      |interpreter|
      LITERALS
      |$useEditorForShowOutput|
      (|on| |off|)
      |off|)
```

———-

26.46.45 set output 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
```

26.46.46 defvar $texFormat

— **initvars** —

```
(defvar |$texFormat| nil "create output in TeX format ")
```

———-

26.46.47 defvar $texOutputFile

— **initvars** —

```
(defvar |$texOutputFile| "CONSOLE"
    "where TeX output goes (enter {\em console} or a pathname)")
```

———-

— **outputtex** —

```
     (|tex|
      "create output in TeX style "
      |interpreter|
      FUNCTION
      |setOutputTex|
      (("create output in TeX format "
        LITERALS
        |$texFormat|
        (|off| |on|)
        |off|)
       (|break| |$texFormat|)
       ("where TeX output goes (enter {\em console} or a pathname)"
```

```
        FILENAME
        |$texOutputFile|
        |chkOutputFileName|
        "console"))
      NIL)
```

26.46.48 defun setOutputTex

[defiostream p1046]
[concat p1107]
[describeSetOutputTex p954]
[qcdr p??]
[qcar p??]
[member p1108]
[upcase p1140]
[sayKeyedMsg p39]
[shut p1046]
[pathnameType p1102]
[pathnameDirectory p1103]
[pathnameName p1102]
[$filep p??]
[make-outstream p1045]
[object2String p??]
[$texOutputStream p??]
[$texOutputFile p951]
[$texFormat p951]
[$filep p??]

— **defun setOutputTex** —

```
(defun |setOutputTex| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
  (declare (special |$texOutputStream| |$texOutputFile| |$texFormat| $filep))
  (cond
   ((eq arg '|%initialize%|)
    (setq |$texOutputStream|
     (defiostream '((mode . output) (device . console)) 255 0))
    (setq |$texOutputFile| "CONSOLE")
    (setq |$texFormat| nil))
   ((eq arg '|%display%|)
    (if |$texFormat|
     (setq label "On:")
     (setq label "Off:"))
    (concat label |$texOutputFile|))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeSetOutputTex|))
   (t
    (cond
     ((and (consp arg)
```

```
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(y n ye yes no o on of off console
              |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|stex| nil))))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg|
         (format nil
           "To toggle %1 printing on and off, specify %1 )set output %2 ~
            yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
        '(|TeX| |tex|)))
    ((|member| (upcase fn) '(no off)) (setq |$texFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$texFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$texOutputStream|)
     (setq |$texOutputStream|
       (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$texOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn (setq fn (qcar arg))
                (setq tmp1 (qcdr arg))
                (and (consp tmp1)
                     (eq (qcdr tmp1) nil)
                     (progn (setq ft (qcar tmp1)) t))))
   (and (consp arg)
        (progn (setq fn (qcar arg))
               (setq tmp1 (qcdr arg))
               (and (consp tmp1)
               (progn (setq ft (qcar tmp1))
                      (setq tmp2 (qcdr tmp1))
                      (and (consp tmp2)
                           (eq (qcdr tmp2) nil)
                           (progn (setq fm (qcar tmp2)) t)))))))
   (when (setq ptype (|pathnameType| fn))
    (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
    (setq ft ptype))
   (unless fm (setq fm 'A))
   (setq filename ($filep fn ft fm))
   (cond
    ((null filename)
      (|sayKeyedMsg|
       "It is not possible to open or create a file called %1 %2 %3 ."
       (list fn ft fm )))
    ((setq teststream (make-outstream filename 255 0))
     (shut |$texOutputStream|)
     (setq |$texOutputStream| teststream)
     (setq |$texOutputFile| (|object2String| filename))
```

```
        (|sayKeyedMsg|
         "%1 output will be written to file %2 ."
         (list "TeX" |$texOutputFile|)))
       (t (|sayKeyedMsg|
           "It is not possible to open or create a file called %1 %2 %3 ."
           (list fn ft fm )))))
     (t
      (|sayKeyedMsg| "Your argument list is not valid." nil)
      (|describeSetOutputTex|)))))
```

———-

26.46.49 defun describeSetOutputTex

[sayBrightly p??]
[setOutputTex p952]

— **defun describeSetOutputTex** —

```
(defun |describeSetOutputTex| ()
 (|sayBrightly| (list
 " )set output tex "
 "is used to tell AXIOM to turn TeX-style output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output tex <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn TeX printing on"
 '|%l| "  off         turn TeX printing off (default state)"
 '|%l| "  console     send TeX output to screen (default state)"
 '|%l| "  fp<.fe>     send TeX output to file with file prefix fp and file"
 '|%l| "              extension .fe. If not given, .fe defaults to .stex."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 " on and once with the file name. For example, to send"
 '|%l| "TeX output to the file polymer.stex, issue the two commands"
 '|%l|
 '|%l| "  )set output tex on"
 '|%l| "  )set output tex polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 (|setOutputTex| '|%display%|)
 )))
```

———-

26.47 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.
```

26.47.1 defvar $quitCommandType

— **initvars** —

```
(defvar |$quitCommandType| '|unprotected| "protected or unprotected quit")
```

———-

— **quit** —

```
  (|quit|
   "protected or unprotected quit"
   |interpreter|
   LITERALS
   |$quitCommandType|
   (|protected| |unprotected|)
   |unprotected|)
```

———-

26.48 streams

```
              Current Values of  streams  Variables

Variable     Description                            Current Value
-----------------------------------------------------------------
calculate    specify number of elements to calculate    10
showall      display all stream elements computed       off
```

— **streams** —

```
  (|streams|
   "set some options for working with streams"
   |interpreter|
   TREE
```

```
   |novar|
   (
\getchunk{streamscalculate}
\getchunk{streamsshowall}
  ))
```

———-

26.48.1 set streams 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 .
```

26.48.2 defvar $streamCount

— **initvars** —

```
(defvar |$streamCount| 10
    "number of initial stream elements you want calculated")
```

———-

— **streamscalculate** —

```
     (|calculate|
      "specify number of elements to calculate "
      |interpreter|
      FUNCTION
      |setStreamsCalculate|
      (("number of initial stream elements you want calculated "
       INTEGER
       |$streamCount|
       (0 NIL)
       10))
      NIL)
```

———-

26.48.3 defun setStreamsCalculate

[object2String p??]
[describeSetStreamsCalculate p957]
[sayMessage p??]
[bright p??]
[terminateSystemCommand p704]
[$streamCount p956]

— **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)))))))
```

———-

26.48.4 defun describeSetStreamsCalculate

[sayKeyedMsg p39]
[$streamCount p956]

— **defun describeSetStreamsCalculate** —

```
(defun |describeSetStreamsCalculate| ()
 (declare (special |$streamCount|))
 (|sayKeyedMsg|
  (format nil
   ")set streams calculate is used to tell Axiom how many elements of a ~
    stream to calculate when a computation uses the stream. The value ~
    given after calculate must either be the word all or a positive ~
    integer.  %l %l The current setting is %1 .")
   (list |$streamCount|)))
```

———-

26.48.5 set streams showall

```
--------------------- The showall Option ----------------------

 Description: display all stream elements computed

 The showall option may be followed by any one of the following:

    on
 -> off

 The current setting is indicated.
```

26.48.6 defvar $streamsShowAll

— **initvars** —

```
(defvar |$streamsShowAll| nil "display all stream elements computed")
```

———-

— **streamsshowall** —

```
    (|showall|
     "display all stream elements computed"
     |interpreter|
     LITERALS
     |$streamsShowAll|
     (|on| |off|)
     |off|)
```

———-

26.49 set system

```
              Current Values of  system  Variables

Variable     Description                            Current Value
-----------------------------------------------------------------
functioncode show gen. LISP for functions when compiled off
optimization show optimized LISP code                   off
prettyprint  prettyprint BOOT func's as they compile    off
```

— **system** —

```
  (|system|
   "set some system development variables"
   |development|
```

```
   TREE
   |novar|
   (
\getchunk{systemfunctioncode}
\getchunk{systemoptimization}
\getchunk{systemprettyprint}
   ))
```

———-

26.49.1 set system 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.
```

26.49.2 defvar $reportCompilation

— **initvars** —

```
(defvar |$reportCompilation| nil "show gen. LISP for functions when compiled")
```

———-

— **systemfunctioncode** —

```
     (|functioncode|
      "show gen. LISP for functions when compiled"
      |development|
      LITERALS
      |$reportCompilation|
      (|on| |off|)
      |off|)
```

———-

26.49.3 set system 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.
```

26.49.4 defvar $reportOptimization

— **initvars** —

```
(defvar |$reportOptimization| nil "show optimized LISP code")
```

———-

— **systemoptimization** —

```
    (|optimization|
     "show optimized LISP code"
     |development|
     LITERALS
     |$reportOptimization|
     (|on| |off|)
     |off|)
```

———-

26.49.5 set system 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.
```

26.49.6 defvar $prettyprint

— **initvars** —

```
(defvar $prettyprint t "prettyprint BOOT func's as they compile")
```

———-

— **systemprettyprint** —

```
(|prettyprint|
 "prettyprint BOOT func's as they compile"
 |development|
 LITERALS
 $prettyprint
 (|on| |off|)
 |on|)
```

———-

26.50 set 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.
```

26.50.1 defvar $UserLevel

— **initvars** —

```
(defvar |$UserLevel| '|development| "operation access level of system user")
```

———-

— **userlevel** —

```
(|userlevel|
 "operation access level of system user"
 |interpreter|
 LITERALS
 |$UserLevel|
 (|interpreter| |compiler| |development|)
 |development|)
```

———-

— **initvars** —

```
(defvar |$setOptions| '(
\getchunk{breakmode}
\getchunk{compile}
\getchunk{debug}
\getchunk{expose}
\getchunk{functions}
\getchunk{fortran}
\getchunk{kernel}
\getchunk{hyperdoc}
\getchunk{help}
\getchunk{history}
\getchunk{messages}
\getchunk{naglink}
\getchunk{output}
\getchunk{quit}
\getchunk{streams}
\getchunk{system}
\getchunk{userlevel}
 ))
```

———-

26.50.2 defvar $setOptionNames

— **initvars** —

```
(defvar |$setOptionNames| (mapcar #'car |$setOptions|))
```

———-

— **postvars** —

```
(eval-when (eval load)
 (|initializeSetVariables| |$setOptions|))
```

———-

26.51 Set code

26.51.1 defun set

[set1 p963]
[$setOptions p??]

— **defun set** —

```
(defun |set| (l)
 (declare (special |$setOptions|))
 (|set1| l |$setOptions|))
```

———-

26.51.2 defun set1

This function will be called with the top level arguments to)set. For instance, given the command

```
)set break break
```

this function gets

```
(set1 (|break| |break|) ....)
```

and given the command

```
)set mes auto off
```

this function gets

```
(set1 (|mes| |auto| |off|) ....)
```

which, because "message" is a TREE, generates the recursive call:

```
(set1 (|auto| |off|) <the message subtree>)
```

The "autoload" subtree contains a FUNCTION called printLoadMessages, which gets called with `%describe%`

[displaySetVariableSettings p860]
[seq p**??**]
[exit p**??**]
[selectOption p728]
[downcase p1140]
[lassoc p**??**]
[satisfiesUserLevel p703]
[sayKeyedMsg p39]
[poundsign p**??**]
[displaySetOptionInformation p858]
[sayMSG p40]
[sayMessage p**??**]
[bright p**??**]
[object2String p**??**]
[translateYesNo2TrueFalse p861]
[use-fast-links p**??**]
[literals p**??**]
[tree p**??**]
[set1 p963]
[$setOptionNames p962]
[$UserLevel p961]
[$displaySetValue p908]

— **defun set1** —

```
(defun |set1| (l settree)
 (let (|$setOptionNames| arg setdata st setfunarg num upperlimit arg2)
  (declare (special |$setOptionNames| |$UserLevel| |$displaySetValue|))
  (cond
   ((null l) (|displaySetVariableSettings| settree '||))
   (t
    (setq |$setOptionNames|
      (do ((t1 settree (cdr t1)) t0 (x nil))
          ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
       (seq
        (exit
         (setq t0 (cons (elt x 0) t0))))))
    (setq arg
     (|selectOption| (downcase (car l)) |$setOptionNames| '|optionError|))
    (setq setdata (cons arg (lassoc arg settree)))
    (cond
     ((null (|satisfiesUserLevel| (third setdata)))
      (|sayKeyedMsg|
        "Your %1 is ambiguous. The following are abbreviated by %2 :"
        (list |$UserLevel| "set option" nil)))
     ((eql 1 (|#| l)) (|displaySetOptionInformation| arg setdata))
     (t
      (setq st (fourth setdata))
      (case (fourth setdata)
       (function
        (setq setfunarg
         (if (eq (elt l 1) 'default)
          '|%initialize%|
          (ifcdr l)))
        (if (canFuncall? (fifth setdata))
          (funcall (fifth setdata) setfunarg)
          (|sayMSG| (concatenate 'string "   Function not implemented. "
           (string (fifth setdata)))))
        (when |$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 |$displaySetValue| (null arg2))
   (|displaySetOptionInformation| arg setdata)))
 (cond
  ((null arg2)
   (|sayMessage|
    `(" Your value" ,@(|bright| (|object2String| (elt l 1)))
      "is not among the valid choices.")))
  (t nil)))
(literals
 (cond
  ((setq arg2
    (|selectOption| (elt l 1)
     (cons '|default| (sixth setdata)) nil))
   (cond
    ((eq arg2 'default)
     (set (fifth setdata)
       (|translateYesNo2TrueFalse| (seventh setdata))))
    (t
     (cond ((eq arg2 '|nobreak|)
       #+:GCL (si::use-fast-links t)))
     (cond
      ((eq arg2 '|fastlinks|)
       #+:GCL (si::use-fast-links nil)
       (setq arg2 '|break|)))
     (set (fifth setdata) (|translateYesNo2TrueFalse| arg2))))))
 (when (or |$displaySetValue| (null arg2))
   (|displaySetOptionInformation| arg setdata))
 (cond
  ((null arg2)
   (|sayMessage|
    (cons " Your value"
     (append (|bright| (|object2String| (elt l 1)))
      (cons "is not among the valid choices." nil)))))
  (t nil)))
(tree (|set1| (ifcdr l) (sixth setdata)) nil)
(t
 (|sayMessage|
  `("Cannot handle set tree node type" ,@(|bright| st) |yet|))
 nil))))))))
```

———-

26.52)show Command

26.52.1 show 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

———-

17

26.52.2 defun The)show command

[showSpad2Cmd p967]

— defun show —

```
(defun |show| (arg) (|showSpad2Cmd| arg))
```

———-

26.52.3 defun The internal)show command

[member p1108]
[helpSpad2Cmd p782]
[sayKeyedMsg p39]
[qcar p??]
[reportOperations p969]
[$showOptions p??]
[$e p285]
[$env p284]
[$InteractiveFrame p34]
[$options p63]

— 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|
            (format nil
             "Record(a:A,...,b:B) %l Record takes any number of ~
```

17 "display" (26.18.3 p 768) "set" (26.51.1 p 962) "what" (26.61.3 p 1007)

```
        selector-domain pairs as arguments: %i %l a, a selector, an ~
        element of domain Symbol %l A, a domain of category ~
        SetCategory %l ... %l b, a selector, an element of domain ~
        Symbol %l B, a domain of category SetCategory %u %l ~
        This constructor is a primitive in Axiom. ~
        The selectors a,...,b of a Record type must be distinct. %l %l ~
        In order for more information to be displayed about %1 , ~
        you must give it specific arguments. For example: %2 %l ~
        You can also use the HyperDoc Browser.")
      (list constr ")show Record(a: Integer, b: String)" )))
   ((eq constr '|Mapping|)
     (|sayKeyedMsg|
      (format nil
       "Mapping(T, S, ...) %l Mapping takes any number of arguments ~
        of the form: %i %l T, a domain of category SetCategory %l ~
        S, a domain of category SetCategory %l ... %u %l ~
        Mapping(T, S, ...) denotes the class of objects which are ~
        mappings from a source domain (S, ...) into a target domain T. ~
        The 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, Mapping(T, A, B) denotes ~
        the class of mappings from (A, B) into T. %l ~
        This constructor is a primitive in Axiom. ~
        For more information, use the HyperDoc Browser.")
      nil))
   (t
    (|sayKeyedMsg|
      (format nil
       "Tagged union: Union(a:A, ..., b:B) %l Union takes any number ~
        of 'tag'-domain pairs of arguments: %i %l a, a tag, an ~
        element of domain Symbol %l A, a domain of category ~
        SetCategory %l ... %l b, a tag, an element of domain ~
        Symbol %l B, a domain of category SetCategory %u %l ~
        This constructor is a primitive in Axiom. ~
        In this tagged Union, tags a, ..., b must be distinct. %l %l ~
        In order for more information to be displayed about %1 , ~
        you must give it specific arguments. For example: %2 %l ~
        You can also use the HyperDoc Browser.")
      (list constr ")show Union(a: Integer, b: String)" )))
    (|sayKeyedMsg|
     (format nil
      "Untagged union: Union(A, ..., B) %l Union takes any number ~
       of domain arguments: %i %l A, a domain of category ~
       SetCategory %l ... %l B, a domain of category SetCategory %u %l ~
       In this untagged form of Union, domains A, ..., B must be ~
       distinct. In order for more information to be displayed about ~
       %1 , you must give it specific arguments. For example: %2 %l ~
       You can also use the HyperDoc Browser.")
      (list constr ")show Union(Integer, String)" ))))
((and (consp constr) (eq (qcar constr) '|Mapping|))
   (|sayKeyedMsg|
     (format nil
      "Mapping(T, S, ...) %l Mapping takes any number of arguments ~
       of the form: %i %l T, a domain of category SetCategory %l ~
```

```
            S, a domain of category SetCategory %l ... %u %l ~
            Mapping(T, S, ...) denotes the class of objects which are ~
            mappings from a source domain (S, ...) into a target domain T. ~
            The 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, Mapping(T, A, B) denotes ~
            the class of mappings from (A, B) into T. %l ~
            This constructor is a primitive in Axiom. ~
            For more information, use the HyperDoc Browser.")
          nil))
    (t (|reportOperations| constr constr))))
  (t (|reportOperations| arg arg)))))))
```

———-

26.52.4 defun reportOperations

[sayBrightly p**??**]
[bright p**??**]
[sayKeyedMsg p39]
[qcar p**??**]
[isNameOfType p**??**]
[isDomainValuedVariable p1019]
[reportOpsFromUnitDirectly0 p974]
[opOf p**??**]
[unabbrev p**??**]
[reportOpsFromLisplib0 p971]
[evaluateType p996]
[mkAtree p**??**]
[removeZeroOneDestructively p**??**]
[isType p**??**]
[$env p284]
[$eval p**??**]
[$genValue p312]
[$quadSymbol p**??**]
[$doNotAddEmptyModeIfTrue p60]

— defun reportOperations —

```
(defun |reportOperations| (oldArg u)
 (let (|$env| |$eval| |$genValue| |$doNotAddEmptyModeIfTrue|
               tmp1 v unitForm tree unitFormp)
 (declare (special |$env| |$eval| |$genValue| |$quadSymbol|
                   |$doNotAddEmptyModeIfTrue|))
  (setq |$env| (list (list nil)))
  (setq |$eval| t)
  (setq |$genValue| t)
  (when u
   (setq |$doNotAddEmptyModeIfTrue| t)
   (cond
    ((equal u |$quadSymbol|)
```

```
 (|sayBrightly|
   (cons "   mode denotes" (append (|bright| "any") (list '|type|)))))
((eq u '%)
 (|sayKeyedMsg|
  (format nil
   "The )show system command is used to display information about ~
    types or partial types.  For example, )show Integer will show ~
    information about Integer.")
   nil)
 (|sayKeyedMsg|
  (format nil
   "%l %% is a special variable holding the result of the last ~
    computation. Issue )display properties %% to see this value.")
    nil))
((and (null (and (consp u) (eq (qcar u) '|Record|)))
      (null (and (consp u) (eq (qcar u) '|Union|)))
      (null (|isNameOfType| u))
      (null (and (consp u)
                 (eq (qcar u) '|typeOf|)
                 (progn
                  (setq tmp1 (qcdr u))
                  (and (consp tmp1) (eq (qcdr tmp1) nil))))))
 (when (atom oldArg) (setq oldArg (list oldArg)))
 (|sayKeyedMsg|
  (format nil
   "The )show system command is used to display information about ~
    types or partial types.  For example, )show Integer will show ~
    information about Integer.")
   nil)
 (dolist (op oldArg)
   (|sayKeyedMsg|
    (format nil
     "%l %1 is not the name of a known type constructor. If you want ~
      to see information about any operations named %1, issue ~
      %ceon )display operations %1 %ceoff")
    (list (|opOf| op)))))
((setq v (|isDomainValuedVariable| u)) (|reportOpsFromUnitDirectly0| v))
(t
  (if (atom u)
   (setq unitForm (|opOf| (|unabbrev| u)))
   (setq unitForm (|unabbrev| u)))
  (if (atom unitForm)
   (|reportOpsFromLisplib0| unitForm u)
   (progn
    (setq unitFormp (|evaluateType| unitForm))
    (setq tree (|mkAtree| (|removeZeroOneDestructively| unitForm)))
    (if (setq unitFormp (|isType| tree))
      (|reportOpsFromUnitDirectly0| unitFormp)
      (|sayKeyedMsg|
       (format nil
        "It is not known what %1p is, so no information about it can be ~
         displayed.")
       (list unitForm)))))))))
```

———-

26.52.5 defun reportOpsFromLisplib0

[reportOpsFromLisplib1 p971]
[reportOpsFromLisplib p971]
[$useEditorForShowOutput p950]

— **defun reportOpsFromLisplib0** —

```
(defun |reportOpsFromLisplib0| (unitForm u)
 (declare (special |$useEditorForShowOutput|))
  (if |$useEditorForShowOutput|
   (|reportOpsFromLisplib1| unitForm u)
   (|reportOpsFromLisplib| unitForm u)))
```

———-

26.52.6 defun reportOpsFromLisplib1

[pathname p1103]
[erase p??]
[defiostream p1046]
[sayShowWarning p979]
[reportOpsFromLisplib p971]
[shut p1046]
[editFile p777]
[$sayBrightlyStream p??]
[$erase p??]

— **defun reportOpsFromLisplib1** —

```
(defun |reportOpsFromLisplib1| (unitForm u)
 (let (|$sayBrightlyStream| showFile)
 (declare (special |$sayBrightlyStream| $erase))
  (setq showFile (|pathname| (list 'show 'listing 'a)))
  ($erase showFile)
  (setq |$sayBrightlyStream|
   (defiostream `((file ,showFile) (mode . output)) 255 0))
  (|sayShowWarning|)
  (|reportOpsFromLisplib| unitForm u)
  (shut |$sayBrightlyStream|)
  (|editFile| showFile)))
```

———-

26.52.7 defun reportOpsFromLisplib

[constructor? p??]
[sayKeyedMsg p39]

[getConstructorSignature p??]
[ifcdr p??]
[getdatabase p1070]
[eqsubstlist p??]
[nreverse0 p??]
[sayBrightly p??]
[concat p1107]
[bright p??]
[form2StringWithWhere p??]
[isExposedConstructor p973]
[concat p1107]
[namestring p1102]
[selectOptionLC p728]
[dc1 p??]
[specialChar p1043]
[remdup p??]
[msort p??]
[form2String p??]
[say2PerLine p??]
[formatAttribute p??]
[displayOperationsFromLisplib p974]
[$linelength p936]
[$showOptions p??]
[$options p63]
[$FormalMapVariableList p15]

— **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| "%1 is unknown, so no information is available." (list u))
   (progn
    (setq argml (when (setq s (|getConstructorSignature| op)) (ifcdr s)))
    (setq typ (getdatabase op 'constructorkind))
    (setq nArgs (|#| argml))
    (setq argList (ifcdr (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| (concat ")edit " (|namestring| sourceFile)))
              (cons "to see algebra source code for"
               (append (|bright| fn) (list '|%l|))))))
;TPDHERE -- add domain doc to output
;    (setq domaindoc (car (cdadar (getdatabase op 'documentation))))
;    (|sayBrightly| (cleanupLine domaindoc))
;    (terpri)
    (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|)
       (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " Attributes ")
       (|sayBrightly| "")
       (setq attList
        (remdup
         (msort
          (mapcar #'(lambda (x) (caar x))
           (reverse (getdatabase op 'attributes))))))
       (if (null attList)
        (|sayBrightly|
         (|concat| (|form2String| functorForm)
                   '|has no attributes.| '|%l|))
         (|say2PerLine| (mapcar #'|formatAttribute| attList))))
      ((eq opt '|operations|)
       (|displayOperationsFromLisplib| functorForm))))))
```

———-

26.52.8 defun isExposedConstructor

[getalist p**??**]
[$localExposureData p147]
[$globalExposureGroupAlist p148]

— **defun isExposedConstructor** —

```
(defun |isExposedConstructor| (name)
 (let (x found)
 (declare (special |$globalExposureGroupAlist| |$localExposureData|))
  (cond
    ((member name '(|Union| |Record| |Mapping|)) t)
```

```
    ((member name (elt |$localExposureData| 2)) nil)
    ((member name (elt |$localExposureData| 1)) t)
    (t
     (loop for g in (elt |$localExposureData| 0)
      when (not found)
      do
        (setq x (getalist |$globalExposureGroupAlist| g))
        (when (and x (getalist x name)) (setq found t)))
     found))))
```

———-

26.52.9 defun displayOperationsFromLisplib

[getdatabase p1070]
[specialChar p1043]
[remdup p??]
[msort p??]
[eqsubstlist p??]
[formatOperationAlistEntry p??]
[say2PerLine p??]
[$FormalMapVariableList p15]
[$linelength p936]

— **defun displayOperationsFromLisplib** —

```
(defun |displayOperationsFromLisplib| (form)
 (let (name argl kind opList opl ops)
 (declare (special |$FormalMapVariableList| $linelength))
  (setq name (car form))
  (setq argl (cdr form))
  (setq kind (getdatabase name 'constructorkind))
  (setq opList (getdatabase name 'operationalist))
  (when opList
    (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " Operations ")
    (setq opl
     (remdup (msort (eqsubstlist argl |$FormalMapVariableList| opList))))
    (setq ops nil)
    (dolist (x opl)
     (setq ops (append ops (|formatOperationAlistEntry| x))))
    (|say2PerLine| ops))))
```

———-

26.52.10 defun reportOpsFromUnitDirectly0

[reportOpsFromUnitDirectly1 p978]
[reportOpsFromUnitDirectly p975]
[$useEditorForShowOutput p950]

— defun reportOpsFromUnitDirectly0 —

```
(defun |reportOpsFromUnitDirectly0| (D)
 (declare (special |$useEditorForShowOutput|))
  (if |$useEditorForShowOutput|
   (|reportOpsFromUnitDirectly1| D)
   (|reportOpsFromUnitDirectly| D)))
```

———-

26.52.11 defun reportOpsFromUnitDirectly

[member p1108]
[qcar p??]
[evalDomain p993]
[getdatabase p1070]
[sayBrightly p??]
[concat p1107]
[formatOpType p??]
[isExposedConstructor p973]
[bright p??]
[sayBrightly p??]
[concat p1107]
[namestring p1102]
[selectOptionLC p728]
[specialChar p1043]
[remdup p??]
[msort p??]
[formatAttribute p??]
[getl p1110]
[systemErrorHere p??]
[nreverse0 p??]
[getOplistForConstructorForm p977]
[say2PerLine p??]
[formatOperation p??]
[$commentedOps p??]
[$CategoryFrame p??]
[$linelength p936]
[$options p63]
[$showOptions p??]

— defun reportOpsFromUnitDirectly —

```
(defun |reportOpsFromUnitDirectly| (unitForm)
 (let (|$commentedOps| isRecordOrUnion unit top kind abb sourceFile verb opt
       attList constructorFunction tmp1 funlist a sigList tmp2)
 (declare (special |$commentedOps| |$CategoryFrame| $linelength |$options|
                   |$showOptions|))
  (setq isRecordOrUnion
   (and (consp unitForm)
        (progn (setq a (qcar unitForm)) t)
```

```
       (|member| a '(|Record| |Union|))))
  (setq unit (|evalDomain| unitForm))
  (setq top (car unitForm))
  (setq kind (getdatabase top 'constructorkind))
  (|sayBrightly|
   (|concat| " " (|formatOpType| unitForm)
             " is a " kind " 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| (concat ")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|)
     (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " Attributes ")
     (if isRecordOrUnion
       (|sayBrightly| "   Records and Unions have no attributes.")
       (progn
        (|sayBrightly| "")
        (setq attList
         (remdup
          (msort
           (mapcar #'(lambda (unit2) (car unit2)) (reverse (elt unit 2))))))
        (|say2PerLine|
         (mapcar #'|formatAttribute| attList))
        nil)))
    ((eq opt '|operations|)
     (setq |$commentedOps| 0)
;      --new form is (<op> <signature> <slotNumber> <condition> <kind>)
     (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " Operations ")
     (|sayBrightly| "")
     (cond
      (isRecordOrUnion
       (setq constructorFunction (getl top '|makeFunctionList|))
       (unless constructorFunction
        (|systemErrorHere| "reportOpsFromUnitDirectly"))
       (setq tmp1
        (funcall constructorFunction '$ unitForm |$CategoryFrame|))
       (setq funlist (car tmp1))
       (setq sigList
```

```
       (remdup
        (msort
         (dolist (fun funlist (nreverse0 tmp2))
          (push '(((,(caar fun) ,(cadar fun)) t (,(caddar fun) 0 1)))
            tmp2))))))
      (t
       (setq sigList
        (remdup (msort (|getOplistForConstructorForm| unitForm))))))
     (|say2PerLine|
      (mapcar #'(lambda (x) (|formatOperation| x unit)) sigList))
     (unless (= |$commentedOps| 0)
      (|sayBrightly|
       (list "Functions that are not yet implemented are preceded by"
             (|bright| "--"))))
     (|sayBrightly| "")))
 nil))
```

———-

26.52.12 defun getOplistForConstructorForm

The new form is an op-Alist which has entries

```
(<op> . signature-Alist)
```

where signature-Alist has entries

```
(<signature> . item)
```

where item has form (¡slotNumber¿ ¡condition¿ ¡kind¿)

```
(<slotNumber> <condition> <kind>)
```

where ¡kind¿ = ELT — CONST — Subsumed — (XLAM..) ..

```
<kind> =  ELT | CONST | Subsumed | (XLAM..) ..
```

— defun getOplistForConstructorForm —

```
(defun |getOplistForConstructorForm| (form)
 (let (argl pairlis opAlist op signatureAlist result)
 (declare (special |$FormalMapVariableList|))
  (setq op (car form))
  (setq argl (cdr form))
  (setq pairlis
   (loop for fv in |$FormalMapVariableList|
         for arg in argl
    collect (cons fv arg)))
  (setq opAlist (|getOperationAlistFromLisplib| op))
  (loop for item in opAlist do
    (setq op (car item))
    (setq signatureAlist (cdr item))
    (setq result
     (append result
      (|getOplistWithUniqueSignatures| op pairlis signatureAlist))))
  result))
```

———-

26.52.13 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))
```

———-

26.52.14 defun reportOpsFromUnitDirectly1

[pathname p1103]
[erase p**??**]
[defiostream p1046]
[sayShowWarning p979]
[reportOpsFromUnitDirectly p975]
[shut p1046]
[editFile p777]
[$sayBrightlyStream p**??**]
[$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)))
```

———-

26.52.15 defun sayShowWarning

[sayBrightly p??]

— **defun sayShowWarning** —

```
(defun |sayShowWarning| ()
 (|sayBrightly|
   "Warning: this is a temporary file and will be deleted the next")
 (|sayBrightly|
   "         time you use )show. Rename it and FILE if you wish to")
 (|sayBrightly| "         save the contents.")
 (|sayBrightly| ""))
```

———-

26.53)spool Command

26.53.1 spool 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
```

————-

18

[18] “cd” (26.11 p 739)

26.54)summary Command

26.54.1 summary 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
)browse       : start an Axiom http server on 127.0.0.1 port 8085
)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
)fin          : drop into lisp, use (restart) to return to the session
)frame        : manage interpreter workspaces
)history      : manage aspects of interactive session
)include      : insert a file into a .input file
)library      : introduce new constructors
)license      : display the Axiom license file
)lisp         : evaluate a LISP expression
)ltrace       : trace functions
)pquit        : ask if you really want to exit Axiom
)quit         : exit Axiom
)read         : execute AXIOM commands from a file
)regress      : regression test an output spool 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
)tangle       : extract chunks from a literate program to an input file
)trace        : trace execution of functions
)trademark    : declare that Axiom is a trademark of this software effort
)undo         : restore workspace to earlier state
)what         : search for various things by name
```

———-

26.54.2 defun summary

[obey p**??**]
[concat p1107]

[getenviron p291]

— **defun summary** —

```
(defun |summary| (l)
 (declare (ignore l))
 (obey (concat "cat " (getenviron "AXIOM") "/doc/spadhelp/summary.help")))
```

————-

26.55)synonym Command

26.55.1 synonym 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
```

———-

19

26.55.2 defun The)synonym command

[synonymSpad2Cmd p984]

— **defun synonym** —

```
(defun |synonym| (&rest ignore)
 (declare (ignore ignore))
 (|synonymSpad2Cmd|))
```

———-

26.55.3 defun The)synonym command implementation

[getSystemCommandLine p985]
[printSynonyms p723]
[processSynonymLine p986]
[putalist p??]
[terminateSystemCommand p704]
[$CommandSynonymAlist p727]

— **defun synonymSpad2Cmd** —

```
(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|)))
```

———-

26.55.4 defun Return a sublist of applicable synonyms

The argument is a list of synonyms, and this returns a sublist of applicable synonyms at the current user level. [string2id-n p??]
[selectOptionLC p728]
[commandsForUserLevel p701]
[$systemCommands p696]
[$UserLevel p961]

19 “set” (26.51.1 p 962) “what” (26.61.3 p 1007)

— 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))
```

———-

26.55.5 defun Get the system command from the input line

[strpos p1106]
[substring p293]
[$currentLine p??]

— defun getSystemCommandLine —

```
(defun |getSystemCommandLine| ()
 (let (p line)
 (declare (special |$currentLine|))
  (setq p (strpos ")" |$currentLine| 0 nil))
  (if p
   (setq line (substring |$currentLine| p nil))
   (setq line |$currentLine|))
 (string-left-trim '(#\space) line)))
```

———-

26.55.6 defun Remove system keyword

[maxindex p??]

— defun processSynonymLine,removeKeyFromLine —

```
(defun |processSynonymLine,removeKeyFromLine| (line)
 (prog (mx)
  (return
   (seq
    (setq line (string-left-trim " " line))
    (setq mx (maxindex line))
    (exit
     (do ((i 0 (1+ i)))
         ((> i mx) nil)
       (seq
        (exit
```

```
(if (char= (elt line i) #\space)
 (exit
  (return
   (do ((j (1+ i) (1+ j)))
       ((> j mx) nil)
     (seq
      (exit
       (if (char\= (elt line j) #\space)
        (exit
         (return
          (substring line j nil))))))))))))))))))
```

———-

26.55.7 defun processSynonymLine

[processSynonymLine,removeKeyFromLine p985]

— **defun processSynonymLine** —

```
(defun |processSynonymLine| (line)
 (cons
  (string2id-n line 1)
  (|processSynonymLine,removeKeyFromLine| line)))
```

———-

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

26.56)system Command

26.56.1 system man page

— system.help —

```
==========================================================================
A.25.  )system
==========================================================================

User Level Required:  interpreter

Command Syntax:

  - )system cmdExpression

Command Description:

This command may be used to issue commands to the operating system while
remaining in AXIOM. The cmdExpression is passed to the operating system for
execution.

To get an operating system shell, issue, for example, )system sh. When you
enter the key combination, Ctrl-D (pressing and holding the Ctrl key and then
pressing the D key) the shell will terminate and you will return to AXIOM. We
do not recommend this way of creating a shell because Lisp may field some
interrupts instead of the shell. If possible, use a shell running in another
window.

If you execute programs that misbehave you may not be able to return to
AXIOM. If this happens, you may have no other choice than to restart AXIOM
and restore the environment via )history )restore, if possible.

Also See:
o )boot
o )fin
o )lisp
o )pquit
o )quit
```

———-

[20]

This command is in the list of `$noParseCommands` 26.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 26.2.1

[20] "boot" (26.4 p 734) "fin" (26.20.2 p 779) "lisp" (26.27 p 831) "pquit" (26.29.2 p 834) "quit" (26.30.2 p 836)

26.57)tangle Command

26.57.1 tangle man page

— tangle.help —

```
====================================================================
A.19.  )tangle
====================================================================

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 (|$InteractiveMode| namestring dot1 dot2 outfile
       (chunkname "*") (extension "input"))
 (declare (special |$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 "~%")))
```

```
(format t (concatenate 'string namestring " file not found~%")))))
```

26.58)trademark Command

26.58.1 trademark man page

— **trademark.help** —

```
====================================================================
A.15.  )trademark
====================================================================

Command Syntax:

  -  )trademark

Command Description:

This command displays the Axiom trademark information.

Also See:
o )license
```

———-

26.59)undo Command

26.59.1 undo man page

— undo.help —

```
=======================================================================
A.27.  )undo
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )undo
  - )undo integer
  - )undo integer [option]
  - )undo )redo

where option is one of

  - )after
  - )before

Command Description:

This command is used to restore the state of the user environment to an
earlier point in the interactive session. The argument of an )undo is an
integer which must designate some step number in the interactive session.

)undo n
)undo n )after

These commands return the state of the interactive environment to that
immediately after step n. If n is a positive number, then n refers to step
nummber n. If n is a negative number, it refers to the nth previous command
(that is, undoes the effects of the last -n commands).

A )clear all resets the )undo facility. Otherwise, an )undo undoes the effect
of )clear with options properties, value, and mode, and that of a previous
undo. If any such system commands are given between steps n and n + 1 (n >
0), their effect is undone for )undo m for any 0 < m <= n .

The command )undo is equivalent to )undo -1 (it undoes the effect of the
previous user expression). The command )undo 0 undoes any of the above system
commands issued since the last user expression.

)undo n )before

This command returns the state of the interactive environment to that
immediately before step n. Any )undo or )clear system commands given before
step n will not be undone.
```

```
)undo )redo

This command reads the file redo.input. created by the last )undo command.
This file consists of all user input lines, excluding those backtracked over
due to a previous )undo.

The command )history )write will eliminate the ‘‘undone’’ command lines of
your program.

Also See:
o )history
```

[21]

26.60 Evaluation

Some Antique Comments About the Interpreter

EVAL BOOT contains the top level interface to the Scratchhpad-II interpreter. The Entry point into the interpreter from the parser is processInteractive.

The type analysis algorithm is contained in the file BOTMUP BOOT, and MODSEL boot, the map handling routines are in MAP BOOT and NEWMAP BOOT, and the interactive coerce routines are in COERCE BOOT and COERCEFN BOOT.

Conventions: All spad values in the interpreter are passed around in triples. These are lists of three items:

```
[value,mode,environment]
```

The value may be wrapped (this is a pair whose CAR is the atom WRAPPED and whose CDR is the value), which indicates that it is a real value, or unwrapped in which case it needs to be EVALed to produce the proper value. The mode is the type of value, and should always be completely specified (not contain `$EmptyMode`). The environment is always empty, and is included for historical reasons.

Modemaps: Modemaps are descriptions of compiled Spad function which the interpreter uses to perform type analysis. They consist of patterns of types for the arguments, and conditions the types must satisfy for the function to apply. For each function name there is a list of modemaps in file modemap DATABASE for each distinct function with that name. The following is the list of the modemaps for "*" (multiplication. The first modemap (the one with the labels) is for module mltiplication which is multiplication of an element of a module by a member of its scalar domain.

This is the signature pattern for the modemap, it is of the form:

```
(DomainOfComputation TargetType <ArgumentType ...>)
      |
      |                This is the predicate that needs to be
      |                 satisfied for the modemap to apply
      |                           |
      V                           |
```

[21] "history" (26.23.11 p 791)

```
    /-----------/                    |
( ( (*1 *1 *2 *1)                    V
      /-------------------------------------------------------/
    ( (AND (ofCategory *1 (Module *2)) (ofCategory *2 (SimpleRing))) )
     . CATDEF) <-- This is the file where the function was defined
  ( (*1 *1 *2 *1)
    ( (AND (isDomain *2 (Integer)) (ofCategory *1 (AbelianGroup))) )
     . CATDEF)
  ( (*1 *1 *2 *1)
    ( (AND
        (isDomain *2 (NonNegativeInteger))
        (ofCategory *1 (AbelianMonoid))) )
     . CATDEF)
  ((*1 *1 *1 *1) ((ofCategory *1 (SemiGroup)) ) . CATDEF)
    )
```

Environments: Environments associate properties with atoms.

Some common properties are:

- **modeSet:** During interpretation we build a modeSet property for each node in the expression. This is (in theory) a list of all the types possible for the node. In the current implementation these modeSets always contain a single type.
- **value:** Value properties are always triples. This is where the values of variables are stored. We also build value properties for internal nodes during the bottom up phase.
- **mode:** This is the declared type of an identifier.

Frequently used global variables:

- `$genValue`: if true then evaluate generated code, otherwise leave code unevaluated. If `$genValue` is false then we are compiling.
- `$op`: name of the top level operator (unused except in map printing)
- `$mapList`: list of maps being type analyzed, used in recursive map type anlysis.
- `$compilingLoop`: true when compiling a loop body, used to control nesting level of interp-only loop CATCH points
- `$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

26.60.1 defun evalDomain

[sayMSG p40]
[concat p1107]

[prefix2String p??]
[startTimingProcess p??]
[eval p??]
[mkEvalable p994]
[stopTimingProcess p??]
[$evalDomain p993]

— **defun evalDomain** —

```
(defun |evalDomain| (form)
 (let (result)
 (declare (special |$evalDomain|))
  (when |$evalDomain|
    (|sayMSG| (|concat| "   instantiating " (|prefix2String| form))))
  (|startTimingProcess| '|instantiation|)
  (setq result (|eval| (|mkEvalable| form)))
  (|stopTimingProcess| '|instantiation|)
  result))
```

———-

26.60.2 defun mkEvalable

[qcar p??]
[qcdr p??]
[mkEvalable p994]
[devaluate p??]
[mkEvalableRecord p996]
[mkEvalableUnion p995]
[mkEvalableMapping p996]
[loadLibIfNecessary p??]
[getdatabase p1070]
[mkq p??]
[constructor? p??]
[fbpip p??]
[bpiname p??]
[$Integer p632]
[$EmptyMode p629]

— **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
       (|loadLibIfNecessary| op t)
       (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))
                ((simple-vector-p x) (mkq x))
                (t
                  (|loadLibIfNecessary| x t)
                  (|mkEvalable| x))))
              ((and (consp x) (eq (qcar x) 'quote)) x)
              ((and (consp x) (eq (qcar x) '|#|) (consp (qcdr x))
                    (eq (qcdr (qcdr x)) nil))
                (list 'size (mkq (qcar (qcdr x)))))
              (t (mkq x))))))
         (t
          (cons op
           (loop for x in argl
            collect (|mkEvalable| x))))))
     ((equal form |$EmptyMode|) |$Integer|)
     ((and (identp form) (|constructor?| form)) (list form))
     ((fbpip form) (bpiname form))
     (t form))))
```

———-

26.60.3 defun mkEvalableUnion

[mkEvalable p994]

— **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))))))
```

———-

26.60.4 defun isTaggedUnion

— **defun isTaggedUnion** —

```
(defun |isTaggedUnion| (u)
 (and (eq (car u) '|Union|) (eq (caadr u) '|:|)))
```

———-

26.60.5 defun mkEvalableRecord

[mkEvalable p994]

— **defun mkEvalableRecord** —

```
(defun |mkEvalableRecord| (form)
  (cons
   (car form)
   (loop for item in (rest form)
    collect (list (quote |:|) (second item) (|mkEvalable| (third item))))))
```

———-

26.60.6 defun mkEvalableMapping

[mkEvalable p994]

— **defun mkEvalableMapping** —

```
(defun |mkEvalableMapping| (form)
 (cons
  (car form)
  (loop for d in (rest form)
   collect (|mkEvalable| d))))
```

———-

26.60.7 defun evaluateType

Takes a parsed, unabbreviated type and evaluates it, replacing type valued variables with their values, and calling bottomUp on non-type valued arguemnts to the constructor and finally checking to see whether the type satisfies the conditions of its modemap.

[isDomainValuedVariable p1019]
[qcar p**??**]
[qcdr p**??**]
[mkAtree p**??**]

[bottomUp p??]
[objVal p462]
[getValue p??]
[evaluateSignature p1001]
[member p1108]
[evaluateType p996]
[constructor? p??]
[throwEvalTypeMsg p1000]
[$EmptyMode p629]
[$expandSegments p??]

— **defun evaluateType** —

```
(defun |evaluateType| (form)
 (let (|$expandSegments| domain formp op argl)
  (declare (special |$expandSegments| |$EmptyMode|))
  (cond
    ((setq domain (|isDomainValuedVariable| form)) domain)
    ((equal form |$EmptyMode|) form)
    ((eq form '?) |$EmptyMode|)
    ((stringp form) form)
    ((eq form '$) form)
    (t
     (setq |$expandSegments| nil)
     (cond
       ((and (consp form) (eq (qcar form) '|typeOf|) (consp (qcdr form))
             (eq (qcdr (qcdr form)) nil))
         (setq formp (|mkAtree| form))
         (|bottomUp| formp)
         (|objVal| (|getValue| formp)))
       ((consp form)
         (setq op (qcar form))
         (setq argl (qcdr form))
         (cond
          ((eq op 'category)
            (cond
             ((consp argl)
               (cons op
                (cons (qcar argl)
                 (loop for s in (qcdr argl)
                  collect (|evaluateSignature| s)))))
             (t form)))
          ((|member| op '(|Join| |Mapping|))
            (cons op
             (loop for arg in argl
              collect (|evaluateType| arg))))
          ((eq op '|Union|)
            (cond
             ((and argl (consp (car argl)) (consp (qcdr (car argl)))
                   (consp (qcdr (qcdr (car argl))))
                   (eq (qcdr (qcdr (qcdr (car argl)))) nil)
                   (|member| (qcar (car argl)) '(|:| |Declare|)))
               (cons op
                (loop for item in argl
```

```
            collect
             (list '|:| (second item) (|evaluateType| (third item))))))
          (t
           (cons op
            (loop for arg in argl
             collect (|evaluateType| arg))))))
     ((eq op '|Record|)
       (cons op
        (loop for item in argl
         collect
          (list '|:| (second item) (|evaluateType| (third item))))))
     ((eq op '|Enumeration|) form)
     (t (|evaluateType1| form))))
  ((|constructor?| form)
    (if (atom form)
      (|evaluateType| (list form))
      (|throwEvalTypeMsg|
       (format nil
        "Although %1 is the name of a constructor, a full type must be ~
         specified in the context you have used it.  Issue )show %2 ~
         for more information.")
        (list form form))))
  (t (|throwEvalTypeMsg| "%1p is not a valid type." (list form))))))))
```

———-

26.60.8 defun Eval args passed to a constructor

Evaluates the arguments passed to a constructor.

[constructor? p**??**]
[getConstructorSignature p**??**]
[throwEvalTypeMsg p1000]
[replaceSharps p1018]
[categoryForm? p**??**]
[evaluateType p996]
[evalCategory p1019]
[getdatabase p1070]
[mkAtree p**??**]
[putTarget p**??**]
[bottumUp p**??**]
[qcar p**??**]
[qcdr p**??**]
[getAndEvalConstructorArgument p1018]
[coerceOrRetract p686]
[objValUnwrap p462]
[throwKeyedMsgCannotCoerceWithValue p**??**]
[makeOrdinal p1000]
[$quadSymbol p**??**]
[$EmptyMode p629]

— defun evaluateType1 —

```
(defun |evaluateType1| (form)
 (let (op argl sig ml xp tree tmp1 m1 z1 zt zv v typeList (argnum 0))
 (declare (special |$quadSymbol| |$EmptyMode|))
  (setq op (car form))
  (setq argl (cdr form))
  (cond
   (((|constructor?| op)
     (cond
      ((null (setq sig (|getConstructorSignature| form)))
       (|throwEvalTypeMsg|
        "You cannot now use %1p in the context you have it." (list form)))
      (t
       (setq ml (cdr sig))
       (setq ml (|replaceSharps| ml form))
       (cond
        ((not (eql (|#| argl) (|#| ml)))
          (|throwEvalTypeMsg|
           (format nil
            "Although %1 is the name of a constructor, a full type must be ~
             specified in the context you have used it.  Issue )show %2 ~
             for more information.")
          (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| "%1p is not a valid type."
                     (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|
                 "Cannot convert the %1 argument of %3p to the type %2p ."
                 (list (|makeOrdinal| (incf argnum)) m form))))))
             typeList)))
       (cons op (nreverse typeList)))))))
    (t (|throwEvalTypeMsg|
        "Category, domain or package constructor %1 is not available."
         (list op))))))
```

———-

26.60.9 defvar $noEvalTypeMsg

— **initvars** —

```
(defvar |$noEvalTypeMsg| nil)
```

———-

26.60.10 defun throwEvalTypeMsg

[spadThrow p**??**]
[throwKeyedMsg p**??**]
[$noEvalTypeMsg p1000]

— **defun throwEvalTypeMsg** —

```
(defun |throwEvalTypeMsg| (msg args)
 (declare (special |$noEvalTypeMsg|))
 (if |$noEvalTypeMsg|
  (|spadThrow|)
  (|throwKeyedMsg| msg args)))
```

———-

26.60.11 defun makeOrdinal

— **defun makeOrdinal** —

```
(defun |makeOrdinal| (i)
 (elt '(|first| |second| |third| |fourth| |fifth| |sixth| |seventh|
        |eighth| |ninth| |tenth|)
  (1- i)))
```

———-

26.60.12 defun evaluateSignature

Calls evaluateType on a signature.

[evaluateType p996]

— 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)))
```

———-

26.60.13 defun recordFrame

[diffAlist p1002]
[seq p??]
[exit p??]
[$frameRecord p45]
[$InteractiveFrame p34]
[$previousBindings p45]

— defun recordFrame —

```
(defun recordFrame (systemNormal)
 (prog (currentAlist delta)
 (declare (special |$frameRecord| |$InteractiveFrame| |$previousBindings|))
  (return
   (seq
     (setq currentAlist (ifcar |$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|))
     ; copy all but the individual properties
     ; note that this loop makes no sense. In boot it read:
     ; [cons(first x, [cons(first y,rest y) for y in rest x]) for x
     ;  in caar $InteractiveFrame
     ; ... but cons(first y, rest y) == y
     (setq |$previousBindings|
        (prog (tmp0)
```

```
        (setq tmp0 nil)
        (return
         (do ((tmp1 (caar |$InteractiveFrame|) (cdr tmp1)) (x nil))
             ((or (atom tmp1)
                  (progn (setq x (car tmp1)) nil))
              (nreverse0 tmp0))
          (seq
           (exit
            (setq tmp0
             (cons
              (cons
               (car x)
               (prog (tmp2)
                (setq tmp2 nil)
                (return
                 (do ((tmp3 (cdr x) (cdr tmp3)) (y nil))
                     ((or (atom tmp3)
                          (progn (setq y (car tmp3)) nil))
                      (nreverse0 tmp2))
                  (seq
                   (exit
                    (setq tmp2 (cons (cons (car y) (cdr y)) tmp2)))))))))
              tmp0)))))))
    (first |$frameRecord|)))))
```

26.60.14 defun diffAlist

```
diffAlist(new,old) ==
--record only those properties which are different
  for (pair := [name,:proplist]) in new repeat
    -- name has an entry both in new and old world
    -- (1) if the old world had no proplist for that variable, then
    --     record NIL as the value of each new property
    -- (2) if the old world does have a proplist for that variable, then
    --     a) for each property with a value: give the old value
    --     b) for each property missing:      give NIL as the old value
    oldPair := ASSQ(name,old) =>
      null (oldProplist := CDR oldPair) =>
      --record old values of new properties as NIL
        acc := [ [name,:[ [prop] for [prop,:.] in proplist] ],:acc]
      deltas := nil
      for (propval := [prop,:val]) in proplist repeat
        null (oldPropval := ASSOC(prop,oldProplist)) => --missing property
          deltas := [ [prop],:deltas]
        EQ(CDR oldPropval,val) => 'skip
        deltas := [oldPropval,:deltas]
      deltas => acc := [ [name,:NREVERSE deltas],:acc]
    acc := [ [name,:[ [prop] for [prop,:.] in proplist] ],:acc]
--record properties absent on new list (say, from a )cl all)
  for (oldPair := [name,:r]) in old repeat
    r and null LASSQ(name,new) =>
```

```
      acc := [oldPair,:acc]
    -- name has an entry both in new and old world
    -- (1) if the new world has no proplist for that variable
    --     (a) if the old world does, record the old proplist
    --     (b) if the old world does not, record nothing
    -- (2) if the new world has a proplist for that variable, it has
    --     been handled by the first loop.
  res := NREVERSE acc
  if BOUNDP '$reportundo and $reportundo then reportUndo res
  res
```

[assq p1110]
[tmp1 p??]
[seq p??]
[exit p??]
[assoc p??]
[lassq p??]
[reportUndo p52]

— defun diffAlist —

```
(defun |diffAlist| (new old)
 (prog (proplist oldPair oldProplist val oldPropval deltas prop name r acc res)
  (return
   (seq
    (progn
     (do ((tmp0 new (cdr tmp0)) (pair nil))
         ((or (atom tmp0)
              (progn (setq pair (car tmp0)) nil)
              (progn
               (progn
                (setq name (car pair))
                (setq proplist (cdr pair))
                pair)
               nil))
             nil)
      (seq
       (exit
        (cond
         ((setq oldPair (assq name old))
          (cond
           ((null (setq oldProplist (cdr oldPair)))
            (setq acc
             (cons
              (cons
               name
               (prog (tmp1)
                (setq tmp1 nil)
                (return
                 (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
                     ((or (atom tmp2)
                          (progn (setq tmp3 (car tmp2)) nil)
                          (progn
                           (progn (setq prop (car tmp3)) tmp3)
```

```
                        nil))
                    (nreverse0 tmp1))
                (seq
                 (exit
                  (setq tmp1 (cons (cons prop nil) tmp1))))))))
            acc)))
       (t
         (setq deltas nil)
         (do ((tmp4 proplist (cdr tmp4)) (|propval| nil))
             ((or (atom tmp4)
                  (progn (setq |propval| (car tmp4)) nil)
                  (progn
                   (progn
                    (setq prop (car |propval|))
                    (setq val (cdr |propval|))
                    |propval|)
                   nil))
              nil)
           (seq
            (exit
             (cond
              ((null (setq oldPropval (|assoc| prop oldProplist)))
                (setq deltas (cons (cons prop nil) deltas)))
              ((eq (cdr oldPropval) val) '|skip|)
              (t (setq deltas (cons oldPropval deltas)))))))
         (when deltas
           (setq acc
            (cons (cons name (nreverse deltas)) acc))))))
    (t
      (setq acc
       (cons
        (cons
         name
         (prog (tmp5)
          (setq tmp5 nil)
          (return
           (do ((tmp6 proplist (cdr tmp6)) (tmp7 nil))
               ((or (atom tmp6)
                    (progn (setq tmp7 (CAR tmp6)) nil)
                    (progn
                     (progn (setq prop (CAR tmp7)) tmp7)
                     nil))
                (nreverse0 tmp5))
             (seq
              (exit
               (setq tmp5 (cons (cons prop nil) tmp5))))))))
        acc)))))
(seq
 (do ((tmp8 old (cdr tmp8)) (oldPair nil))
     ((or (atom tmp8)
          (progn (setq oldPair (car tmp8)) nil)
          (progn
           (progn
            (setq name (car oldPair))
```

```
                (setq r (cdr oldPair))
                oldPair)
               nil))
             nil)
       (seq
        (exit
         (cond
          ((and r (null (lassq name new)))
            (exit
             (setq acc (cons oldPair acc))))))))
      (setq res (nreverse acc))
      (cond
       ((and (boundp '|$reportundo|) |$reportundo|)
         (|reportUndo| res)))
      (exit res)))))))
```

———-

26.60.15 defun clearFrame

[clearCmdAll p745]
[$frameRecord p45]
[$previousBindings p45]

— **defun clearFrame** —

```
(defun |clearFrame| ()
 (declare (special |$frameRecord| |$previousBindings|))
  (|clearCmdAll|)
  (setq |$frameRecord| nil)
  (setq |$previousBindings| nil))
```

———-

26.61)what Command

26.61.1 what man page

— what.help —

```
====================================================================
A.28.  )what
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )what categories pattern1 [pattern2 ...]
  - )what commands   pattern1 [pattern2 ...]
  - )what domains    pattern1 [pattern2 ...]
  - )what operations pattern1 [pattern2 ...]
  - )what packages   pattern1 [pattern2 ...]
  - )what synonym    pattern1 [pattern2 ...]
  - )what things     pattern1 [pattern2 ...]
  - )apropos         pattern1 [pattern2 ...]

Command Description:

This command is used to display lists of things in the system. The patterns
are all strings and, if present, restrict the contents of the lists. Only
those items that contain one or more of the strings as substrings are
displayed. For example,

)what synonym

displays all command synonyms,

)what synonym ver

displays all command synonyms containing the substring ``ver'',

)what synonym ver pr

displays all command synonyms containing the substring ``ver'' or the
substring ``pr''. Output similar to the following will be displayed

---------------- System Command Synonyms -----------------


user-defined synonyms satisfying patterns:
      ver pr


  )apr ........................... )what things
  )apropos ....................... )what things
  )prompt ........................ )set message prompt
```

```
  )version ..................... )lisp *yearweek*

Several other things can be listed with the )what command:

  categories displays a list of category constructors.
  commands  displays a list of  system commands available  at your
    user-level. Your user-level is set via the )set userlevel command. To get
    a description of a particular command, such as ``)what'', issue )help
    what.
  domains   displays a list of domain constructors.
  operations displays a list of operations in  the system library.
    It is recommended that you qualify this command with one or more
    patterns, as there are thousands of operations available. For example,
    say you are looking for functions that involve computation of
    eigenvalues. To find their names, try )what operations eig. A rather
    large list of operations is loaded into the workspace when this command
    is first issued. This list will be deleted when you clear the workspace
    via )clear all or )clear completely. It will be re-created if it is
    needed again.
  packages  displays a list of package constructors.
  synonym  lists system command synonyms.
  things    displays all  of the  above types for  items containing
    the pattern strings as substrings. The command synonym )apropos is
    equivalent to )what things.

Also See:
o )display
o )set
o )show
```

———-

22

26.61.2 defvar $whatOptions

— **initvars** —

```
(defvar |$whatOptions| '(|operations| |categories| |domains| |packages|
                         |commands| |synonyms| |things|))
```

———-

26.61.3 defun what

[whatSpad2Cmd p1008]

— **defun what** —

```
(defun |what| (l)
```

[22] "display" (26.18.3 p 768) "set" (26.51.1 p 962) "show" (26.52.2 p 967)

```
(|whatSpad2Cmd| 1))
```

———-

26.61.4 defun whatSpad2Cmd,fixpat

[qcar p??]
[downcase p1140]

— **defun whatSpad2Cmd,fixpat** —

```
(defun |whatSpad2Cmd,fixpat| (x)
 (let (xp)
  (if (and (consp x) (progn (setq xp (qcar x)) t))
   (downcase xp)
   (downcase x))))
```

———-

26.61.5 defun whatSpad2Cmd

[reportWhatOptions p1009]
[selectOptionLC p728]
[sayKeyedMsg p39]
[seq p??]
[exit p??]
[whatSpad2Cmd,fixpat p1008]
[whatSpad2Cmd p1008]
[filterAndFormatConstructors p1012]
[whatCommands p1010]
[apropos p1013]
[printSynonyms p723]
[$e p285]
[$whatOptions p1007]

— **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|
  (format nil
   "Your argument is not valid for the )what system command. %l %l ~
    Use the )show system command to display the operations for a ~
    constructor. Use the )display operations system command to see ~
    information about an operation. These may be abbreviated to ~
    )sh and )d op, respectively.")
   nil))
(t
 (setq args
  (prog (t0)
   (setq t0 nil)
   (return
     (do ((t1 args (cdr t1)) (p nil))
         ((or (atom t1)
              (progn (setq p (car t1)) nil))
        (nreverse0 t0))
      (seq
       (exit
        (setq t0 (cons (|whatSpad2Cmd,fixpat| p) t0))))))))
 (seq
  (cond
   ((eq key '|things|)
     (do ((t2 |$whatOptions| (cdr t2)) (opt nil))
         ((or (atom t2) (progn (setq opt (CAR t2)) nil)) nil)
      (seq
       (exit
        (cond
         ((null (member opt '(|things|)))
           (exit (|whatSpad2Cmd| (cons opt args)))))))))
   ((eq key '|categories|)
     (|filterAndFormatConstructors| '|category| "Categories" args))
   ((eq key '|commands|) (|whatCommands| args))
   ((eq key '|domains|)
     (|filterAndFormatConstructors| '|domain| "Domains" args))
   ((eq key '|operations|)
      (|apropos| args))
   ((eq key '|packages|)
     (|filterAndFormatConstructors| '|package| "Packages" args))
   (t
    (cond ((eq key '|synonyms|)
             (|printSynonyms| args)))))))))))
```

———-

26.61.6 defun Show keywords for)what command

[sayBrightly p??]
[$whatOptions p1007]

— **defun reportWhatOptions** —

```
(defun |reportWhatOptions| ()
 (let (optlist)
 (declare (special |$whatOptions|))
  (setq optlist
   (reduce #'append
    (mapcar #'(lambda (x) `(|%l| "        " ,x)) |$whatOptions|)))
  (|sayBrightly|
   `("  )what" "argument keywords are" ,@optlist
     |%l| "   or abbreviations thereof." |%l| |%l| "   Issue" ")what ?"
     "for more information."))))
```

———-

26.61.7 defun The)what commands implementation

[concat p1107]
[specialChar p1043]
[filterListOfStrings p1011]
[commandsForUserLevel p701]
[sayMessage p??]
[blankList p??]
[sayAsManyPerLineAsPossible p??]
[say p??]
[sayKeyedMsg p39]
[$systemCommands p696]
[$linelength p936]
[$UserLevel p961]

— **defun whatCommands** —

```
(defun |whatCommands| (patterns)
 (let (label ell)
 (declare (special |$systemCommands| $linelength |$UserLevel|))
  (setq label
   (concat '|System Commands for User Level: |
     (princ-to-string |$UserLevel|)))
  (format t "~v,,,'-:@< ~a ~>~%" (- $linelength 2) label)
  (setq ell
   (|filterListOfStrings| patterns
    (mapcar #'princ-to-string (|commandsForUserLevel| |$systemCommands|))))
  (when patterns
   (if ell
     (|sayMessage|
      `("System commands at this level matching patterns:" |%l| "   "
         ,@(append (|blankList| patterns) (list nil))))
     (|sayMessage|
      `("No system commands at this level matching patterns:" |%l| "   "
        ,@(append (|blankList| patterns) (list nil))))))
  (when ell
   (|sayAsManyPerLineAsPossible| ell)
   (say " "))
  (unless patterns
```

```
(|sayKeyedMsg|
 (format nil
  "For more information about individual commands, use the )help ~
   system command followed by the command name or the command name ~
   followed by a question mark. Some commands (such as )lisp ) may ~
   require the )help lisp format.  For example, issue )help help or ~
   )help %x1 ? to find out more about the help command itself.")
  nil))))
```

———-

26.61.8 defun Find all names contained in a pattern

Names and patterns are lists of strings. This returns a list of strings in names that contains any of the strings in the patterns [satisfiesRegularExpressions p1012]

— **defun filterListOfStrings** —

```
(defun |filterListOfStrings| (patterns names)
 (let (result)
 (if (or (null patterns) (null names))
  names
  (dolist (name (reverse names) result)
   (when (|satisfiesRegularExpressions| name patterns)
    (push name result))))))
```

———-

26.61.9 defun Find function of names contained in pattern

The argument names and patterns are lists of strings. The argument fn is something like CAR or CADR This returns a list of strings in names that contains any of the strings in patterns.

[satisfiesRegularExpressions p1012]

— **defun filterListOfStringsWithFn** —

```
(defun |filterListOfStringsWithFn| (patterns names fn)
 (let (result)
 (if (or (null patterns) (null names))
  names
  (dolist (name (reverse names) result)
   (when (|satisfiesRegularExpressions| (funcall fn name) patterns)
    (push name result))))))
```

———-

26.61.10 defun satisfiesRegularExpressions

[strpos p1106]

— defun satisfiesRegularExpressions —

```
(defun |satisfiesRegularExpressions| (name patterns)
 (let ((dname (downcase (copy name))))
  (dolist (pattern patterns)
   (when (strpos pattern dname 0 "@")
    (return-from nil t)))))
```

———-

26.61.11 defun filterAndFormatConstructors

[sayMessage p??]
[blankList p??]
[pp2Cols p??]
[specialChar p1043]
[filterListOfStringsWithFn p1011]
[whatConstructors p1013]
[function p??]
[$linelength p936]

— defun filterAndFormatConstructors —

```
(defun |filterAndFormatConstructors| (constrType label patterns)
 (prog (l)
 (declare (special $linelength))
  (return
   (progn
    (format t "~v,,,'-:@< ~a ~>~%" (- $linelength 2) label)
    (setq l
     (|filterListOfStringsWithFn| patterns
       (|whatConstructors| constrType)
       #'cdr))
    (cond (patterns
     (cond
      ((null l)
       (|sayMessage|
        (cons "   No "
         (cons label
          (cons " with names matching patterns:"
           (cons '|%l|
            (cons "    "
              (append (|blankList| patterns)
               (cons " " nil)))))))))
      (t
        (|sayMessage|
         (cons label
          (cons " with names matching patterns:"
           (cons '|%l|
```

```
            (cons "    "
              (append (|blankList| patterns)
               (cons " " nil)))))))))))
    (cond (l (|pp2Cols| l)))))))
```

———-

26.61.12 defun whatConstructors

[boot-equal p??]
[getdatabase p1070]
[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)))))))))))))))
```

———-

26.61.13 Display all operation names containing the fragment

Argument l is a list of operation name fragments. This displays all operation names containing these fragments.

[allOperations p1091]
[filterListOfStrings p1011]
[seq p??]
[exit p??]
[downcase p1140]
[sayMessage p??]

[sayAsManyPerLineAsPossible p**??**]
[msort p**??**]
[sayKeyedMsg p39]

— defun apropos —

```
(defun |apropos| (arg)
 "Display all operation names containing the fragment"
 (prog (ops)
  (return
   (seq
    (progn
     (setq ops
      (cond
       ((null arg) (|allOperations|))
       (t
         (|filterListOfStrings|
          (prog (t0)
           (setq t0 nil)
           (return
            (do ((t1 arg (cdr t1)) (p nil))
                ((or (atom t1) (progn (setq p (car t1)) nil))
                  (nreverse0 t0))
            (seq (exit (setq t0 (cons (downcase (princ-to-string p)) t0)))))))
          (|allOperations|)))))
     (cond
      (ops
       (|sayMessage| "Operations whose names satisfy the above pattern(s):")
       (|sayAsManyPerLineAsPossible| (msort ops))
       (|sayKeyedMsg|
        (format nil
         "%1 To get more information about an operation such as %1, issue ~
         the command )display op %1")
         (cons (car ops) nil)))
      (t
       (|sayMessage| "   There are no operations containing those patterns")
       nil)))))))
```

————-

26.62)workfiles Command

26.62.1 workfiles man page

26.62.2 defun workfiles

[workfilesSpad2Cmd p1015]

— **defun workfiles** —

```
(defun |workfiles| (l)
 (|workfilesSpad2Cmd| l))
```

————-

26.62.3 defun workfilesSpad2Cmd

[throwKeyedMsg p??]
[selectOptionLC p728]
[pathname p1103]
[delete p??]
[makeInputFilename p1047]
[sayKeyedMsg p39]
[namestring p1102]
[updateSourceFiles p778]
[say p??]
[specialChar p1043]
[sortby p??]
[sayBrightly p??]
[$options p63]
[$sourceFiles p??]
[$linelength p936]

— **defun workfilesSpad2Cmd** —

```
(defun |workfilesSpad2Cmd| (args)
 (let (deleteflag type flist type1 fl)
  (declare (special |$options| |$sourceFiles| $linelength))
  (cond
   (args
    (|throwKeyedMsg|
     (format nil
      "Arguments are not allowed for the )workfiles system command. ~
       The )boot, )lisp, )meta and )delete options may be used with this ~
       command, however. Issue )help workfiles for more information.")
     nil))
   (t
     (setq deleteflag nil)
     (do ((t0 |$options| (cdr t0)) (t1 nil))
         ((or (atom t0)
              (progn (setq t1 (car t0)) nil)
```

```
          (progn (progn (setq type (car t1)) t1) nil))
       nil)
  (setq type1
     (|selectOptionLC| type '(|boot| |lisp| |meta| |delete|) nil))
  (cond
   ((null type1)
    (|throwKeyedMsg|
     (format nil
      "%1 is not an allowable option for the )workfiles system command. ~
       The )boot, )lisp, )meta and )delete options may be used with this ~
       command, however. Issue )help workfiles for more information.")
     (cons type nil)))
   ((eq type1 '|delete|) (setq deleteflag t))))
(do ((t2 |$options| (cdr t2)) (t3 nil))
    ((or (atom t2)
     (progn (setq t3 (CAR t2)) nil)
     (progn
      (progn
       (setq type (car t3))
       (setq flist (cdr t3)) t3)
      nil))
     nil)
 (setq type1 (|selectOptionLC| type '(|boot| |lisp| |meta| |delete|) nil))
 (unless (eq type1 '|delete|)
   (dolist (file flist)
    (setq fl (|pathname| (list file type1 "*")))
    (cond
     (deleteflag
      (setq |$sourceFiles| (|delete| fl |$sourceFiles|)))
     ((null (makeInputFilename fl))
      (|sayKeyedMsg|
       (format nil
        "The file %1 will not be added to the list of working source ~
         files because the file does not exist.")
       (list (|namestring| fl))))
     (t (|updateSourceFiles| fl))))))
(say " ")
(format t "~v,,,'-:@<~a~>~%" (- $linelength 2)
  " User-specified work files ")
(say " ")
(if (null |$sourceFiles|)
 (say "   no files specified")
 (progn
   (setq |$sourceFiles| (sortby '|pathnameType| |$sourceFiles|))
   (do ((t5 |$sourceFiles| (cdr t5)) (fl nil))
       ((or (atom t5) (progn (setq fl (car t5)) nil)) nil)
      (|sayBrightly| (list "   " (|namestring| fl))))))))))
```

———-

Chapter 27

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.

27.0.4 defun getAndEvalConstructorArgument

[getValue p??]
[objMode p462]
[isWrapped p1485]
[objVal p462]
[isLocalVar p??]
[compFailure p??]
[mkObjWrap p461]
[timedEVALFUN p??]

— **defun getAndEvalConstructorArgument** —

```
(defun |getAndEvalConstructorArgument| (tree)
 (let (triple)
  (setq triple (|getValue| tree))
  (cond
   ((eq (|objMode| triple) '(|Domain|)) triple)
   ((|isWrapped| (|objVal| triple)) triple)
   ((|isLocalVar| (|objVal| triple))
     (|compFailure| "   Local variable or parameter used in type"))
   (t
    (mkObjWrap (|timedEVALFUN| (|objVal| triple)) (|objMode| triple))))))
```

———-

27.0.5 defun replaceSharps

Replaces all sharps in x by the arguments of domain d. Replaces all replaces the triangle variables.

[subCopy p??]
[$TriangleVariableList p??]
[$FormalMapVariableList p15]

— **defun replaceSharps** —

```
(defun |replaceSharps| (x d)
 (let (sl)
 (declare (special |$TriangleVariableList| |$FormalMapVariableList|))
  (loop for e in (rest d) for var in |$FormalMapVariableList|
   do (setq sl (cons (cons var e) sl)))
  (setq x (|subCopy| x sl))
  (setq sl nil)
  (loop for e in (rest d) for var in |$TriangleVariableList|
   do (setq sl (cons (cons var e) sl)))
  (|subCopy| x sl)))
```

———-

27.0.6 defun isDomainValuedVariable

Returns the value of form if form is a variable with a type value.

[identp p1107]
[get p??]
[member p1108]
[objMode p462]
[objValUnwrap p462]
[$e p285]
[$env p284]
[$InteractiveFrame p34]

— **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))))
```

———-

27.0.7 defun evalCategory

[ofCategory p637]
[isPartialMode p638]

— **defun evalCategory** —

```
(defun |evalCategory| (d c)
  (or (|isPartialMode| d) (|ofCategory| d c)))
```

———-

Chapter 28

Handling input files

28.0.8 defun Handle .axiom.input file

[/editfile p755]

— **defun readSpadProfileIfThere** —

```
(defun |readSpadProfileIfThere| ()
 (let ((file (list '|.axiom| '|input|)))
 (declare (special /editfile))
  (when (makeInputFilename file) (setq /editfile file) (/rq))))
```

————-

28.0.9 defvar boot-line-stack

— **initvars** —

```
(defvar boot-line-stack nil "List of lines returned from preparse")
```

————-

28.0.10 defvar in-stream

— **initvars** —

```
(defvar in-stream t "Current input stream.")
```

————-

28.0.11 defvar out-stream

— **initvars** —

```
(defvar out-stream t "Current output stream.")
```

————-

28.0.12 defvar file-closed

— **initvars** —

```
(defvar file-closed nil  "Way to stop EOF tests for console input.")
```

————-

28.0.13 defvar echo-meta

— **initvars** —

```
(defvar echo-meta nil "T if you want a listing of what has been read.")
```

————-

28.0.14 defvar $noSubsumption

— **initvars** —

```
(defvar |$noSubsumption| t)
```

————-

28.0.15 defvar $envHashTable

The `$envHashTable` variable is a hashtable that optimizes lookups in the environment, which normally involve search. This gets populated in the addBinding function.

— **initvars** —

```
(defvar |$envHashTable| nil)
```

————-

28.0.16 defun Dynamically add bindings to the environment

[getProplist p1023]
[addBindingInteractive p1033]
[hput p1105]
[$InteractiveMode p284]
[$envHashTable p1022]

— **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)))))))
```

———-

28.0.17 defun Fetch a property list for a symbol from CategoryFrame

[getProplist p1023]
[search p1024]
[$CategoryFrame p**??**]

— **defun getProplist** —

```
(defun |getProplist| (x e)
 (let (u pl)
 (declare (special |$CategoryFrame|))
  (cond
   ((null (atom x)) (|getProplist| (car x) e))
   ((setq u (|search| x e)) u)
   ((setq pl (|search| x |$CategoryFrame|)) pl))))
```

———-

28.0.18 defun Search for a binding in the environment list

[searchCurrentEnv p1024]
[searchTailEnv p1024]

— **defun search** —

```
(defun |search| (x e)
 (let ((curEnv (car e)) (tailEnv (cdr e)))
  (or (|searchCurrentEnv| x curEnv) (|searchTailEnv| x tailEnv))))
```

———-

28.0.19 defun Search for a binding in the current environment

```
searchCurrentEnv(x,currentEnv) ==
  for contour in currentEnv repeat
    if u:= ASSQ(x,contour) then return (signal:= u)
  KDR signal
```

[assq p1110]

— **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)))))
     (ifcdr signal))))))
```

———-

28.0.20 defun searchTailEnv

```
;searchTailEnv(x,e) ==
;  for env in e repeat
;    signal:=
;      for contour in env repeat
;        if (u:= ASSQ(x,contour)) and ASSQ("FLUID",u) then return (signal:= u)
;      if signal then return signal
```

```
;  KDR signal
```

[assq p1110]

— defun searchTailEnv —

```
(defun |searchTailEnv| (x e)
 (prog (u signal)
  (return
   (seq
    (progn
     (do ((thise e (cdr thise)) (env nil))
         ((or (atom thise) (progn (setq env (car thise)) nil)) nil)
       (seq
        (exit
         (setq signal
          (progn
           (do ((cone env (cdr cone)) (contour nil))
               ((or (atom cone) (progn (setq contour (car cone)) nil)) nil)
             (seq
              (exit
               (cond
                ((and (setq u (assq x contour)) (assq 'fluid u))
                 (return (setq signal u)))
                (t nil)))))
           (cond
            (signal (return signal))
            (t nil)))))))
     (ifcdr signal))))))
```

———-

Chapter 29

Line Handling

29.0.21 Line Buffer

The philosophy of lines is that

- NEXT LINE will always return a non-blank line or fail.
- Every line is terminated by a blank character.

Hence there is always a current character, because there is never a non-blank line, and there is always a separator character between tokens on separate lines. Also, when a line is read, the character pointer is always positioned ON the first character.

29.0.22 defstruct line

— **initvars** —

```
(defstruct line "Line of input file to parse."
           (buffer (make-string 0) :type string)
           (current-char #\Return :type character)
           (current-index 1 :type fixnum)
           (last-index 0 :type fixnum)
           (number 0 :type fixnum))
```

———-

29.0.23 defvar current-line

The current input line.

— **initvars** —

```
(defvar current-line (make-line))
```

———-

29.0.24 defmacro line-clear

[$line p1027]

— **defmacro line-clear 0** —

```
(defmacro line-clear (line)
 `(let ((l ,line))
   (setf (line-buffer l) (make-string 0))
   (setf (line-current-char l) #\return)
   (setf (line-current-index l) 1)
   (setf (line-last-index l) 0)
   (setf (line-number l) 0)))
```

————-

29.0.25 defun line-print

[$line p1027]
[$out-stream p1022]

— **defun line-print 0** —

```
(defun line-print (line)
 (declare (special out-stream))
  (format out-stream "~&~5D> ~A~%" (Line-Number line) (Line-Buffer Line))
  (format out-stream "~v@T^~%" (+ 7 (Line-Current-Index line))))
```

————-

29.0.26 defun line-at-end-p

[$line p1027]

— **defun line-at-end-p 0** —

```
(defun line-at-end-p (line)
  "Tests if line is empty or positioned past the last character."
  (>= (line-current-index line) (line-last-index line)))
```

————-

29.0.27 defun line-past-end-p

[$line p1027]

— **defun line-past-end-p 0** —

```
(defun line-past-end-p (line)
  "Tests if line is empty or positioned past the last character."
  (> (line-current-index line) (line-last-index line)))
```

———-

29.0.28 defun line-next-char

[$line p1027]

— defun line-next-char 0 —

```
(defun line-next-char (line)
  (elt (line-buffer line) (1+ (line-current-index line))))
```

———-

29.0.29 defun line-advance-char

[$line p1027]

— defun line-advance-char 0 —

```
(defun line-advance-char (line)
  (setf (line-current-char line)
        (elt (line-buffer line) (incf (line-current-index line)))))
```

———-

29.0.30 defun line-current-segment

[$line p1027]

— defun line-current-segment 0 —

```
(defun line-current-segment (line)
  "Buffer from current index to last index."
  (if (line-at-end-p line)
   (make-string 0)
   (subseq (line-buffer line)
           (line-current-index line)
           (line-last-index line))))
```

———-

29.0.31 defun line-new-line

[$line p1027]

— defun line-new-line 0 —

```
(defun line-new-line (string line &optional (linenum nil))
  "Sets string to be the next line stored in line."
```

```
  (setf (line-last-index line) (1- (length string)))
  (setf (line-current-index line) 0)
  (setf (line-current-char line)
        (or (and (> (length string) 0) (elt string 0)) #\Return))
  (setf (line-buffer line) string)
  (setf (line-number line) (or linenum (1+ (line-number line)))))
```

———-

29.0.32 defun next-line

[$in-stream p1021]
[$line-handler p1033]

— **defun next-line 0** —

```
(defun next-line (&optional (in-stream t))
 (declare (special in-stream line-handler))
 (funcall Line-Handler in-stream))
```

———-

29.0.33 defun Advance-Char

[Line-At-End-P p??]
[Line-Advance-Char p??]
[next-line p1030]
[current-char p??]
[$in-stream p1021]
[$line p1027]

— **defun Advance-Char** —

```
(defun Advance-Char ()
  "Advances IN-STREAM, invoking Next Line if necessary."
 (declare (special in-stream))
 (loop
  (cond
   ((not (Line-At-End-P Current-Line))
    (return (Line-Advance-Char Current-Line)))
   ((next-line in-stream)
    (return (current-char)))
   ((return nil)))))
```

———-

29.0.34 defun storeblanks

— **defun storeblanks 0** —

```
(defun storeblanks (line n)
 (do ((i 0 (1+ i)))
     ((= i n) line)
  (setf (char line i) #\ )))
```

———-

29.0.35 defun initial-substring

— **defun initial-substring 0** —

```
(defun initial-substring (pattern line)
   (let ((ind (mismatch pattern line)))
     (or (null ind) (eql ind (size pattern)))))
```

———-

29.0.36 defun get-a-line

[is-console p**??**]
[get-a-line mkprompt (vol5)]
[read-a-line p**??**]

get-a-line : **FileStream** → **String** where FileStream might be

```
#<input stream "/research/t1/src/algebra/EQ.spad">
```

and the returned string might be

```
")abbrev domain EQ Equation"
```

— **defun get-a-line 0** —

```
(defun get-a-line (stream)
 (when (is-console stream) (princ (mkprompt)))
 (let ((ll (read-a-line stream)))
  (if (and (stringp ll) (adjustable-array-p ll))
   (make-array (array-dimensions ll) :element-type 'string-char
                 :adjustable t :initial-contents ll)
   ll)))
```

———-

Chapter 30

File Parsing

30.0.37 defun Bind a variable in the interactive environment

[assq p1110]

— **defun addBindingInteractive** —

```
(defun |addBindingInteractive| (var proplist e)
 (let ((curContour (caar e)) u)
  (cond
   ((setq u (assq var curContour)) (rplacd u proplist) e)
   (t (rplac (caar e) (cons (cons var proplist) curContour)) e))))
```

————-

30.0.38 defvar line-handler

— **initvars** —

```
(defparameter line-handler 'next-META-line "Who grabs lines for us.")
```

————-

30.0.39 defvar $spad-errors

— **initvars** —

```
(defvar $spad_errors (vector 0 0 0))
```

————-

30.0.40 defvar xtokenreader

— **initvars** —

```
(defvar xtokenreader 'spadtok)
```

————-

30.0.41 defun Initialize the spad reader

[ioclear p1037]
[$spad-errors p1033]
[spaderrorstream p??]
[*standard-output* p??]
[xtokenreader p1034]
[line-handler p1033]
[meta-error-handler p??]
[file-closed p1022]
[boot-line-stack p1021]

— **defun init-boot/spad-reader** —

```
(defun init-boot/spad-reader ()
 (declare (special $spad_errors spaderrorstream *standard-output*
           xtokenreader line-handler meta-error-handler file-closed
           boot-line-stack))
  (setq $spad_errors (vector 0 0 0))
  (setq spaderrorstream *standard-output*)
  (setq xtokenreader 'get-BOOT-token)
  (setq line-Handler 'next-BOOT-line)
  (setq meta-error-handler 'spad-syntax-error)
  (setq file-closed nil)
  (setq boot-line-stack nil)
  (ioclear))
```

————-

30.0.42 defun spad-syntax-error

[bumperrorcount p??]
[consoleinputp p??]
[spad-long-error p1035]
[spad-short-error p1035]
[ioclear p1037]
[debugmode p??]
[spad-reader p??]

— **defun spad-syntax-error** —

```
(defun spad-syntax-error (&rest byebye)
```

```
  "Print syntax error indication, underline character, scrub line."
  (declare (special debugmode byebye))
  (bumperrorcount '|syntax|)
  (cond ((and (eq debugmode 'yes) (not(consoleinputp in-stream)))
         (spad-long-error))
        ((spad-short-error)))
  (ioclear)
  (throw 'spad_reader nil))
```

———-

30.0.43 defun spad-long-error

[spad-error-loc p1036]
[iostat p1036]
[out-stream p1022]
[spaderrorstream p??]

— **defun spad-long-error** —

```
(defun spad-long-error ()
 (declare (special spaderrorstream))
  (spad-error-loc spaderrorstream)
  (iostat)
  (unless (equal out-stream spaderrorstream)
    (spad-error-loc out-stream)
    (terpri out-stream)))
```

———-

30.0.44 defun spad-short-error

[line-past-end-p p1028]
[line-print p1028]
[$current-line p1027]

— **defun spad-short-error** —

```
(defun spad-short-error ()
 (if (line-past-end-p Current-Line)
    (format t "~&The current line is empty.~%")
    (progn
     (format t "~&The current line is:~%~%")
     (line-print current-line))))
```

———-

30.0.45 defun spad-error-loc

— defun spad-error-loc —

```
(defun spad-error-loc (str)
  (format str "******** Boot Syntax Error detected ********"))
```

————-

30.0.46 defun iostat

[line-past-end-p p1028]
[line-print p1028]
[token-stack-show p1037]
[next-lines-show p1036]
[$boot p734]
[$spad p280]
[$current-line p1027]

— defun iostat —

```
(defun iostat ()
  "Tell me what the current state of the parsing world is."
  (declare (special $boot $spad))
  (if (line-past-end-p Current-Line)
    (format t "~&The current line is empty.~%")
    (progn
     (format t "~&The current line is:~%~%")
     (line-print current-line)))
  (if (or $boot $spad) (next-lines-show))
  (token-stack-show)
  nil)
```

————-

30.0.47 defun next-lines-show

[boot-line-stack p1021]

— defun next-lines-show —

```
(defun next-lines-show ()
  (declare (special boot-line-stack))
  (and boot-line-stack (format t "Currently preparsed lines are:~%~%"))
  (mapcar #'(lambda (line)
              (format t "~&~5D> ~A~%" (car line) (cdr Line)))
          boot-line-stack))
```

————-

30.0.48 defun token-stack-show

[token-type p??]
[valid-tokens p??]
[current-token p??]
[next-token p??]
[prior-token p??]

— **defun token-stack-show** —

```
(defun token-stack-show ()
  (if (= valid-tokens 0)
   (format t "~%There are no valid tokens.~%")
   (format t "~%The number of valid tokens is ~S.~%" valid-tokens))
  (when (> valid-tokens 0)
   (format t "The current token is~%")
   (describe current-token))
  (when (> valid-tokens 1)
   (format t "The next token is~%")
   (describe next-token))
  (when (token-type prior-token)
   (format t "The prior token was~%")
   (describe prior-token)))
```

————-

30.0.49 defun ioclear

The IO state manipulation routines assume that

- one I/O stream pair is in effect at any moment
- there is a current line
- there is a current token and a next token
- there is a reduction stack

[line-clear p1028]
[reduce-stack-clear p??]
[current-fragment p??]
[current-line p1027]
[ioclear token-install (vol9)]
[$boot p734]
[$spad p280]

— **defun ioclear** —

```
(defun ioclear (&optional (in t) (out t))
 (declare (special current-fragment current-line $boot $spad)
          (ignore in out))
  (setq current-fragment nil)
  (line-clear current-line)
  (token-install nil nil current-token nil)
  (token-install nil nil next-token nil)
```

```
(token-install nil nil prior-token nil)
(reduce-stack-clear)
(if (or $boot $spad) (setq boot-line-stack nil))
nil)
```

———-

Chapter 31

Handling output

31.1 Special Character Tables

31.1.1 defvar $defaultSpecialCharacters

— **initvars** —

```
(defvar |$defaultSpecialCharacters| (list
 (int-char 28)    ; upper left corner
 (int-char 27)    ; upper right corner
 (int-char 30)    ; lower left corner
 (int-char 31)    ; lower right corner
 (int-char 79)    ; vertical bar
 (int-char 45)    ; horizontal bar
 (int-char 144)   ; APL quad
 (int-char 173)   ; left bracket
 (int-char 189)   ; right bracket
 (int-char 192)   ; left brace
 (int-char 208)   ; right brace
 (int-char 59)    ; top    box tee
 (int-char 62)    ; bottom box tee
 (int-char 63)    ; right  box tee
 (int-char 61)    ; left   box tee
 (int-char 44)    ; center box tee
 (int-char 224))) ; back slash
```

———-

31.1.2 defvar $plainSpecialCharacters0

— **initvars** —

```
(defvar |$plainSpecialCharacters0| (list
 (int-char 78)     ; upper left corner   (+)
```

```
 (int-char 78)     ; upper right corner  (+)
 (int-char 78)     ; lower left corner   (+)
 (int-char 78)     ; lower right corner  (+)
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

31.1.3 defvar $plainSpecialCharacters1

— **initvars** —

```
(defvar |$plainSpecialCharacters1| (list
 (int-char 107)    ; upper left corner   (,)
 (int-char 107)    ; upper right corner  (,)
 (int-char 125)    ; lower left corner   (')
 (int-char 125)    ; lower right corner  (')
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

31.1.4 defvar $plainSpecialCharacters2

— **initvars** —

```
(defvar |$plainSpecialCharacters2| (list
 (int-char 79)     ; upper left corner   (|)
```

```
 (int-char 79)     ; upper right corner  (|)
 (int-char 79)     ; lower left corner   (|)
 (int-char 79)     ; lower right corner  (|)
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

31.1.5 defvar $plainSpecialCharacters3

— **initvars** —

```
(defvar |$plainSpecialCharacters3| (list
 (int-char 96)     ; upper left corner   (-)
 (int-char 96)     ; upper right corner  (-)
 (int-char 96)     ; lower left corner   (-)
 (int-char 96)     ; lower right corner  (-)
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

31.1.6 defvar $plainRTspecialCharacters

— **initvars** —

```
(defvar |$plainRTspecialCharacters| (list
 (QUOTE +)       ; upper left corner   (+)
```

```
 (QUOTE +)       ; upper right corner  (+)
 (QUOTE +)       ; lower left corner   (+)
 (QUOTE +)       ; lower right corner  (+)
 (QUOTE |\||)    ; vertical bar
 (QUOTE -)       ; horizontal bar      (-)
 (QUOTE ?)       ; APL quad            (?)
 (QUOTE [)       ; left bracket
 (QUOTE ])       ; right bracket
 (QUOTE {)       ; left brace
 (QUOTE })       ; right brace
 (QUOTE +)       ; top    box tee      (+)
 (QUOTE +)       ; bottom box tee      (+)
 (QUOTE +)       ; right  box tee      (+)
 (QUOTE +)       ; left   box tee      (+)
 (QUOTE +)       ; center box tee      (+)
 (QUOTE |\\|)))  ; back slash
```

———-

31.1.7 defvar $RTspecialCharacters

— **initvars** —

```
(defvar |$RTspecialCharacters| (list
 (intern (string (code-char 218))) ;-- upper left corner   (+)
 (intern (string (code-char 191))) ;-- upper right corner  (+)
 (intern (string (code-char 192))) ;-- lower left corner   (+)
 (intern (string (code-char 217))) ;-- lower right corner  (+)
 (intern (string (code-char 179))) ;-- vertical bar
 (intern (string (code-char 196))) ;-- horizontal bar      (-)
 (list (code-char #x1d) (code-char #xe2))
                                   ;-- APL quad           (?)
 (QUOTE [)                         ;-- left bracket
 (QUOTE ])                         ;-- right bracket
 (QUOTE {)                         ;-- left brace
 (QUOTE })                         ;-- right brace
 (intern (string (code-char 194))) ;-- top    box tee      (+)
 (intern (string (code-char 193))) ;-- bottom box tee      (+)
 (intern (string (code-char 180))) ;-- right  box tee      (+)
 (intern (string (code-char 195))) ;-- left   box tee      (+)
 (intern (string (code-char 197))) ;-- center box tee      (+)
 (QUOTE |\\|)))                    ;-- back slash
```

———-

31.1.8 defvar $specialCharacters

— **initvars** —

```
(defvar |$specialCharacters| |$RTspecialCharacters|)
```

———-

31.1.9 defvar $specialCharacterAlist

— **initvars** —

```
(defvar |$specialCharacterAlist|
 '((|ulc| . 0)
   (|urc| . 1)
   (|llc| . 2)
   (|lrc| . 3)
   (|vbar| . 4)
   (|hbar| . 5)
   (|quad| . 6)
   (|lbrk| . 7)
   (|rbrk| . 8)
   (|lbrc| . 9)
   (|rbrc| . 10)
   (|ttee| . 11)
   (|btee| . 12)
   (|rtee| . 13)
   (|ltee| . 14)
   (|ctee| . 15)
   (|bslash| . 16)))
```

———-

31.1.10 defun Look up a special character code for a symbol

This function looks up a symbol in `$specialCharacterAlist`, gets the index into the EBCDIC table, and returns the appropriate character. **TPDHERE: Make this more international, not EBCDIC**.

[ifcdr p**??**]
[assq p1110]
[$specialCharacters p1042]
[$specialCharacterAlist p1043]

— **defun specialChar** —

```
(defun |specialChar| (symbol)
 (let (code)
 (declare (special |$specialCharacters| |$specialCharacterAlist|))
  (if (setq code (ifcdr (assq symbol |$specialCharacterAlist|)))
   (elt |$specialCharacters| code)
   "?")))
```

———-

Chapter 32

Stream and File Handling

32.0.11 defun make-instream

[makeInputFilename p1047]

— **defun make-instream** —

```
(defun make-instream (filespec &optional (recnum 0))
 (declare (ignore recnum))
   (cond ((numberp filespec) (make-synonym-stream '*terminal-io*))
         ((null filespec) (error "not handled yet"))
         (t (open (makeInputFilename filespec)
                  :direction :input :if-does-not-exist nil))))
```

———-

32.0.12 defun make-outstream

[make-filename p??]

— **defun make-outstream** —

```
(defun make-outstream (filespec &optional (width nil) (recnum 0))
 (declare (ignore width) (ignore recnum))
   (cond ((numberp filespec) (make-synonym-stream '*terminal-io*))
         ((null filespec) (error "not handled yet"))
         (t (open (make-filename filespec) :direction :output))))
```

———-

32.0.13 defun make-appendstream

[make-filename p??]

— **defun make-appendstream** —

```
(defun make-appendstream (filespec &optional (width nil) (recnum 0))
 "fortran support"
 (declare (ignore width) (ignore recnum))
 (cond
  ((numberp filespec) (make-synonym-stream '*terminal-io*))
  ((null filespec) (error "make-appendstream: not handled yet"))
  ('else (open (make-filename filespec) :direction :output
          :if-exists :append :if-does-not-exist :create))))
```

———-

32.0.14 defun defiostream

— **defun defiostream** —

```
(defun defiostream (stream-alist buffer-size char-position)
 (declare (ignore buffer-size))
   (let ((mode (or (cdr (assoc 'mode stream-alist)) 'input))
         (filename (cdr (assoc 'file stream-alist)))
         (dev (cdr (assoc 'device stream-alist))))
      (if (eq dev 'console) (make-synonym-stream '*terminal-io*)
       (let ((strm (case mode
                         ((output o) (open (make-filename filename)
                                           :direction :output))
                         ((input i) (open (makeInputFilename filename)
                                           :direction :input)))))
         (if (and (numberp char-position) (> char-position 0))
           (file-position strm char-position))
           strm))))
```

———-

32.0.15 defun shut

[shut is-console (vol9)]

— **defun shut** —

```
(defun shut (st)
  (if (is-console st)
   st
   (if (streamp st) (close st) -1)))
```

———-

32.0.16 defun eofp

— **defun eofp** —

```
(defun eofp (stream) (null (peek-char nil stream nil nil)))
```

———-

32.0.17 defun makeStream

[make-appendstream p1045]
[make-outstream p1045]

— **defun makeStream** —

```
(defun |makeStream| (append filename i j)
 (if append
  (make-appendstream filename i j)
  (make-outstream filename i j)))
```

———-

32.0.18 defun Construct a new input file name

— **defun makeInputFilename** —

```
(defun makeInputFilename (filearg &optional (filetype nil))
 (let*
  ((filename  (make-filename filearg filetype))
   (dirname (pathname-directory filename))
   (ft (pathname-type filename))
   (dirs (getDirectoryList ft))
   (newfn nil))
  (if (or (null dirname) (eqcar dirname :relative))
   (dolist (dir dirs (probeName filename))
    (when (probe-file (setq newfn (concatenate 'string dir filename)))
     (return newfn)))
   (probeName filename))))
```

———-

32.0.19 defun getDirectoryList

[$current-directory p301]
[$UserLevel p961]
[$library-directory-list p176]
[$directory-list p176]

— **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=)))
```

———-

32.0.20 defun probeName

Sometimes we are given a file and sometimes we are given the name of an Axiom KAF (Keyed-Access File). KAF files are actually directories with a single file called "index.kaf". We check for the latter case and return the directory name as the filename, per Axiom convention.

— **defun probeName** —

```
(defun probeName (file)
 (when (or (probe-file file)
           (probe-file (concatenate 'string (namestring file) "/index.kaf")))
 (namestring file)))
```

———-

32.0.21 defun makeFullNamestring

— **defun makeFullNamestring** —

```
(defun makeFullNamestring (filearg &optional (filetype nil))
  (namestring (merge-pathnames (make-filename filearg filetype))))
```

———-

32.0.22 defun Replace a file by erase and rename

[makeFullNamestring p1048]

— **defun replaceFile** —

```
(defun replaceFile (filespec1 filespec2)
 ($erase (setq filespec1 (makeFullNamestring filespec1)))
 (rename-file (makeFullNamestring filespec2) filespec1))
```

———-

Chapter 33

The Spad Server Mechanism

33.0.23 defun openserver

This is a cover function for the C code used for communication interface.

— **defun openserver** —

```
(defun openserver (name)
  (open_server name))
```

———-

Chapter 34

Axiom Build-time Functions

34.0.24 defun spad-save

The **spad-save** function is just a cover function for more lisp system specific save functions. There is no standard name for saving a lisp image so we make one and conditionalize it at compile time.

This function is passed the name of an image that will be saved. The saved image contains all of the loaded functions.

This is used in the src/interp/Makefile.pamphlet in three places:

- creating depsys, an image for compiling axiom.

  Some of the Common Lisp code we compile uses macros which are assumed to be available at compile time. The **DEPSYS** image is created to contain the compile time environment and saved. We pipe compile commands into this environment to compile from Common Lisp to machine dependent code.

  ```
  DEPSYS=${OBJ}/${SYS}/bin/depsys
  ```

- creating savesys, an image for running axiom.

  Once we've compile all of the Common Lisp files we fire up a clean lisp image called **LOADSYS**, load all of the final executable code and save it out as **SAVESYS**. The **SAVESYS** image is copied to the `${MNT}/${SYS}/bin` subdirectory and becomes the axiom executable image.

  ```
  LOADSYS= ${OBJ}/${SYS}/bin/lisp
  SAVESYS= ${OBJ}/${SYS}/bin/interpsys
  AXIOMSYS= ${MNT}/${SYS}/bin/AXIOMsys
  ```

- creating debugsys, an image with all interpreted functions loaded.

  Occasionally we need to really get into the system internals. The best way to do this is to run almost all of the lisp code interpreted rather than compiled (note that cfuns.lisp and sockio.lisp still need to be loaded in compiled form as they depend on the loader to link with lisp internals). This image is nothing more than a load of the file src/interp/debugsys.lisp.pamphlet. If you need to make test modifications you can add code to that file and it will show up here.

  ```
  DEBUGSYS=${OBJ}/${SYS}/bin/debugsys
  ```

[save-system p**??**]
[$SpadServer p178]
[$openServerIfTrue p177]

— **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 35

Exposure Groups

Exposure groups are a way of controlling the namespace available to the user. Certain algebra files are only useful for internal purposes but they contain functions have common names (like "map". In order to separate the user visible functions from the internal functions the algebra files are collected into "exposure groups". These large groups are grouped into sets in the variable `$globalExposureGroupAlist`.

Exposure group information is kept in the local frame. For more information "The Frame Mechanism" 3.1 on page 15.

Chapter 36

Databases

36.1 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.

36.1.1 kaf File Format

This documentation refers to kaf files which are random access files. nrlib files are kaf files (look for nrlib/index.kaf) The format of a random access file is

```
byte-offset-of-key-table
first-entry
second-entry
...
last-entry
((key1 . first-entry-byte-address)
 (key2 . second-entry-byte-address)
 ...
 (keyN . last-entry-byte-address))
```

The key table is a standard lisp alist.

To open a database you fetch the first number, seek to that location, and (read) which returns the key-data alist. To look up data you index into the key-data alist, find the ith-entry-byte-address, seek to that address, and (read).

For instance, see src/share/algebra/users.daase/index.kaf

One existing optimization is that if the data is a simple thing like a symbol then the nth-entry-byte-address is replaced by immediate data.

Another existing one is a compression algorithm applied to the data so that the very long names don't take up so much space. We could probably remove the compression algorithm as 64k is no longer considered 'huge'. The database-abbreviation routine handles this on read and write-compress handles this on write.

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.

36.1.2 Database Files

Database files are very similar to kaf files except that there is an optimization (currently broken) which makes the first item a pair of two numbers. The first number in the pair is the offset of the key-value table, the second is a time stamp. If the time stamp in the database matches the time stamp in the image the database is not needed (since the internal hash tables already contain all of the information). When the database is built the time stamp is saved in both the gcl image and the database.

Regarding the 'ancestors field 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.

Each constructor (e.g. LIST) had one library directory (e.g. LIST.nrlib). This directory contained a random access file called the index.kaf file. These files contain runtime information such as the operationAlist and the ConstructorModemap. At system build time we merge all of these .nrlib/index.kaf files into one database, INTERP.daase. Requests to get information from this database are cached so that multiple references do not cause additional disk i/o.

This database is left open at all times as it is used frequently by the interpreter. one minor complication is that newly compiled files need to override information that exists in this

database.

The design calls for constructing a random read (kaf format) file that is accessed by functions that cache their results. when the database is opened the list of constructor-index pairs is hashed by constructor name. a request for information about a constructor causes the information to replace the index in the hash table. since the index is a number and the data is a non-numeric sexpr there is no source of confusion about when the data needs to be read.

The format of this new database is as follows:

```
first entry:
  an integer giving the byte offset to the constructor alist
  at the bottom of the file
second and subsequent entries (one per constructor)
  (operationAlist)
  (constructorModemap)
  ....
last entry: (pointed at by the first entry)
  an alist of (constructor . index) e.g.
     ( (PI offset-of-operationAlist offset-of-constructorModemap)
     (NNI offset-of-operationAlist offset-of-constructorModemap)
      ....)
 This list is read at open time and hashed by the car of each item.
```

The system has been changed to use the property list of the symbols rather than hash tables. since we already hashed once to get the symbol we need only an offset to get the property list. this also has the advantage that eq hash tables no longer need to be moved during garbage collection.

There are 3 potential speedups that could be done.

- the best would be to use the value cell of the symbol rather than the property list but i'm unable to determine all uses of the value cell at the present time.
- a second speedup is to guarantee that the property list is a single item, namely the database structure. this removes an assoc but leaves one open to breaking the system if someone adds something to the property list. this was not done because of the danger mentioned.
- a third speedup is to make the getdatabase call go away, either by making it a macro or eliding it entirely. this was not done because we want to keep the flexibility of changing the database forms.

The new design does not use hash tables. the database structure contains an entry for each item that used to be in a hash table. initially the structure contains file-position pointers and these are replaced by real data when they are first looked up. the database structure is kept on the property list of the constructor, thus, (get '—DenavitHartenbergMatrix— 'database) will return the database structure object.

Each operation has a property on its symbol name called 'operation which is a list of all of the signatures of operations with that name.

36.1.3 defstruct database

— initvars —

```
(defstruct database
 abbreviation               ; interp.
 ancestors                  ; interp.
 constructor                ; interp.
 constructorcategory        ; interp.
 constructorkind            ; interp.
 constructormodemap         ; interp.
 cosig                      ; interp.
 defaultdomain              ; interp.
 modemaps                   ; interp.
 niladic                    ; interp.
 object                     ; interp.
 operationalist             ; interp.
 documentation              ; browse.
 constructorform            ; browse.
 attributes                 ; browse.
 predicates                 ; browse.
 sourcefile                 ; browse.
 parents                    ; browse.
 users                      ; browse.
 dependents                 ; browse.
 spare                      ; superstition
 ) ; database structure
```

———-

36.1.4 defvar *defaultdomain-list*

There are only a small number of domains that have default domains. rather than keep this slot in every domain we maintain a list here.

— **initvars** —

```
(defvar *defaultdomain-list* '(
  (|MultisetAggregate| |Multiset|)
  (|FunctionSpace| |Expression|)
  (|AlgebraicallyClosedFunctionSpace| |Expression|)
  (|ThreeSpaceCategory| |ThreeSpace|)
  (|DequeueAggregate| |Dequeue|)
  (|ComplexCategory| |Complex|)
  (|LazyStreamAggregate| |Stream|)
  (|AssociationListAggregate| |AssociationList|)
  (|QuaternionCategory| |Quaternion|)
  (|PriorityQueueAggregate| |Heap|)
  (|PointCategory| |Point|)
  (|PlottableSpaceCurveCategory| |Plot3D|)
  (|PermutationCategory| |Permutation|)
  (|StringCategory| |String|)
  (|FileNameCategory| |FileName|)
  (|OctonionCategory| |Octonion|)))
```

———-

36.1.5 defvar *operation-hash*

— **initvars** —

```
(defvar *operation-hash* nil "given an operation name, what are its modemaps?")
```

———-

36.1.6 defvar *hasCategory-hash*

This hash table is used to answer the question"does domain x have category y?". this is answered by constructing a pair of (x . y) and doing an equal hash into this table.

— **initvars** —

```
(defvar *hasCategory-hash* nil "answers x has y category questions")
```

———-

36.1.7 defvar *miss*

This variable is used for debugging. If a hash table lookup fails and this variable is non-nil then a message is printed.

— **initvars** —

```
(defvar *miss* nil "print out cache misses on getdatabase calls")
```

———-

Note that constructorcategory information need only be kept for items of type category. this will be fixed in the next iteration when the need for the various caches are reviewed

Note that the *modemaps-hash* information does not need to be kept for system files. these are precomputed and kept in modemap.daase however, for user-defined files these are needed. Currently these are added to the database for 2 reasons; there is a still-unresolved issue of user database extensions and this information is used during database build time

36.1.8 Database streams

This are the streams for the databases. They are always open. There is an optimization for speeding up system startup. If the database is opened and the ..-stream-stamp* variable matches the position information in the database then the database is NOT read in and is assumed to match the in-core version

36.1.9 defvar *interp-stream*

— **initvars** —

```
(defvar *interp-stream* nil "an open stream to the interpreter database")
```

———-

36.1.10 defvar *interp-stream-stamp*

— **initvars** —

```
(defvar *interp-stream-stamp* 0 "*interp-stream* (position . time)")
```

———-

36.1.11 defvar *operation-stream*

This is indexed by operation, not constructor

— **initvars** —

```
(defvar *operation-stream* nil "the stream to operation.daase")
```

———-

36.1.12 defvar *operation-stream-stamp*

— **initvars** —

```
(defvar *operation-stream-stamp* 0 "*operation-stream* (position . time)")
```

———-

36.1.13 defvar *browse-stream*

— **initvars** —

```
(defvar *browse-stream* nil "an open stream to the browser database")
```

———-

36.1.14 defvar *browse-stream-stamp*

— **initvars** —

```
(defvar *browse-stream-stamp* 0 "*browse-stream* (position . time)")
```

———-

36.1.15 defvar *category-stream*

This is indexed by (domain . category)

— **initvars** —

```
(defvar *category-stream* nil "an open stream to the category table")
```

————-

36.1.16 defvar *category-stream-stamp*

— **initvars** —

```
(defvar *category-stream-stamp* 0 "*category-stream* (position . time)")
```

————-

36.1.17 defvar *allconstructors*

— **initvars** —

```
(defvar *allconstructors* nil "a list of all the constructors in the system")
```

————-

36.1.18 defvar *allOperations*

— **initvars** —

```
(defvar *allOperations* nil "a list of all the operations in the system")
```

————-

36.1.19 defun Reset all hash tables before saving system

[interpopen p1064]
[operationopen p1067]
[browseopen p1065]
[categoryopen p1066]
[initial-getdatabase p1062]
[*sourcefiles* p1077]
[*interp-stream* p1059]
[*operation-stream* p1060]
[*category-stream* p1061]
[*browse-stream* p1060]
[*category-stream-stamp* p1061]

[*operation-stream-stamp* p1060]
[*interp-stream-stamp* p1060]
[*allconstructors* p1061]
[*operation-hash* p1059]
[*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))
```

———-

36.1.20 defun Preload algebra into saved system

[getdatabase p1070]
[getenviron p291]

— **defun initial-getdatabase** —

```
(defun initial-getdatabase ()
 "fetch data we want in the saved system"
 (let (hascategory constructormodemapAndoperationalist operation constr)
 (format t "Initial getdatabase~%")
 (setq hascategory '(
  (|Equation| . |Ring|)
  (|Expression| . |CoercibleTo|) (|Expression| . |CommutativeRing|)
  (|Expression| . |IntegralDomain|) (|Expression| . |Ring|)
  (|Float| . |RetractableTo|)
  (|Fraction| . |Algebra|) (|Fraction| . |CoercibleTo|)
  (|Fraction| . |OrderedSet|) (|Fraction| . |RetractableTo|)
```

```
 (|Integer| . |Algebra|) (|Integer| . |CoercibleTo|)
 (|Integer| . |ConvertibleTo|) (|Integer| . |LinearlyExplicitRingOver|)
 (|Integer| . |RetractableTo|)
 (|List| . |CoercibleTo|) (|List| . |FiniteLinearAggregate|)
 (|List| . |OrderedSet|)
 (|Polynomial| . |CoercibleTo|) (|Polynomial| . |CommutativeRing|)
 (|Polynomial| . |ConvertibleTo|) (|Polynomial| . |OrderedSet|)
 (|Polynomial| . |RetractableTo|)
 (|Symbol| . |CoercibleTo|) (|Symbol| . |ConvertibleTo|)
 (|Variable| . |CoercibleTo|)))
(dolist (pair hascategory) (getdatabase pair 'hascategory))
(setq constructormodemapAndoperationalist '(
 |BasicOperator|  |Boolean|
 |CardinalNumber| |Color|  |Complex|
 |Database|
 |Equation| |EquationFunctions2| |Expression|
 |Float| |Fraction| |FractionFunctions2|
 |Integer| |IntegralDomain|
 |Kernel|
 |List|
 |Matrix| |MappingPackage1|
 |Operator| |OutputForm|
 |NonNegativeInteger|
 |ParametricPlaneCurve| |ParametricSpaceCurve| |Point| |Polynomial|
 |PolynomialFunctions2| |PositiveInteger|
 |Ring|
 |SetCategory| |SegmentBinding| |SegmentBindingFunctions2| |DoubleFloat|
 |SparseMultivariatePolynomial| |SparseUnivariatePolynomial| |Segment|
 |String| |Symbol|
 |UniversalSegment|
 |Variable|  |Vector|))
(dolist (con constructormodemapAndoperationalist)
 (getdatabase con 'constructormodemap)
 (getdatabase con 'operationalist))
(setq operation '(
 |+| |-| |*| |/| |**| |coerce| |convert| |elt| |equation|
 |float| |sin| |cos| |map| |SEGMENT|))
(dolist (op operation) (getdatabase op 'operation))
(setq constr '( ;these are sorted least-to-most freq. delete early ones first
 |Factored| |SparseUnivariatePolynomialFunctions2| |TableAggregate&|
 |RetractableTo&| |RecursiveAggregate&| |UserDefinedPartialOrdering|
 |None| |UnivariatePolynomialCategoryFunctions2| |IntegerPrimesPackage|
 |SetCategory&| |IndexedExponents| |QuotientFieldCategory&| |Polynomial|
 |EltableAggregate&| |PartialDifferentialRing&| |Set|
 |UnivariatePolynomialCategory&| |FlexibleArray|
 |SparseMultivariatePolynomial| |PolynomialCategory&|
 |DifferentialExtension&| |IndexedFlexibleArray| |AbelianMonoidRing&|
 |FiniteAbelianMonoidRing&| |DivisionRing&| |FullyLinearlyExplicitRingOver&|
 |IndexedVector| |IndexedOneDimensionalArray| |LocalAlgebra| |Localize|
 |Boolean| |Field&| |Vector| |IndexedDirectProductObject| |Aggregate&|
 |PolynomialRing| |FreeModule| |IndexedDirectProductAbelianGroup|
 |IndexedDirectProductAbelianMonoid| |SingletonAsOrderedSet|
 |SparseUnivariatePolynomial| |Fraction| |Collection&| |HomogeneousAggregate&|
 |RepeatedSquaring| |IntegerNumberSystem&| |AbelianSemiGroup&|
```

```
  |AssociationList| |OrderedRing&| |SemiGroup&| |Symbol|
  |UniqueFactorizationDomain&| |EuclideanDomain&| |IndexedAggregate&|
  |GcdDomain&| |IntegralDomain&| |DifferentialRing&| |Monoid&| |Reference|
  |UnaryRecursiveAggregate&| |OrderedSet&| |AbelianGroup&| |Algebra&|
  |Module&| |Ring&| |StringAggregate&| |AbelianMonoid&|
  |ExtensibleLinearAggregate&| |PositiveInteger| |StreamAggregate&|
  |IndexedString| |IndexedList| |ListAggregate&| |LinearAggregate&|
  |Character| |String| |NonNegativeInteger| |SingleInteger|
  |OneDimensionalArrayAggregate&| |FiniteLinearAggregate&| |PrimitiveArray|
  |Integer| |List| |OutputForm|))
 (dolist (con constr)
  (let ((c (concatenate 'string
             (getenviron "AXIOM") "/algebra/"
             (string (getdatabase con 'abbreviation)) ".o")))
    (format t "   preloading ~a.." c)
    (if (probe-file c)
     (progn
      (put con 'loaded c)
      (load c)
      (format t "loaded.~%"))
     (format t "skipped.~%"))))
 (format t "~%")))
```

———-

36.1.21 defun Open the interp database

Format of an entry in interp.daase:

```
  (constructor-name
    operationalist
    constructormodemap
    modemaps            -- this should not be needed. eliminate it.
    object              -- the name of the object file to load for this con.
    constructorcategory -- note that this info is the cadar of the
          constructormodemap for domains and packages so it is stored
          as NIL for them. it is valid for categories.
    niladic             -- t or nil directly
    unused
    cosig                -- kept directly
    constructorkind      -- kept directly
    defaultdomain        -- a short list, for %i
    ancestors            -- used to compute new category updates
  )
```

[make-database p**??**]
[DaaseName p1082]
[$spadroot p178]
[*allconstructors* p1061]
[*interp-stream* p1059]
[*interp-stream-stamp* p1060]

— **defun interpopen** —

```
(defun interpopen ()
 "open the interpreter database and hash the keys"
 (declare (special $spadroot *allconstructors* *interp-stream*
                   *interp-stream-stamp*))
 (let (constructors pos stamp dbstruct)
  (setq *interp-stream* (open (DaaseName "interp.daase" nil)))
  (setq stamp (read *interp-stream*))
  (unless (equal stamp *interp-stream-stamp*)
   (format t "   Re-reading interp.daase")
   (setq *interp-stream-stamp* stamp)
   (setq pos (car stamp))
   (file-position *interp-stream* pos)
   (setq constructors (read *interp-stream*))
   (dolist (item constructors)
    (setq *allconstructors* (adjoin (first item) *allconstructors*))
    (setq dbstruct (make-database))
    (setf (get (car item) 'database) dbstruct)
    (setf (database-operationalist dbstruct) (second item))
    (setf (database-constructormodemap dbstruct) (third item))
    (setf (database-modemaps dbstruct) (fourth item))
    (setf (database-object dbstruct) (fifth item))
    (setf (database-constructorcategory dbstruct) (sixth item))
    (setf (database-niladic dbstruct) (seventh item))
    (setf (database-abbreviation dbstruct) (eighth item))
    (setf (get (eighth item) 'abbreviationfor) (first item)) ;invert
    (setf (database-cosig dbstruct) (ninth item))
    (setf (database-constructorkind dbstruct) (tenth item))
    (setf (database-ancestors dbstruct) (nth 11 item))))
  (format t "~&")))
```

———-

This is an initialization function for the constructor database it sets up 2 hash tables, opens the database and hashes the index values.

There is a slight asymmetry in this code. The sourcefile information for system files is only the filename and extension. For user files it contains the full pathname. when the database is first opened the sourcefile slot contains system names. The lookup function has to prefix the "$spadroot" information if the directory-namestring is null (we don't know the real root at database build time).

An object-hash table is set up to look up nrlib and ao information. this slot is empty until a user does a)library call. We remember the location of the nrlib or ao file for the users local library at that time. A `NIL` result from this probe means that the library is in the system-specified place. When we get into multiple library locations this will also contain system files.

36.1.22 defun Open the browse database

Format of an entry in browse.daase:

```
( constructorname
    sourcefile
```

```
    constructorform
    documentation
    attributes
    predicates
)
```

[$spadroot p178]
[*allconstructors* p1061]
[*browse-stream* p1060]
[*browse-stream-stamp* p1060]

— **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 contructor ~a~%" item)
     (format t "that is not in the interp.daase file. we cannot~%")
     (format t "get the database structure for this constructor and~%")
     (warn "will create a new one~%")
     (setf (get (car item) 'database) (setq dbstruct (make-database)))
     (setq *allconstructors* (adjoin item *allconstructors*)))
    (setf (database-sourcefile dbstruct) (second item))
    (setf (database-constructorform dbstruct) (third item))
    (setf (database-documentation dbstruct) (fourth item))
    (setf (database-attributes dbstruct) (fifth item))
    (setf (database-predicates dbstruct) (sixth item))
    (setf (database-parents dbstruct) (seventh item))))
  (format t "~&")))
```

———-

36.1.23 defun Open the category database

[$spadroot p178]
[*hasCategory-hash* p1059]
[*category-stream* p1061]
[*category-stream-stamp* p1061]

— **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 "~&")))
```

———-

36.1.24 defun Open the operations database

[$spadroot p178]
[*operation-hash* p1059]
[*operation-stream* p1060]
[*operation-stream-stamp* p1060]

— **defun operationopen** —

```
(defun operationopen ()
 "read operation database and hash the keys"
 (declare (special $spadroot *operation-hash* *operation-stream*
                   *operation-stream-stamp*))
 (let (operations pos stamp)
  (setq *operation-stream* (open (DaaseName "operation.daase" nil)))
  (setq stamp (read *operation-stream*))
  (unless (equal stamp *operation-stream-stamp*)
   (format t "   Re-reading operation.daase")
   (setq *operation-stream-stamp* stamp)
   (setq pos (car stamp))
   (file-position *operation-stream* pos)
   (setq operations (read *operation-stream*))
   (dolist (item operations)
    (setf (gethash (car item) *operation-hash*) (cdr item))))
  (format t "~&")))
```

———-

36.1.25 defun Add operations from newly compiled code

[getdatabase p1070]
[*operation-hash* p1059]

— defun addoperations —

```
(defun addoperations (constructor oldmaps)
 "add ops from a )library domain to *operation-hash*"
 (declare (special *operation-hash*))
 (dolist (map oldmaps) ; out with the old
  (let (oldop op)
   (setq op (car map))
   (setq oldop (getdatabase op 'operation))
   (setq oldop (lisp::delete (cdr map) oldop :test #'equal))
   (setf (gethash op *operation-hash*) oldop)))
 (dolist (map (getdatabase constructor 'modemaps)) ; in with the new
  (let (op newmap)
   (setq op (car map))
   (setq newmap (getdatabase op 'operation))
   (setf (gethash op *operation-hash*) (cons (cdr map) newmap)))))
```

———-

36.1.26 defun Show all database attributes of a constructor

[getdatabase p1070]

— defun showdatabase —

```
(defun showdatabase (constructor)
 (format t "~&~a: ~a~%" 'constructorkind
  (getdatabase constructor 'constructorkind))
 (format t "~&~a: ~a~%" 'cosig
  (getdatabase constructor 'cosig))
 (format t "~&~a: ~a~%" 'operation
  (getdatabase constructor 'operation))
 (format t "~&~a: ~%" 'constructormodemap)
  (pprint (getdatabase constructor 'constructormodemap))
 (format t "~&~a: ~%" 'constructorcategory)
  (pprint (getdatabase constructor 'constructorcategory))
 (format t "~&~a: ~%" 'operationalist)
  (pprint (getdatabase constructor 'operationalist))
 (format t "~&~a: ~%" 'modemaps)
  (pprint (getdatabase constructor 'modemaps))
 (format t "~&~a: ~a~%" 'hascategory
  (getdatabase constructor 'hascategory))
 (format t "~&~a: ~a~%" 'object
  (getdatabase constructor 'object))
 (format t "~&~a: ~a~%" 'niladic
  (getdatabase constructor 'niladic))
 (format t "~&~a: ~a~%" 'abbreviation
  (getdatabase constructor 'abbreviation))
```

```
(format t "~&~a: ~a~%" 'constructor?
 (getdatabase constructor 'constructor?))
(format t "~&~a: ~a~%" 'constructor
 (getdatabase constructor 'constructor))
(format t "~&~a: ~a~%" 'defaultdomain
 (getdatabase constructor 'defaultdomain))
(format t "~&~a: ~a~%" 'ancestors
 (getdatabase constructor 'ancestors))
(format t "~&~a: ~a~%" 'sourcefile
 (getdatabase constructor 'sourcefile))
(format t "~&~a: ~a~%" 'constructorform
 (getdatabase constructor 'constructorform))
(format t "~&~a: ~a~%" 'constructorargs
 (getdatabase constructor 'constructorargs))
(format t "~&~a: ~a~%" 'attributes
 (getdatabase constructor 'attributes))
(format t "~&~a: ~%" 'predicates)
 (pprint (getdatabase constructor 'predicates))
(format t "~&~a: ~a~%" 'documentation
 (getdatabase constructor 'documentation))
(format t "~&~a: ~a~%" 'parents
 (getdatabase constructor 'parents)))
```

———-

36.1.27 defun Set a value for a constructor key in the database

[make-database p**??**]

— defun setdatabase —

```
(defun setdatabase (constructor key value)
 (let (struct)
  (when (symbolp constructor)
   (unless (setq struct (get constructor 'database))
    (setq struct (make-database))
    (setf (get constructor 'database) struct))
   (case key
    (abbreviation
     (setf (database-abbreviation struct) value)
     (when (symbolp value)
      (setf (get value 'abbreviationfor) constructor)))
    (constructorkind
     (setf (database-constructorkind struct) value))))))
```

———-

36.1.28 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)))))
```

———-

36.1.29 defun Get constructor information for a database key

[warn p**??**]
[$spadroot p178]
[*miss* p1059]
[*hascategory-hash* p**??**]
[*operation-hash* p1059]
[*browse-stream* p1060]
[*defaultdomain-list* p1058]
[*interp-stream* p1059]
[*category-stream* p1061]
[*hasCategory-hash* p1059]
[*operation-stream* p1060]

— **defun getdatabase** —

```
(defun getdatabase (constructor key)
 (declare (special $spadroot) (special *miss*))
 (when (eq *miss* t) (format t "getdatabase call: ~20a ~a~%" constructor key))
 (let (data table stream ignore struct)
  (declare (ignore ignore)
           (special *hascategory-hash* *operation-hash*
                    *browse-stream* *defaultdomain-list* *interp-stream*
                    *category-stream* *hasCategory-hash* *operation-stream*))
  (when (or (symbolp constructor)
            (and (eq key 'hascategory) (consp constructor)))
  (case key
; note that abbreviation, constructorkind and cosig are heavy hitters
; thus they occur first in the list of things to check
   (abbreviation
    (setq stream *interp-stream*)
    (when (setq struct (get constructor 'database))
      (setq data (database-abbreviation struct))))
   (constructorkind
    (setq stream *interp-stream*)
    (when (setq struct (get constructor 'database))
     (setq data (database-constructorkind struct))))
   (cosig
    (setq stream *interp-stream*)
    (when (setq struct (get constructor 'database))
     (setq data (database-cosig struct))))
   (operation
    (setq stream *operation-stream*)
    (setq data (gethash constructor *operation-hash*)))
```

```
(constructormodemap
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructormodemap struct))))
(constructorcategory
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructorcategory struct))
  (when (null data) ;domain or package then subfield of constructormodemap
   (setq data (cadar (getdatabase constructor 'constructormodemap))))))
(operationalist
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-operationalist struct))))
(modemaps
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-modemaps struct))))
(hascategory
 (setq table  *hasCategory-hash*)
 (setq stream *category-stream*)
 (setq data (gethash constructor table)))
(object
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-object struct))))
(niladic
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-niladic struct))))
(constructor?
 (when (setq struct (get constructor 'database))
   (setq data (when (database-operationalist struct) t))))
(superdomain ; only 2 superdomains in the world
 (case constructor
  (|NonNegativeInteger|
   (setq data '((|Integer|) (IF (< |#1| 0) |false| |true|))))
  (|PositiveInteger|
   (setq data '((|NonNegativeInteger|) (< 0 |#1|))))))
(constructor
 (when (setq data (get constructor 'abbreviationfor))))
(defaultdomain
 (setq data (cadr (assoc constructor *defaultdomain-list*))))
(ancestors
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-ancestors struct))))
(sourcefile
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-sourcefile struct))))
(constructorform
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
```

```
    (setq data (database-constructorform struct))))
  (constructorargs
   (setq data (cdr (getdatabase constructor 'constructorform))))
  (attributes
   (setq stream *browse-stream*)
   (when (setq struct (get constructor 'database))
    (setq data (database-attributes struct))))
  (predicates
   (setq stream *browse-stream*)
   (when (setq struct (get constructor 'database))
    (setq data (database-predicates struct))))
  (documentation
   (setq stream *browse-stream*)
   (when (setq struct (get constructor 'database))
    (setq data (database-documentation struct))))
  (parents
   (setq stream *browse-stream*)
   (when (setq struct (get constructor 'database))
    (setq data (database-parents struct))))
  (users
   (setq stream *browse-stream*)
   (when (setq struct (get constructor 'database))
    (setq data (database-users struct))))
  (dependents
   (setq stream *browse-stream*)
   (when (setq struct (get constructor 'database))
    (setq data (database-dependents struct))))
  (otherwise  (warn "~%(GETDATABASE ~a ~a) failed~%" constructor key)))
 (when (numberp data)      ;fetch the real data
  (when *miss* (format t "getdatabase miss: ~20a ~a~%" constructor key))
  (file-position stream data)
  (setq data (read stream))
  (case key ; cache the result of the database read
   (operation           (setf (gethash constructor *operation-hash*) data))
   (hascategory         (setf (gethash constructor *hascategory-hash*) data))
   (constructorkind     (setf (database-constructorkind struct) data))
   (cosig               (setf (database-cosig struct) data))
   (constructormodemap  (setf (database-constructormodemap struct) data))
   (constructorcategory (setf (database-constructorcategory struct) data))
   (operationalist      (setf (database-operationalist struct) data))
   (modemaps            (setf (database-modemaps struct) data))
   (object              (setf (database-object struct) data))
   (niladic             (setf (database-niladic struct) data))
   (abbreviation        (setf (database-abbreviation struct) data))
   (constructor         (setf (database-constructor struct) data))
   (ancestors           (setf (database-ancestors struct) data))
   (constructorform     (setf (database-constructorform struct) data))
   (attributes          (setf (database-attributes struct) data))
   (predicates          (setf (database-predicates struct) data))
   (documentation       (setf (database-documentation struct) data))
   (parents             (setf (database-parents struct) data))
   (users               (setf (database-users struct) data))
   (dependents          (setf (database-dependents struct) data))
   (sourcefile          (setf (database-sourcefile struct) data))))
```

```
  (case key ; fixup the special cases
   (sourcefile
    (when (and data (string= (directory-namestring data) "")
               (string= (pathname-type data) "spad"))
     (setq data
      (concatenate 'string $spadroot "/../../src/algebra/" data))))
   (object                                ; fix up system object pathname
    (if (consp data)
      (setq data
            (if (string= (directory-namestring (car data)) "")
                (concatenate 'string $spadroot "/algebra/" (car data) ".o")
              (car data)))
     (when (and data (string= (directory-namestring data) ""))
      (setq data (concatenate 'string $spadroot "/algebra/" data ".o")))))))
 data))
```

———-

36.1.30 defun The)library top level command

[localdatabase p1073]
[extendLocalLibdb p??]
[serverReadLine is-console (vol9)]
[tersyscommand p704]
[$newConlist p??]
[$options p63]

— **defun library** —

```
(defun |library| (args)
 (let (original-directory)
 (declare (special |$options| |$newConlist|))
  (setq original-directory (get-current-directory))
  (setq |$newConlist| nil)
  (localdatabase args |$options|)
  (|extendLocalLibdb| |$newConlist|)
  (system::chdir original-directory)
  (tersyscommand)))
```

———-

36.1.31 defun Read a local filename and update the hash tables

The localdatabase function tries to find files in the order of: nrlib/index.kaf [sayKeyedMsg p39]
[localnrlib p1075]
[$forceDatabaseUpdate p??]
[$ConstructorCache p??]
[*index-filename* p??]

— defun localdatabase —

```
(defun localdatabase (filelist options &optional (make-database? nil))
 "read a local filename and update the hash tables"
 (labels (
  (processOptions (options)
   (let (only dir noexpose)
    (when (setq only (assoc '|only| options))
     (setq options (lisp::delete only options :test #'equal))
     (setq only (cdr only)))
    (when (setq dir (assoc '|dir| options))
     (setq options (lisp::delete dir options :test #'equal))
     (setq dir (second dir))
     (when (null dir)
      (|sayKeyedMsg|
       "Ignoring )dir because an explicit directory was not given after )dir."
        nil)))
    (when (setq noexpose (assoc '|noexpose| options))
     (setq options (lisp::delete noexpose options :test #'equal))
     (setq noexpose 't) )
    (when options
     (format t "   Ignoring unknown )library option: ~a~%" options))
    (values only dir noexpose)))
  (processDir (dirarg thisdir)
   (let (allfiles)
    (declare (special vmlisp::*index-filename*))
    (system::chdir (string dirarg))
    (setq allfiles (directory "*"))
    (system::chdir thisdir)
     (mapcan #'(lambda (f)
      (when (string-equal (pathname-type f) "nrlib")
       (list (concatenate 'string (namestring f) "/"
                          vmlisp::*index-filename*)))) allfiles))))
 (let (thisdir nrlibs object only dir key  (|$forceDatabaseUpdate| t) noexpose)
  (declare (special |$forceDatabaseUpdate| vmlisp::*index-filename*
                    |$ConstructorCache|))
  (setq thisdir (namestring (truename ".")))
  (setq noexpose nil)
  (multiple-value-setq (only dir noexpose) (processOptions options))
     ;don't force exposure during database build
  (if make-database? (setq noexpose t))
  (when dir (setq nrlibs (processDir dir thisdir)))
  (dolist (file filelist)
   (let* ((file (string file))
          (filename (pathname-name file))
          (namedir (directory-namestring file)))
    (unless namedir (setq thisdir (concatenate 'string thisdir "/")))
    (cond
     ((setq file (probe-file
       (concatenate 'string namedir filename ".nrlib/"
                    vmlisp::*index-filename*)))
      (push (namestring file) nrlibs))
     ('else (format t "   )library cannot find the file ~a.~%" filename)))))
  (dolist (file (nreverse nrlibs))
   (setq key (pathname-name (first (last (pathname-directory file)))))
```

```
   (setq object (concatenate 'string (directory-namestring file) "code"))
   (localnrlib key file object make-database? noexpose))
  (clrhash |$ConstructorCache|))))
```

———-

36.1.32 defun Update the database from an nrlib index.kaf file

[getdatabase p1070]
[make-database p**??**]
[addoperations p1068]
[sublislis p**??**]
[updateDatabase p1077]
[installConstructor p**??**]
[updateCategoryTable p**??**]
[categoryForm? p**??**]
[setExposeAddConstr p142]
[startTimingProcess p**??**]
[loadLibNoUpdate p1095]
[sayKeyedMsg p39]
[$FormalMapVariableList p15]
[*allOperations* p1061]
[*allconstructors* p1061]

— **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 (string object)))))))
 (setf (database-abbreviation dbstruct) abbrev)
 (setf (get abbrev 'abbreviationfor) key)
 (setf (database-operationalist dbstruct) nil)
 (setf (database-operationalist dbstruct)
  (fetchdata alist in "operationAlist"))
 (setf (database-constructormodemap dbstruct)
  (fetchdata alist in "constructorModemap"))
 (setf (database-modemaps dbstruct) (fetchdata alist in "modemaps"))
 (setf (database-sourcefile dbstruct) (fetchdata alist in "sourceFile"))
 (when make-database?
  (setf (database-sourcefile dbstruct)
   (file-namestring  (database-sourcefile dbstruct))))
 (setf (database-constructorkind dbstruct)
  (setq kind (fetchdata alist in "constructorKind")))
 (setf (database-constructorcategory dbstruct)
  (fetchdata alist in "constructorCategory"))
 (setf (database-documentation dbstruct)
  (fetchdata alist in "documentation"))
 (setf (database-attributes dbstruct)
  (fetchdata alist in "attributes"))
 (setf (database-predicates dbstruct)
  (fetchdata alist in "predicates"))
 (setf (database-niladic dbstruct)
  (when (fetchdata alist in "NILADIC") t))
(addoperations key oldmaps)
(unless make-database?
 (if (eq kind '|category|)
  (setf (database-ancestors dbstruct)
   (sublislis |$FormalMapVariableList|
    (cdr constructorform) (fetchdata alist in "ancestors"))))
 (|updateDatabase| key key systemdir?) ;makes many hashtables???
 (|installConstructor| key kind) ;used to be key cname ...
 (|updateCategoryTable| key kind)
 (if |$InteractiveMode| (setq |$CategoryFrame| |$EmptyEnvironment|)))
(setf (database-cosig dbstruct)
  (cons nil (mapcar #'|categoryForm?|
   (cddar (database-constructormodemap dbstruct)))))
(remprop key 'loaded)
(if (null noexpose) (|setExposeAddConstr| (cons key nil)))
(setf (symbol-function key) ; sets the autoload property for cname
  #'(lambda (&rest args)
   (unless (get key 'loaded)
    (|startTimingProcess| '|load|)
    (|loadLibNoUpdate| key key object)) ; used to be cname key
   (apply key args)))
(|sayKeyedMsg| "%1 will be automatically loaded when needed from %2"
 (list key object))))))
```

———-

36.1.33 defun updateDatabase

For now in NRUNTIME do database update only if forced [constructor? p**??**]
[clearClams p**??**]
[clearAllSlams p**??**]
[$forceDatabaseUpdate p**??**]

— **defun updateDatabase** —

```
(defun |updateDatabase| (fname cname systemdirp)
 (declare (ignore fname))
 (declare (special |$forceDatabaseUpdate|))
  (when |$forceDatabaseUpdate|
   (when (|constructor?| cname)
    (|clearClams|)
    (|clearAllSlams| nil)
    (when (getl cname 'loaded) (|clearConstructorCaches|)))
   (when (or |$forceDatabaseUpdate| (null systemdirp))
    (|clearClams|)
    (|clearAllSlams| nil))))
```

————-

36.1.34 defvar *sourcefiles*

— **initvars** —

```
(defvar *sourcefiles* nil)
```

————-

36.1.35 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.

[localdatabase p1073]
[getenviron p291]
[browserAutoloadOnceTrigger p??]
[mkTopicHashTable p??]
[buildLibdb p??]
[dbSplitLibdb p??]
[mkUsersHashTable p??]
[saveUsersHashTable p1081]
[mkDependentsHashTable p??]
[saveDependentsHashTable p1081]
[write-browsedb p1088]
[write-operationdb p1089]
[write-categorydb p1089]
[allConstructors p1090]
[categoryForm? p??]
[domainsOf p??]
[getConstructorForm p??]
[write-interpdb p1086]
[write-warmdata p1090]
[$constructorList p??]
[*sourcefiles* p1077]
[*allconstructors* p1061]
[*operation-hash* p1059]

— **defun make-databases** —

```
(defun make-databases (ext dirlist)
 (labels (
  (build-name-to-pamphlet-hash (dir)
   (let ((ht (make-hash-table)) (eof '(done)) point mark abbrev name file ns)
    (dolist (fn (directory dir))
     (when (and (string= (pathname-type fn) "pamphlet")
            (or (string= (pathname-name fn) "bookvol10.2")   ; category
                (string= (pathname-name fn) "bookvol10.3")   ; domain
                (string= (pathname-name fn) "bookvol10.4")   ; package
                (string= (pathname-name fn) "bookvol10.5"))) ; numerics
      (with-open-file (f fn)
       (do ((ln (read-line f nil eof) (read-line f nil eof))
            (line 0 (incf line)))
           ((eq ln eof))
```

```
      (when (and (setq mark (search ")abb" ln)) (= mark 0))
        (setq mark (position #\space ln :from-end t))
        (setq name (intern (string-trim '(#\space) (subseq ln mark))))
        (cond
          ((setq mark (search "domain" ln)) (setq mark (+ mark 7)))
          ((setq mark (search "package" ln)) (setq mark (+ mark 8)))
          ((setq mark (search "category" ln)) (setq mark (+ mark 9))))
        (setq point (position #\space ln :start (+ mark 1)))
        (setq abbrev
         (intern (string-trim '(#\space) (subseq ln mark point))))
        (setq ns (namestring fn))
        (setq mark (position #\/ ns :from-end t))
        (setq file (subseq ns (+ mark 1)))
        (setf (gethash abbrev ht) file)
        (setf (gethash (format nil "~a-line" abbrev) ht) line)
        (setf (gethash name ht) file)
        (setf (gethash (format nil "~a-line" name) ht) line))))))
    ht))
   ;; these are types which have no library object associated with them.
   ;; we store some constructed data to make them perform like library
   ;; objects, the *operationalist-hash* key entry is used by allConstructors
 (withSpecialConstructors ()
  (declare (special *allconstructors*))
  ; note: if item is not in *operationalist-hash* it will not be written
  ; Category
  (setf (get '|Category| 'database)
    (make-database :operationalist nil :niladic t))
  (push '|Category| *allconstructors*)
  ; UNION
  (setf (get '|Union| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Union| *allconstructors*)
  ; RECORD
  (setf (get '|Record| 'database)
   (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Record| *allconstructors*)
  ; MAPPING
  (setf (get '|Mapping| 'database)
   (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Mapping| *allconstructors*)
  ; ENUMERATION
  (setf (get '|Enumeration| 'database)
   (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Enumeration| *allconstructors*)
  )
 (final-name (root)
   (format nil "~a.daase~a" root ext))
 )
(let (d)
 (declare (special |$constructorList| *sourcefiles*
                   *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 (getenviron "AXIOM") "/autoload/topics"))  ;; hack
  (|browserAutoloadOnceTrigger|)
  (|mkTopicHashTable|)
  (setq |$constructorList| nil) ;; affects buildLibdb
  (setq *sourcefiles* (build-name-to-pamphlet-hash
    (concatenate 'string (getenviron "AXIOM") "/../../books/*.pamphlet")))
  (|buildLibdb|)
  (|dbSplitLibdb|)
; (|dbAugmentConstructorDataTable|)
  (|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")))))
```

———-

36.1.36 defun saveDependentsHashTable

[erase p??]
[writeLib1 p??]
[msort p??]
[hkeys p1105]
[rwrite p813]
[hget p1105]
[rshut p??]
[$depTb p??]
[$erase p??]

— **defun saveDependentsHashTable** —

```
(defun |saveDependentsHashTable| ()
 (let (stream)
 (declare (special |$depTb| $erase))
  ($erase '|dependents| 'database '|a|)
  (setq stream (|writeLib1| '|dependents| 'database '|a|))
  (dolist (k (msort (hkeys |$depTb|)))
    (|rwrite| k (hget |$depTb| k) stream))
  (rshut stream)))
```

———-

36.1.37 defun saveUsersHashTable

[erase p??]
[writeLib1 p??]
[msort p??]
[hkeys p1105]
[rwrite p813]
[hget p1105]
[rshut p??]
[$erase p??]
[$usersTb p??]

— **defun saveUsersHashTable** —

```
(defun |saveUsersHashTable| ()
 (let (stream)
 (declare (special |$usersTb| $erase))
  ($erase '|users| 'database '|a|)
  (setq stream (|writeLib1| '|users| 'database '|a|))
  (dolist (k (msort (hkeys |$usersTb|)))
    (|rwrite| k (HGET |$usersTb| k) stream))
  (rshut stream)))
```

———-

36.1.38 defun Construct the proper database full pathname

[getenviron p291]
[$spadroot p178]

— **defun DaaseName** —

```
(defun DaaseName (name erase?)
 (let (daase filename)
  (declare (special $spadroot))
  (if (setq daase (getenviron "DAASE"))
   (progn
    (setq filename  (concatenate 'string daase "/algebra/" name))
    (format t "   Using local database ~a.." filename))
   (setq filename (concatenate 'string $spadroot "/algebra/" name)))
  (when erase? (system::system (concatenate 'string "rm -f " filename)))
  filename))
```

———-

36.1.39 Building the interp.daase from hash tables

```
 format of an entry in interp.daase:
  (constructor-name
    operationalist
    constructormodemap
    modemaps             -- this should not be needed. eliminate it.
    object               -- the name of the object file to load for this con.
    constructorcategory -- note that this info is the cadar of the
          constructormodemap for domains and packages so it is stored
          as NIL for them. it is valid for categories.
    niladic              -- t or nil directly
    unused
    cosig                -- kept directly
    constructorkind      -- kept directly
    defaultdomain        -- a short list, for %i
    ancestors            -- used to compute new category updates
  )
```

Here I'll try to outline the interp database write procedure

```
(defun write-interpdb ()
 "build interp.daase from hash tables"
 (declare (special $spadroot *ancestors-hash*))
 (let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
       concategory categorypos kind niladic cosig abbrev defaultdomain
       ancestors ancestorspos out)
  (declare (special *print-pretty*))
  (print "building interp.daase")

; 1. We open the file we're going to create

  (setq out (open "interp.build" :direction :output))

; 2. We reserve some space at the top of the file for the key-time pair
;    We will overwrite these spaces just before we close the file.

  (princ "                              " out)

; 3. Make sure we write it out
  (finish-output out)

; 4. For every constructor in the system we write the parts:

  (dolist (constructor (|allConstructors|))
   (let (struct)

; 4a. Each constructor has a property list. A property list is a list
;     of (key . value) pairs. The property we want is called 'database
;     so there is a ('database . something) in the property list

    (setq struct (get constructor 'database))

; 5 We write the "operationsalist"
; 5a. We remember the current file position before we write
;     We need this information so we can seek to this position on read

    (setq opalistpos (file-position out))

; 5b. We get the "operationalist" 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

    (setq obj
     (pathname-name
       (first (last (pathname-directory (database-object struct))))))

; 9. We write the "constructorcategory", if it is a category, else nil
; 9a. Get the constructorcategory

    (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
;     constructorcatagory will not be a number but will be nil.

    (if concategory  ; if category then write data else write nil
     (progn
      (setq categorypos (file-position out))
      (print concategory out)
      (finish-output out))
     (setq categorypos nil))

; 10. We get a set of properties which are kept as "immediate" data
;     This means that the key table will hold this data directly
;     rather than as a byte index into the file.
; 10a. niladic data

    (setq niladic (database-niladic struct))

; 10b. abbreviation data (e.g. POLY for polynomial)

    (setq abbrev (database-abbreviation struct))

; 10c. cosig data

    (setq cosig (database-cosig struct))
```

```
; 10d. kind data

    (setq kind (database-constructorkind struct))

; 10e. defaultdomain data

    (setq defaultdomain (database-defaultdomain struct))

; 11. The ancestor data might exist. If it does we fetch it
;     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 immediatly
;     save the image. The saved image already has the data since we
;     just wrote it out. If the *interp-stream-stamp* and the database
;     time stamp differ we "reread" the database on startup. Actually
;     we just open the database and fetch as needed. You can see fetches
;     by setting the *miss* variable non-nil.

  (print (cons masterpos (get-universal-time)) out)

; 19. We make sure we write it.

  (finish-output out)

; 20 And we are done

  (close out)))
```

36.1.40 defun Write the interp database

[$spadroot p178]
[*ancestors-hash* p**??**]
[*print-pretty* p**??**]

— **defun write-interpdb** —

```
(defun write-interpdb ()
 "build interp.daase from hash tables"
 (declare (special $spadroot *ancestors-hash*))
 (let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
       concategory categorypos kind niladic cosig abbrev defaultdomain
       ancestors ancestorspos out)
  (declare (special *print-pretty*))
  (print "building interp.daase")
  (setq out (open "interp.build" :direction :output))
  (princ "                              " out)
  (finish-output out)
  (dolist (constructor (|allConstructors|))
   (let (struct)
    (setq struct (get constructor 'database))
    (setq opalistpos (file-position out))
    (print (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)
    (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))
     (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)))
```

———-

36.1.41 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.

36.1.42 defun Write the browse database

[allConstructors p1090]
[$spadroot p178]
[*sourcefiles* p1077]
[*print-pretty* p??]

— **defun write-browsedb** —

```
(defun write-browsedb ()
 "make browse.daase from hash tables"
 (declare (special $spadroot *sourcefiles*))
 (let (master masterpos src formpos docpos attpos predpos *print-pretty* out)
  (declare (special *print-pretty*))
  (print "building browse.daase")
  (setq out (open "browse.build" :direction :output))
  (princ "                              " out)
  (finish-output out)
  (dolist (constructor (|allConstructors|))
   (let (struct)
    (setq struct (get constructor 'database))
     ; sourcefile is small. store the string directly
    (setq src (gethash constructor *sourcefiles*))
    (setq formpos (file-position out))
    (print (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)))
```

———-

36.1.43 Building the category.daase from hash tables

This is a single table of category hash table information, dumped in the database format.

36.1.44 defun Write the category database

[genCategoryTable p??]
[*print-pretty* p??]
[*hasCategory-hash* p1059]

— **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)))
```

————-

36.1.45 Building the operation.daase from hash tables

This is a single table of operations hash table information, dumped in the database format.

36.1.46 defun Write the operations database

[*operation-hash* p1059]

— **defun write-operationdb** —

```
(defun write-operationdb ()
 (let (pos master out)
  (declare (special leaves *operation-hash*))
  (setq out (open "operation.build" :direction :output))
  (princ "                              " out)
  (finish-output out)
```

```
  (maphash #'(lambda (key value)
   (setq pos (file-position out))
   (print 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)))
```

———-

36.1.47 Database support operations

36.1.48 defun Data preloaded into the image at build time

[$topicHash p??]

— **defun write-warmdata** —

```
(defun write-warmdata ()
 "write out information to be loaded into the image at build time"
 (declare (special |$topicHash|))
 (with-open-file (out "warm.data" :direction :output)
  (format out "(in-package \"BOOT\")~%")
  (format out "(setq |$topicHash| (make-hash-table))~%")
  (maphash #'(lambda (k v)
   (format out "(setf (gethash '|~a| |$topicHash|) ~a)~%" k v)) |$topicHash|)))
```

———-

36.1.49 defun Return all constructors

[*allconstructors* p1061]

— **defun allConstructors** —

```
(defun |allConstructors| ()
 (declare (special *allconstructors*))
 *allconstructors*)
```

———-

36.1.50 defun Return all operations

[*allOperations* p1061]
[*operation-hash* p1059]

— **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 37

System Statistics

37.0.51 defun statisticsInitialization

[gbc-time p??]

— **defun statisticsInitialization** —

```
(defun |statisticsInitialization| ()
 "initialize the garbage collection timer"
 #+:akcl (system:gbc-time 0)
 nil)
```

————-

37.1 Lisp Library Handling

37.1.1 defun loadLib

[startTimingProcess p??]
[getdatabase p1070]
[isSystemDirectory p1094]
[pathnameDirectory p1103]
[loadLibNoUpdate p1095]
[sayKeyedMsg p39]
[namestring p1102]
[clearConstructorCache p??]
[updateDatabase p1077]
[installConstructor p??]
[updateCategoryTable p??]
[categoryForm? p??]
[remprop p??]
[stopTimingProcess p??]
[$InteractiveMode p284]
[$printLoadMsgs p897]

[$forceDatabaseUpdate p**??**]
[$CategoryFrame p**??**]

— **defun loadLib** —

```
(defun |loadLib| (cname)
 (let (fullLibName systemdir? update? kind u sig coSig)
 (declare (special |$CategoryFrame| |$InteractiveMode| |$printLoadMsgs|
                   |$forceDatabaseUpdate|))
  (|startTimingProcess| '|load|)
  (when (setq fullLibName (getdatabase cname 'object))
   (setq systemdir? (|isSystemDirectory| (|pathnameDirectory| fullLibName)))
   (setq update? (or |$forceDatabaseUpdate| (null systemdir?)))
   (cond
    ((null update?) (|loadLibNoUpdate| cname cname fullLibName))
    (t
     (setq kind (getdatabase cname 'constructorkind))
     (when |$printLoadMsgs|
       (|sayKeyedMsg| "Loading %1 for %2 %3"
        (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)))))
```

————-

37.1.2 defun isSystemDirectory

[function p**??**]
[$spadroot p178]

— **defun isSystemDirectory** —

```
(defun |isSystemDirectory| (dir)
 (declare (special $spadroot))
 (every #'char= $spadroot dir))
```

————-

37.1.3 defun loadLibNoUpdate

[getdatabase p1070]
[sayKeyedMsg p39]
[toplevel p??]
[clearConstructorCache p??]
[installConstructor p??]
[stopTimingProcess p??]
[$printLoadMsgs p897]
[$InteractiveMode p284]
[$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| "Loading %1 for %2 %3"
     (list (|namestring| fullLibName) kind cname)))
  (cond
   ((equal (catch 'versioncheck (load fullLibName)) (- 1))
     (princ "   wrong library version...recompile ")
     (princ fullLibName)
     (terpri)
     (toplevel))
   (t
     (|clearConstructorCache| cname)
     (|installConstructor| cname kind)
     (setf (get cname 'loaded) fullLibName)
     (when |$InteractiveMode| (setq |$CategoryFrame| (list (list nil))))
     (|stopTimingProcess| '|load|)))
  t))
```

———-

37.1.4 defun loadFunctor

[loadFunctor p1095]
[loadLibIfNotLoaded p??]

— **defun loadFunctor** —

```
(defun |loadFunctor| (u)
 (cond
   ((null (atom u)) (|loadFunctor| (car u)))
   (t
     (|loadLibIfNotLoaded| u)
     u)))
```

Chapter 38

Special Lisp Functions

38.0.5 defun compiledLookup

[isDomain p??]
[NRTevalDomain p1100]

— defun compiledLookup —

```
(defun |compiledLookup| (op sig dollar)
 (setq dollar (|NRTevalDomain| dollar))
 (|basicLookup| op sig dollar dollar))
```

———-

38.0.6 defmacro hashCode?

— defmacro hashCode? 0 —

```
(defmacro |hashCode?| (x)
 `(integerp ,x))
```

———-

38.0.7 defun basicLookup

[spadcall p??]
[hashCode? p1097]
[opIsHasCat p??]
[HasCategory p??]
[hashType p??]
[hashString p??]
[error p??]
[vecp p??]
[isNewWorldDomain p??]

[oldCompLookup p1100]
[lookupInDomainVector p1099]
[$hashSeg p??]
[$hashOpSet p??]
[$hashOpApply p??]
[$hashOp0 p??]
[$hashOp1 p??]

— defun basicLookup —

```
(defun |basicLookup| (op sig domain dollar)
 (let (hashPercent box dispatch lookupFun hashSig val boxval)
 (declare (special |$hashSeg| |$hashOpSet| |$hashOpApply| |$hashOp0|
                   |$hashOp1|))
  (cond
   ((simple-vector-p domain)
    (if (|isNewWorldDomain| domain)
      (|oldCompLookup| op sig domain dollar)
      (|lookupInDomainVector| op sig domain dollar)))
  (t
   (setq hashPercent
    (if (simple-vector-p dollar)
     (|hashType| (elt dollar 0) 0)
     (|hashType| dollar 0)))
   (setq box (cons nil nil))
   (cond
    ((null (simple-vector-p (setq dispatch (car domain))))
      (|error| '|bad domain format|))
    (t
      (setq lookupFun (elt dispatch 3))
      (cond
        ((eql (elt dispatch 0) 0)
          (setq hashSig
           (cond
            ((|hashCode?| sig) sig)
            ((|opIsHasCat| op) (|hashType| sig hashPercent))
            (t (|hashType| (cons '|Mapping| sig) hashPercent))))
          (when (symbolp op)
           (cond
             ((eq op '|Zero|)   (setq op |$hashOp0|))
             ((eq op '|One|)    (setq op |$hashOp1|))
             ((eq op '|elt|)    (setq op |$hashOpApply|))
             ((eq op '|setelt|) (setq op |$hashOpSet|))
             (t                 (setq op (|hashString| (symbol-name op))))))
          (cond
           ((setq val
              (car
                (spadcall (cdr domain) dollar op hashSig box nil lookupFun)))
            val)
           ((|hashCode?| sig) nil)
           ((or (> (|#| sig) 1) (|opIsHasCat| op)) nil)
           ((setq boxval
            (spadcall (cdr dollar) dollar op
                      (|hashType| (car sig) hashPercent)
```

```
                box nil lookupFun))
      (cons #'identity (car boxval)))
     (t nil)))
  ((|opIsHasCat| op) (|HasCategory| domain sig))
  (t
   (when (|hashCode?| op)
    (cond
     ((eql op |$hashOp1|)     (setq op '|One|))
     ((eql op |$hashOp0|)     (setq op '|Zero|))
     ((eql op |$hashOpApply|) (setq op '|elt|))
     ((eql op |$hashOpSet|)   (setq op '|setelt|))
     ((eql op |$hashSeg|)     (setq op 'segment))))
   (cond
    ((and (|hashCode?| sig) (eql sig hashPercent))
      (spadcall
       (car (spadcall (cdr dollar) dollar op '($) box nil lookupFun))))
    (t
     (car
      (spadcall (cdr dollar) dollar op sig box nil lookupFun)))))))))))
```

———-

38.0.8 defun lookupInDomainVector

[basicLookupCheckDefaults p1099]
[spadcall p??]

— **defun lookupInDomainVector** —

```
(defun |lookupInDomainVector| (op sig domain dollar)
 (if (consp domain)
   (|basicLookupCheckDefaults| op sig domain domain)
   (spadcall op sig dollar (elt domain 1))))
```

———-

38.0.9 defun basicLookupCheckDefaults

[vecp p??]
[error p??]
[hashType p??]
[hashCode? p1097]
[hashString p??]
[spadcall p??]
[$lookupDefaults p??]

— **defun basicLookupCheckDefaults** —

```
(defun |basicLookupCheckDefaults| (op sig domain dollar)
 (declare (ignore domain))
 (let (box dispatch lookupFun hashPercent hashSig)
```

```
(declare (special |$lookupDefaults|))
 (setq box (cons nil nil))
 (cond
  ((null (simple-vector-p (setq dispatch (car dollar))))
    (|error| '|bad domain format|))
  (t
    (setq lookupFun (elt dispatch 3))
    (cond
     ((eql (elt dispatch 0) 0)
       (setq hashPercent
        (if (simple-vector-p dollar)
          (|hashType| (elt dollar 0) 0)
          (|hashType| dollar 0)))
       (setq hashSig
        (if (|hashCode?| sig)
         sig
         (|hashType| (cons '|Mapping| sig) hashPercent)))
       (when (symbolp op) (setq op (|hashString| (symbol-name op))))
       (car (spadcall (cdr dollar) dollar op hashSig
                      box (null |$lookupDefaults|) lookupFun)))
     (t
       (car (spadcall (cdr dollar) dollar op sig box
                      (null |$lookupDefaults|) lookupFun)))))))
```

———-

38.0.10 defun oldCompLookup

[lookupInDomainVector p1099]
[$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)))))
```

———-

38.0.11 defun NRTevalDomain

[qcar p**??**]
[eval p**??**]
[evalDomain p993]

— **defun NRTevalDomain** —

```
(defun |NRTevalDomain| (form)
 (if (and (consp form) (eq (qcar form) 'setelt))
  (|eval| form)
  (|evalDomain| form)))
```

————-

38.1 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.

38.1.1 defun put

— **defun put** —

```
(defun put (sym ind val) (setf (get sym ind) val))
```

————-

38.1.2 defmacro while

While the condition is true, repeat the body. When the condition is false, return t.

— **defmacro while** —

```
(defmacro while (condition &rest body)
 `(loop (if (not ,condition) (return t)) ,@body))
```

————-

38.1.3 defmacro whileWithResult

While the condition is true, repeat the body. When the condition is false, return the result form's value.

— **defmacro whileWithResult** —

```
(defmacro whileWithResult (condition result &rest body)
 `(loop (if (not ,condition) ,@result) ,@body))
```

————-

38.2 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.

38.2.1 defun namestring

[pathname p1103]

— **defun namestring** —

```
(defun |namestring| (arg)
 (namestring (|pathname| arg)))
```

———-

38.2.2 defun pathnameName

[pathname p1103]

— **defun pathnameName** —

```
(defun |pathnameName| (arg)
 (pathname-name (|pathname| arg)))
```

———-

38.2.3 defun pathnameType

[pathname p1103]

— **defun pathnameType** —

```
(defun |pathnameType| (arg)
 (pathname-type (|pathname| arg)))
```

———-

38.2.4 defun pathnameTypeId

[upcase p1140]
[object2Identifier p??]
[pathnameType p1102]

— **defun pathnameTypeId** —

```
(defun |pathnameTypeId| (arg)
 (upcase (|object2Identifier| (|pathnameType| arg))))
```

———-

38.2.5 defun mergePathnames

[pathnameName p1102]
[pathnameType p1102]
[pathnameDirectory p1103]

— **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))))
```

———-

38.2.6 defun pathnameDirectory

[pathname p1103]

— **defun pathnameDirectory** —

```
(defun |pathnameDirectory| (arg)
 (namestring (make-pathname :directory (pathname-directory (|pathname| arg)))))
```

———-

38.2.7 defun Axiom pathnames

[pathname p1103]
[make-filename p??]

— **defun pathname** —

```
(defun |pathname| (p)
 (cond
  ((null p) p)
  ((pathnamep p) p)
  ((symbolp p) (pathname (string p)))
  ((null (consp p)) (pathname p))
  (t
   (when (> (|#| p) 2) (setq p (cons (elt p 0) (cons (elt p 1) nil))))
   (pathname (apply #'make-filename p)))))
```

———-

38.2.8 defun makePathname

[pathname p1103]
[object2String p??]

— **defun makePathname** —

```
(defun |makePathname| (name type dir)
 (declare (ignore dir))
 (|pathname| (list (|object2String| name) (|object2String| type))))
```

————-

38.2.9 defun Delete a file

[erase p??]
[pathname p1103]
[$erase p??]

— **defun deleteFile** —

```
(defun |deleteFile| (arg)
 (declare (special $erase))
 ($erase (|pathname| arg)))
```

————-

38.2.10 defun wrap

[lotsof p1105]
[wrap p1104]

— **defun wrap** —

```
(defun wrap (list-of-items wrapper)
 (prog nil
  (cond
   ((or (not (consp list-of-items)) (not wrapper))
    (return list-of-items))
   ((not (consp wrapper))
    (setq wrapper (lotsof wrapper))))
  (return
    (cons
     (if (first wrapper)
      `(,(first wrapper) ,(first list-of-items))
       (first list-of-items))
     (wrap (cdr list-of-items) (cdr wrapper))))))
```

————-

38.2.11 defun lotsof

— defun lotsof —

```
(defun lotsof (&rest items)
  (setq items (copy-list items))
  (nconc items items))
```

————-

38.2.12 defmacro startsId?

— defmacro startsId? —

```
(defmacro |startsId?| (x)
 `(or (alpha-char-p ,x) (member ,x '(#\? #\% #\!) :test #'char=)))
```

————-

38.2.13 defun hput

— defun hput —

```
(defun hput (table key value)
 (setf (gethash key table) value))
```

————-

38.2.14 defmacro hget

— defmacro hget —

```
(defmacro HGET (table key &rest default)
   `(gethash ,key ,table ,@default))
```

————-

38.2.15 defun hkeys

— defun hkeys —

```
(defun hkeys (table)
 (let (keys)
  (maphash
   #'(lambda (key val) (declare (ignore val)) (push key keys)) table)
   keys))
```

———-

38.2.16 defun digitp

[digitp p1106]

— **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)))))
```

———-

38.2.17 defun pname

Note it is important that PNAME returns nil not an error for non-symbols

— **defun pname 0** —

```
(defun pname (x)
  (cond ((symbolp x) (symbol-name x))
        ((characterp x) (string x))
        (t nil)))
```

———-

38.2.18 defun size

— **defun size** —

```
(defun size (l)
 (cond
  ((vectorp l) (length l))
  ((consp l)   (list-length l))
  (t 0)))
```

———-

38.2.19 defun strpos

— **defun strpos** —

```
(defun strpos (what in start dontcare)
 (setq what (string what) in (string in))
 (if dontcare
  (progn
```

```
   (setq dontcare (character dontcare))
   (search what in :start2 start
                   :test #'(lambda (x y) (or (eql x dontcare) (eql x y)))))
  (if (= start 0)
   (search what in)
   (search what in :start2 start))))
```

———-

38.2.20 defun strposl

Note that this assumes "table" is a string.

— **defun strposl** —

```
(defun strposl (table cvec sint item)
 (setq cvec (string cvec))
 (if (not item)
  (position table cvec :test #'(lambda (x y) (position y x)) :start sint)
  (position table cvec :test-not #'(lambda (x y) (position y x)) :start sint)))
```

———-

38.2.21 defmacro identp

— **defmacro identp 0** —

```
(defmacro identp (x)
 (if (atom x)
  `(and ,x (symbolp ,x))
   (let ((xx (gensym)))
    `(let ((,xx ,x))
      (and ,xx (symbolp ,xx))))))
```

———-

38.2.22 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.

38.2.23 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)

38.2.24 defun brightprint

[messageprint p1109]

— **defun brightprint** —

```
(defun brightprint (x)
 (messageprint x))
```

———-

;; ———————— NEW DEFINITION (override in msgdb.boot.pamphlet)

38.2.25 defun brightprint-0

[messageprint-1 p1109]

— **defun brightprint-0** —

```
(defun brightprint-0 (x)
 (messageprint-1 x))
```

———-

38.2.26 defun member

— **defun member 0** —

```
(defun |member| (item sequence)
 (cond
```

```
  ((symbolp item) (member item sequence :test #'eq))
  ((stringp item) (member item sequence :test #'equal))
  ((and (atom item) (not (arrayp item))) (member item sequence))
  (t (member item sequence :test #'equalp))))
```

———-

38.2.27 defun messageprint

— **defun messageprint** —

```
(defun messageprint (x)
 (mapc #'messageprint-1 x))
```

———-

38.2.28 defun messageprint-1

[identp p1107]
[messageprint-1 p1109]
[messageprint-2 p1109]

— **defun messageprint-1** —

```
(defun messageprint-1 (x)
 (cond
  ((or (eq x '|%l|) (equal x "%l")) (terpri))
  ((stringp x) (princ x))
  ((identp x) (princ x))
  ((atom x) (princ x))
  ((princ "(")
   (messageprint-1 (car x))
   (messageprint-2 (cdr x))
   (princ ")"))))
```

———-

38.2.29 defun messageprint-2

[messageprint-1 p1109]
[messageprint-2 p1109]

— **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)))))
```

———-

38.2.30 defun sayBrightly1

[brightprint-0 p1108]
[brightprint p1108]

— defun sayBrightly1 —

```
(defun sayBrightly1 (x *standard-output*)
  (if (atom x)
   (progn (brightprint-0 x) (terpri) (force-output))
   (progn (brightprint   x) (terpri) (force-output))))
```

———-

38.2.31 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))
```

———-

38.2.32 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 39

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 40

Numeric Function Support

40.0.33 defmacro fracpart

fracpart : or rational float → or rational float
— defmacro fracpart 0 —

```
(defmacro fracpart (x)
 `(cadr (multiple-value-list (floor ,x))))
```

———-

40.0.34 defun list to complex conversion

— defun s-to-c 0 —

```
(defun s-to-c (c)
 (complex (car c) (cdr c)))
```

———-

40.0.35 defun complex to list conversion

— defun c-to-s 0 —

```
(defun c-to-s (c)
 (cons (realpart c) (imagpart c)))
```

———-

40.0.36 defun complex to real conversion

— **defun c-to-r** —

```
(defun c-to-r (c)
 (let ((r (realpart c)) (i (imagpart c)))
   (if (or (zerop i) (< (abs i) (* 1.0E-10 (abs r))))
     r
     (|error| "Result is not real.")) ))
```

————-

40.0.37 defmacro FloatError

— **defmacro FloatError 0** —

```
(defmacro FloatError (formatstring arg)
  `(error (format nil ,formatstring ,arg)))
```

————-

40.0.38 defun Rational approximation to $\Gamma(x)$

[phiRatapprox p1114]

— **defun lnrgammaRatapprox** —

```
(defun lnrgammaRatapprox (x)
 "(x-.5)*log(x) - x + log(sqrt(2.0*Pi)) + phiRatapprox(x)"
  (+ (+ (- (* (- x 0.5) (log x)) x)
        (log (sqrt (* 2.0 Pi))))
     (phiRatapprox x)))
```

————-

40.0.39 defun phiRatapprox

[horner p1117]

— **defun phiRatapprox** —

```
(defun phiRatapprox (x)
 (let ((arg (/ 1 (* x x))))
  (/
   (horner
    '(0.066662907040200753 0.64507302912899211
      0.67082783834332138 0.12398282342474939) arg)
   (* x
    (horner
```

```
    '(1.0 7.9966911236636431 8.0995271894897574 1.4877938810969931) arg)))))
```

———-

40.0.40 defun Log approximation to $\Gamma(x)$

[lnrgammaRatapprox p1114]
[gammaRatapprox p1116]

— **defun lnrgamma** —

```
(defun lnrgamma (x)
 (if (< 20 x)
   (lnrgammaRatapprox x)
   (log (gammaRatapprox x))))
```

———-

40.0.41 defun Stirling's approximation to $\Gamma(x)$

[lnrgamma p1115]

— **defun gammaStirling** —

```
(defun gammaStirling (x)
  (exp (lnrgamma x)))
```

———-

40.0.42 defun rgammaImpl

This code implements the real $\Gamma(x)$ function used in DoubleFloatSpecialFunctions. [FloatError p1114]
[gammaStirling p1115]
[gammaRatapprox p1116]

— **defun rgammaImpl** —

```
(defun rgammaImpl (x)
 (when (complexp x)
   (FloatError "Gamma not implemented for complex value ~D" x))
  (if (zerop (- x 1.0))
    1.0
    (if (< 20 x)
     (gammaStirling x)
     (gammaRatapprox x))))
```

———-

40.0.43 defun gammaRatapprox

[gammaRatapprox p1116]
[FloatError p1114]
[gammaRatkernel p1116]

— **defun gammaRatapprox** —

```
(defun gammaRatapprox (x)
 (let (restx intpartx lx prod reducedarg a n result)
  (cond
   ((and (not (< x 2)) (not (< 3 x)))
     (setq result (gammaRatkernel x)))
   ((< 3 x)
     (setq n (- (floor x) 2))
     (setq a (- (- x n) 2))
     (setq reducedarg (+ 2 a))
     (setq prod
      (reduce #'* (loop for i from 0 to (- n 1) collect (+ reducedarg i))))
     (setq result (* prod (gammaRatapprox reducedarg))))
   ((and (< x 2) (< 0 x))
    (setq n (- 2 (floor x)))
    (setq a (- x (floor x)))
    (setq reducedarg (+ 2 a))
    (setq prod (reduce #'* (loop for i from 0 to (- n 1) collect (+ x i))))
    (setq result (/ (gammaRatapprox reducedarg) prod)))
   (t
    (setq lx (multiple-value-list (floor x)))
    (setq intpartx (+ (car lx) 1))
    (setq restx (cadr lx))
    (cond
     ((zerop restx)
       (FloatError "Gamma undefined for non-positive integers: ~D" x)
       (setq result (/ 1.0 (complex 0.0))))
     (t
      (setq result
       (/ Pi
         (* (* (gammaRatapprox (- 1.0 x))
               (expt (- 1.0) (+ intpartx 1)))
            (sin (* restx Pi)))))))))
  result))
```

———-

40.0.44 defun gammaRatkernel

[horner p1117]

— **defun gammaRatkernel** —

```
(defun gammaRatkernel (x)
  (/
   (horner
```

```
    '(0.77807958561330059 6.1260674503360839 48.95434622790993
     222.11239616801177   893.58180452374972 2077.4597938941874
     3786.0105034825724)
     (- x 2))
   (horner
    '(1  -13.400414785781347 50.788475328895402 83.550058667919771
         -867.23098753110298 476.79386050368794 3786.0105034825719)
     (- x 2.0))))
```

———-

40.0.45 defun Horner's rule of polynomial evaluation

— **defun horner 0** —

```
(defun horner (l x)
 (let ((result 0))
  (loop for el in l do (setq result (+ (* x result) el)))
  result))
```

———-

40.0.46 defun Complex implementation of $\Gamma(z)$

[rgammaImpl p1115]
[clngammaImpl p1117]

— **defun cgammaImpl** —

```
(defun cgammaImpl (z)
  (if (zerop (imagpart z))
    (rgammaImpl (realpart z))
    (exp (clngammaImpl (realpart z) (imagpart z) z))))
```

———-

Compute the conjugate of gamma which is the gamma of the conjugate. Map the 2nd and the 4th quadrants to first and third quadrants.

40.0.47 defun Compute the conjugate of $\Gamma(z)$

[clngammacase1 p1118]
[clngammacase23 p1119]

— **defun clngammaImpl** —

```
(defun clngammaImpl (real imag z)
  (cond
   ((< real 0.0)
     (if (< 0.0 imag)
```

```
      (conjugate (clngammacase1 real (- imag) (complex real (- imag))))
      (clngammacase1 real imag z)))
  ((< imag 0.0)
    (conjugate (clngammacase23 real (- imag) (complex real (- imag)))))
  (t
    (clngammacase23 real imag z))))
```

———-

40.0.48 defun $\Gamma(z)$ negative real branch

[PiMinusLogSinPi p1118]
[clngammaImpl p1117]

— **defun clngammacase1** —

```
(defun clngammacase1 (real imag z)
 (- (PiMinusLogSinPi real imag z)
    (clngammaImpl (- 1.0 real) (- imag) (- 1.0 z))))
```

———-

40.0.49 defun PiMinusLogSinPi

[cgammaG p1118]
[logH p1118]

— **defun PiMinusLogSinPi** —

```
(defun PiMinusLogSinPi (real imag z)
 (- (cgammaG real imag) (logH real imag z)))
```

———-

40.0.50 defun cgammaG

— **defun cgammaG 0** —

```
(defun cgammaG (real imag)
  (- (+ (log (* 2 Pi)) (* Pi imag))
     (* (* (complex 0.0 1.0) Pi) (- real 0.5)))))
```

———-

40.0.51 defun logH

— **defun logH 0** —

```
(defun logH (real imag z)
 (declare (ignore z))
 (let (part1 part2 twopiz2 z1bar)
   (setq z1bar (cadr (multiple-value-list (floor real))))
   (setq twopiz2 (* 2.0 (* Pi imag)))
   (setq part2
     (* (exp twopiz2)
       (+ (* 2.0 (expt (sin (* Pi z1bar)) 2))
          (* (sin (* 2.0 (* Pi z1bar))) (complex 0.0 1.0)))))
   ;--- part1 is another way of saying 1 - exp(2*Pi*z1bar)
   (setq part1 (- (* (tanh (* Pi imag)) (+ 1.0 (exp twopiz2)))))
   (log (+ part1 part2))))
```

———-

40.0.52 defun $\Gamma(z)$ positive real branch

[cgammat p1119]
[clngammacase2 p1119]
[clngammacase3 p1120]

— **defun clngammacase23** —

```
(defun clngammacase23 (real imag z)
 (let ((tz2 (cgammat imag)))
  (if (< real tz2)
    (clngammacase2 real imag tz2 z)
    (clngammacase3 z))))
```

———-

40.0.53 defun cgammat

The cgammat is auxiliary "t" function [kuki72a, kuki72b]

— **defun cgammat 0** —

```
(defun cgammat (x)
 (max 0.10000000000000001 (min 10.0 (- (* 10.0 (sqrt 2.0)) (abs x)))))
```

———-

40.0.54 defun $\Gamma(z)$ case 2

[cgammaBernsum p1120]
[cgammaAdjust p1120]
[logS p1120]

— **defun clngammacase2** —

```
(defun clngammacase2 (real imag tz2 z)
 (let (zpn n)
```

```
  (setq n (float (ceiling (- tz2 real))))
  (setq zpn (+ z n))
  (- (+ (- (* (- z 0.5) (log zpn)) zpn) (cgammaBernsum zpn))
     (cgammaAdjust (logS real imag z n zpn)))))
```

———-

40.0.55 defun logS

— **defun logS 0** —

```
(defun logS (real imag z n zpn)
 (let ((sum 0.0))
  (dotimes (k n)
   (if (< (+ real k) (- 5.0 (* 0.60000000000000009 imag)))
    (setq sum (+ sum (log (/ (+ z k) zpn)))))
    (setq sum (+ sum (log (- 1.0 (/ (- n k) zpn))))))
  sum))
```

———-

40.0.56 defun Adjust logS if imaginary part is negative

The logS result should have its imaginary part adjusted by 2π if it is negative. [kuki72a, kuki72b]

— **defun cgammaAdjust 0** —

```
(defun cgammaAdjust (z)
 (if (< (imagpart z) 0.0)
  (+ z (complex 0.0 (* 2.0 Pi)))
  z))
```

———-

40.0.57 defun $\Gamma(z)$ case 3

[cgammaBernsum p1120]

— **defun clngammacase3** —

```
(defun clngammacase3 (z)
 (+ (- (* (- z 0.5) (log z)) z) (cgammaBernsum z)))
```

———-

40.0.58 defun cgammaBernsum

— **defun cgammaBernsum 0** —

```
(defun cgammaBernsum (z)
 (let (l zsquaredinv zterm sum)
  (setq sum (/ (log (* 2.0 Pi)) 2.0))
  (setq zterm z)
  (setq zsquaredinv (/ 1.0 (* z z)))
  (setq l
   (list 0.083333333333333315 (- 0.0027777777777777779)
         7.9365079365079376E-4 (- 5.9523809523809529E-4)
         8.4175084175084182E-4 (- 0.0019175269175269176)
         0.0064102564102564109))
  (loop for el in l do
    (setq zterm (* zterm zsquaredinv))
    (setq sum (+ sum (* el zterm))))
  sum))
```

———-

40.0.59 defun BesselI

[BesselI p1121]
[FloatError p1114]
[besselIback p1122]
[besselIcheb p1123]

— **defun BesselI** —

```
(defun BesselI (v z)
 (let ((b1 15.0))
  (cond
   ((and (zerop z) (floatp v) (not (< v 0.0)))
     (if (zerop v) 1.0 0.0))
   ; Transformations for negative integer orders
   ((and (floatp v) (zerop (fracpart v)) (minusp v))
     (BesselI (- v) z))
   ; Halfplane transformations for Re(z)<0
   ((< (realpart z) 0.0)
     (* (BesselI v (- z)) (expt (- 1.0) v)))
   ; Conjugation for complex order and real argument
   ((and (< (realpart v) 0.0) (null (zerop (imagpart v))) (floatp z))
     (conjugate (BesselI (conjugate v) z)))
   ; We now know that Re(z)>= 0.0 (asymptotic argument case)
   ((< b1 (abs z))
     (FloatError "BesselI not implemented for ~S" (list v z)))
   ((< b1 (abs v))
     (FloatError "BesselI not implemented for ~S" (list v z)))
   ; case of small argument and order
   ((not (< (realpart v) 0.0))
     (besselIback v z))
   ((< (realpart v) 0.0)
     (besselIcheb z v 50))
   (t
     (FloatError "BesselI not implemented for ~S" (list v z))))))
```

———-

40.0.60 defun besselIback

[BesselIBackRecur p1122]
[cgammaImpl p1117]

— **defun besselIback** —

```
(defun besselIback (v z)
 (let (result vp1 tv n m lm rpv ipv)
  (setq ipv (imagpart v))
  (setq rpv (realpart v))
  (setq lm (multiple-value-list (floor rpv)))
  (setq m (car lm))                ; floor of real part of v
  (setq n (* 2 (max 20 (+ m 10)))) ; how large the back recurrence should be
  (setq tv (+ (cadr lm) (- v rpv)); fractional part of real part of v
                                   ; plus imaginary part of v
  (setq vp1 (+ tv 1.0))
  (setq result (BesselIBackRecur v m tv z "I" n))
  (setq result (* (/ result (cgammaImpl vp1)) (expt (/ z 2.0) tv)))))
```

———-

40.0.61 defun Backward recurrence for Bessel functions

Backward recurrence for Bessel functions. Luke (1975), p. 247. works for $-\pi < argz <= \pi$ and $-\pi < argv <= \pi$

— **defun BesselIBackRecur 0** —

```
(defun BesselIBackRecur (largev argm v z type n)
 (declare (ignore largev))
 (let (v1 pn ct1 xm m m1 val w m2 z2 start zero two one)
  (setq one 1.0)
  (setq two 2.0)
  (setq zero 0.0)
  (setq start (expt 10.0 (- 40)))
  (setq z2 (/ two z))
  (setq m2 (+ n 3))
  (setq w (make-array (+ m2 1)))
  (setf (aref w m2) zero)
  (if (string= type "I")
    (setq val one)
    (setq val (- one)))
  (setq m1 (+ n 2))
  (setf (aref w m1) start)
  (setq m (+ n 1))
  (setq xm (float m))
  (setq ct1 (* z2 (+ xm v)))
  ; initialize
  (loop for m from (+ n 1) downto 1 by 1 do
    (setf (aref w m) (+ (* (aref w (+ m 1)) ct1) (* val (aref w (+ m 2))))))
```

```
    (setq ct1 (- ct1 z2)))
  (setq m (+ 1 (floor (/ n 2))))
  (setq m2 (- (+ m m) 1))
  (cond
    ((eql v 0)
     (setq pn (aref w (+ m2 2)))
     (loop for m2 from (- (* 2 m) 1) downto 3 by 2 do
       (setq pn (- (aref w m2) (* val pn))))
     (setq pn (- (aref w 1) (* val (+ pn pn)))))
    ('T (setq v1 (- v one)) (setq xm (float m))
     (setq ct1 (+ (+ v xm) xm))
     (setq pn (* ct1 (aref w (+ m2 2))))
     (loop for m2 from (- (+ m m) 1) downto 3 by 2 do
       (setq ct1 (- ct1 two))
       (setq pn (- (* ct1 (aref w m2)) (* (/ (* val pn) xm) (+ v1 xm))))
       (setq xm (- xm one)))
     (setq pn (- (aref w 1) (* val pn)))))
  (setq m1 (+ n 2))
  (loop for m from 1 to m1 do
    (setf (aref w m) (/ (aref w m) pn)))
  (aref w (+ argm 1))))
```

———-

40.0.62 defun Compute n terms of the chebychev series for f01

[chebf01coefmake p1123]
[chebstarevalarr p1124]
[cgammaImpl p1117]

— **defun besselIcheb** —

```
(defun besselIcheb (z v n)
 (let (result arr sum tmp1 vp1 w arg)
  (setq arg (/ (* z z) 4.0))
  (setq w (* 2.0 arg))
  (setq vp1 (+ v 1.0))
  (setq tmp1 (chebf01coefmake vp1 w n))
  (setq sum (car tmp1))
  (setq arr (cadr tmp1))
  (setq result
    (* (/ (chebstarevalarr arr (/ arg w) n) (cgammaImpl vp1))
       (expt (/ z 2.0) v)))))
```

———-

40.0.63 defun chebf01coefmake

Program transcribed from Fortran,, p. 80 Luke 1977 [Luke77]

where c is a parameter to 0F1, w is a scale factor so that $0 < z/w < 1$, n such that $n+2$ coefficients will be produced stored in an array indexed from 0 to $n+1$. The *arr* array will

be used to store the Cheb. series coefficients

— **defun chebf01coefmake 0** —

```
(defun chebf01coefmake (c w n)
 (let (p sum rho divfac c1 x1 arr ncount z1 a1 a2 a3 n2 n1 start four)
  (setq four 4.0)
  (setq start (expt 10.0 (- 200)))
  (setq n1 (+ n 1))
  (setq n2 (+ n 2))
  (setq a3 0.0)
  (setq a2 0.0)
  (setq a1 start)      ; arbitrary starting value
  (setq z1 (/ four w))
  (setq ncount n1)
  (setq arr (make-array n2))
  (setf (aref arr ncount) start) ; start off
  (setq x1 n2)
  (setq c1 (- 1.0 c))
  (loop for ncount from n downto 0 do
    (setq divfac (/ 1.0 x1))
    (setq x1 (- x1 1.0))
    (setf (aref arr ncount)
      (* x1
       (- (+ (* (+ divfac (* z1 (- x1 c1))) a1)
             (* (+ (/ 1.0 x1) (* z1 (+ (+ x1 c1) 1.0))) a2))
          (* divfac a3))))
    (setq a3 a2)
    (setq a2 a1)
    (setq a1 (aref arr ncount)))
  (setf (aref arr 0) (/ (aref arr 0) 2.0))
  ; compute scale factor
  (setq rho (aref arr 0))
  (setq sum rho)
  (setq p 1.0)
  (loop for i from 1 to n1 do
   (setq rho (- rho (* p (aref arr i))))
   (setq sum (+ sum (aref arr i)))
   (setq p (- p)))
  (loop for l from 0 to n1 do
   (setf (aref arr l) (/ (aref arr l) rho)))
  (setq sum (/ sum rho))
  (list sum arr)))
```

———-

40.0.64 defun chebstarevalarr

Evaluation of the sum(C(n)*T*(n,x))

— **defun chebstarevalarr 0** —

```
(defun chebstarevalarr (coefarr x n)
 (let (c y temp b)
```

```
  (setq b 0)
  (setq temp 0)
  (setq y (* 2 (- (* 2 x) 1)))
  (loop for i from (+ n 1) downto 0 do
    (setq c b)
    (setq b temp)
    (setq temp (+ (- (* y b) c) (aref coefarr i))))
   (- temp (/ (* y b) 2))))
```

———-

40.0.65 defun lncgamma

[clngamma p1145]

— **defun lncgamma** —

```
(defun lncgamma (z)
 (clngammaImpl (realpart z) (imagpart z) z))
```

———-

40.0.66 defun rPsiImpl

[FloatError p1114]
[cotdiffeval p1126]
[rPsiImpl p1125]
[rPsiW p1126]
[rgammaImpl p1115]

— **defun rPsiImpl** —

```
(defun rPsiImpl (n x)
 (let (skipit sign m)
  (cond
   ((not (< 0.0 x))
     (cond
       ((zerop (fracpart x))
         (FloatError "singularity encountered at ~D" x))
       (t
        (setq m (mod n 2))
        (setq sign (expt (- 1) m))
        (if (equal (fracpart x) 0.5)
         (setq skipit 1)
         (setq skipit 0))
        (* sign
          (+ (* (expt Pi (+ n 1))
                (cotdiffeval n (* Pi (- x)) skipit))
             (rPsiImpl n (- 1.0 x)))))))
   ((eql n 0) (- (rPsiW n x)))
   (t
     (* (* (rgammaImpl (float (+ n 1))) (rPsiW n x))
```

```
(expt (- 1) (mod (+ n 1) 2)))))))
```

———-

40.0.67 defun cotdiffeval

Code for computation of derivatives of cot(z), necessary for polygamma reflection formula. If you want to compute n^{th} derivatives of $\cot(\pi * x)$, you have to multiply the result of cotdiffeval by π^n.

Set $skip = 1$ if arg z is known to be an exact multiple of $\pi/2$

— **defun cotdiffeval 0** —

```
(defun cotdiffeval (n z skipit)
 (let (s sq v t2 t1 m a)
  (setq a (make-array (+ n 2)))
  (setf (aref a 0) 0.0)
  (setf (aref a 1) 1.0)
  (loop for i from 2 to n do
    (setf (aref a i) 0.0))
  (loop for r from 1 to n do
    (setq m (mod (+ r 1) 2))
    (loop for k from m to (+ z 1) by 2 do
      (if (< k 1)
        (setq t1 0)
        (setq t1 (- (* (aref a (- k 1)) (- k 1)))))
      (if (< r k)
        (setq t2 0)
        (setq t2 (- (* (aref a (+ k 1)) (+ k 1)))))
      (setf (aref a k) (+ t1 t2))))
  ; evaluate d^N/dX^N cot(z) via Horner-like rule
  (setq v (cot z))
  (setq sq (* v v))
  (setq s (aref a (+ n 1)))
  (loop for i from (- n 1) downto 0 by 2 do
    (setq s (+ (* s sq) (aref a i))))
  (setq m (mod n 2))
  (when (eql m 0) (setq s (* s v)))
  (if (eql skipit 1)
   (if (eql m 0) 0 (aref a 0))
   s)))
```

———-

40.0.68 defun Amos' w function

Amos' w function, with w(0,x) picked to be $-\psi(x)$ for $x > 0$ [PsiAsymptoticOrder p1127]
[PsiAsymptotic p1128]
[PsiBack p1127]

— **defun rPsiW** —

```
(defun rPsiW (n x)
 (let (bign fln c a xmin beta alpha nd result)
  (when (or (not (< 0 x)) (minusp n))
   (error "rPsiW not implemented for negative n or non-positive x"))
  (setq nd 6) ; magic number for number of digits in a word?
  (setq alpha (+ 3.5 (* 0.40000000000000002 nd)))
  (setq beta
   (+ (+ 0.20999999999999999 (* 8.6770000000000001E-6 (- nd 3)))
      (* 6.0380000000000002E-8 (expt (- nd 3) 2))))
  (setq xmin (float (+ (floor (+ alpha (* beta n))) 1)))
  (when (< 0 n)
    (setq a
     (min 0 (* (/ 1.0 (float n)) (log (/ double-float-epsilon (min 1.0 x))))))
    (setq c (exp a))
    (if (not (< (abs a) 0.001))
      (setq fln (* (/ x c) (- 1.0 c)))
      (setq fln (- (/ (* x a) c))))
    (setq bign (+ (floor fln) 1))
    ; Amos says to use alternative series for large order if ordinary
    ; backwards recurrence too expensive
    (when (and (< bign 15) (< (+ 7.0 x) xmin))
      (setq result (PsiAsymptoticOrder n x bign))))
  (when (and (not result) (not (< x xmin)))
    (setq result (PsiAsymptotic n x)))
  (unless result
    ; ordinary case -- use backwards recursion
    (setq result (PsiBack n x xmin)))
  result))
```

———-

40.0.69 defun PsiAsymptoticOrder

— **defun PsiAsymptoticOrder 0** —

```
(defun PsiAsymptoticOrder (n x nterms)
  (loop for k from 0 to nterms
   sum (/ 1.0 (expt (+ x (float k)) (+ n 1)))))
```

———-

40.0.70 defun PsiBack

[PsiIntpart p1129]
[PsiAsymptotic p1128]

— **defun PsiBack** —

```
(defun PsiBack (n x xmin)
 (let (result x0 xintpart)
  (setq xintpart (PsiIntpart x))
```

```
  (setq x0 (- x xintpart)) ; frac part of x
  (setq result (PsiAsymptotic n (+ (+ x0 xmin) 1.0)))
  ; Why not decrement from x?   See Amos p. 498
  (loop for k from xmin downto xintpart by 1 do
    (setq result (+ result (/ 1.0 (expt (+ x0 (float k)) (+ n 1))))))
  result))
```

———-

40.0.71 defvar PsiAsymptoticBern

The n^{th} derivatives of ln gamma for real x, $n = 0, 1, \ldots$

— **initvars** —

```
(defvar |$PsiAsymptoticBern|
  (vector 0.0 0.16666666666666669   (- 0.033333333333333333)
              0.023809523809523812  (- 0.033333333333333333)
              0.07575757575757576   (- 0.2531135531135531)
              1.1666666666666672    (- 7.0921568627450986)
             54.971177944862163   (- 529.12424242424242)
           6192.123188405797    (- 86580.253113553103)
        1425517.166666667           (- 2.729823106781609E7)
              6.015808739006424E8   (- 1.5116315767092161E10)
              4.2961464306116675E11 (- 1.371165520508833E13)
              4.8833231897359325E14 (- 1.9296579341940072E16)
              8.4169304757368269E17 (- 4.0338071854059463E19)))
```

———-

40.0.72 defun PsiAsymptotic

[$PsiAsymptoticBern p1128]
[rgammaImpl p1115]
[PsiEps p1129]

— **defun PsiAsymptotic** —

```
(defun PsiAsymptotic (n x)
 (let (sum factterm xterm xsq xnp1 xn)
 (declare (special |$PsiAsymptoticBern|))
  (setq xn (expt x n))
  (setq xnp1 (* xn x))
  (setq xsq (* x x))
  (setq xterm xsq)
  (setq factterm (/ (/ (rgammaImpl (+ n 2)) 2.0) (rgammaImpl (float (+ n 1)))))
  ; initialize to 1/n!
  (setq sum (/ (* (aref |$PsiAsymptoticBern| 1) factterm) xterm))
  (loop for k from 2 to 22 do
    (setq xterm (* xterm xsq))
    (cond
     ((eql n 0) (setq factterm (/ 1.0 (float (* 2 k)))))
     ((eql n 1) (setq factterm 1))
```

```
    (t
     (setq factterm
      (/ (* (* factterm (float (- (+ (* 2 k) n) 1)))
            (float (- (+ (* 2 k) n) 2)))
         (* (float (* 2 k)) (float (- (* 2 k) 1)))))))
   (setq sum (+ sum (/ (* (aref |$PsiAsymptoticBern| k) factterm) xterm))))
  (+ (+ (PsiEps n x) (/ 1.0 (* 2.0 xnp1))) (* (/ 1.0 xn) sum))))
```

———-

40.0.73 defun PsiEps

— **defun PsiEps 0** —

```
(defun PsiEps (n x)
  (if (eql n 0)
    (- (log x))
    (/ 1.0 (* (float n) (expt x n)))))
```

———-

40.0.74 defun PsiIntpart

— **defun PsiIntpart 0** —

```
(defun PsiIntpart (x)
 (if (minusp x)
  (- (PsiIntpart (- x)))
  (floor x)))
```

———-

40.0.75 defun cPsiImpl

— **defun cPsiImpl** —

```
(defun cPsiImpl (n z)
 (prog (result m bound nterms conjresult y x)
  (return
   (progn
    (setq x (realpart z))
    (setq y (imagpart z))
    (cond ((zerop y) (return (rPsiImpl n x)))) ; call real function if real
    (when (< y 0.0) ; if imagpart(z) negative, take conjugate of conjugate
      (setq conjresult (cPsiImpl n (complex x (- y))))
      (return (complex (realpart conjresult) (- (imagpart conjresult)))))
    (setq nterms 22)
    (setq bound 10.0)
```

```
    (cond
      ((< x 0.0)
        (FloatError "Psi implementation can't compute at ~S " (list n z)))
      ((and (< 0.0 x) (< bound (abs z)))
        (return (PsiXotic n (PsiAsymptotic n z))))
      (t  ; use recursion formula
       (setq m (ceiling (+ (- (sqrt (- (* bound bound) (* y y))) x) 1.0)))
       (setq result (complex 0.0 0.0))
       (loop for k from 0 to (- m 1) do
         (setq result (+ result (/ 1.0 (expt (+ z (float k)) (+ n 1))))))
       (return (PsiXotic n (+ result (PsiAsymptotic n (+ z m)))))))))
```

———-

40.0.76 defun PsiXotic

The n^{th} derivatives of ln gamma for complex z, $n = 0, 1, \ldots$ requires files rpsi (and dependents), floaterrors currently defined only in right half plane until reflection formula works

— **defun PsiXotic** —

```
(defun PsiXotic (n result)
 (* (* (rgammaImpl (float (+ n 1))) (expt (- 1) (mod (+ n 1) 2))) result))
```

———-

40.0.77 defun BesselJ

BesselJ works for complex and real values of v and z

— **defun BesselJ** —

```
(defun BesselJ (v z)
 (let (arr sum t1 vp1 w arg rz rv n b2 b1)
  (setq b1 10)  ; Ad hoc boundaries for approximation
  (setq b2 10)
  (setq n 50)   ; number of terms in Chebychev series.
  (cond
   ; tests for negative integer order
   ((or (and (floatp v) (zerop (fracpart v)) (minuspl v))
        (and (complexp v) (zerop (imagpart v)) (zerop (fracpart (realpart v)))
             (< (realpart v) 0.0)))
     ; odd or even according to v (9.1.5 A&S)
     ; $J_{-n}(z)=(-1)^{n} J_{n}(z)$
     (* (BesselJ (- v) z) (expt (- 1.0) v)))
   ((or (and (floatp z) (minusp z)) (and (complexp z) (< (realpart z) 0.0)))
     ; negative argument (9.1.35 A&S)
     ; $J_{\nu}(z e^{m \pi i}) = e^{m \nu \pi i} J_{\nu}(z)$
     (* (BesselJ v (- z)) (expt (- 1.0) v)))
   ((and (zerop z)
         (or (and (floatp v) (not (< v 0.0)))
             (and (complexp v) (zerop (imagpart v))
                  (not (< (realpart v) 0.0)))))
     ; zero arg, pos. real order
```

```
 (if (zerop v) 1.0 0.0))
(t
  (setq rv (abs v))
  (setq rz (abs z))
  (cond
   ((and (< b1 rz) (< (* b2 rv) rz))   ; asymptotic argument
     (BesselJAsympt v z))
   ((and (< b1 rv) (< (* b2 rz) rv))   ; asymptotic order
     (BesselJAsymptOrder v z))
   ((and (< rz b1) (< rv b1))          ; small order and argument
      (setq arg (- (/ (* z z) 4.0)))
      (setq w (* 2.0 arg))
      (setq vp1 (+ v 1.0))
      (setq t1 (chebf01coefmake vp1 w n))
      (setq sum (car t1))
      (setq arr (cadr t1))
      ; if we get NaNs then half n
      ((lambda ()
         (loop
          (cond
           ((= sum sum) (return nil))
           (t
             (setq n (floor (/ n 2)))
             (setq t1 (chebf01coefmake vp1 w n))
             (setq sum (car t1))
             (setq arr (cadr t1))
             t1)))))
      ; now n is safe, can we increase it (we know that 2*n is bad)?
      (* (/ (chebstarevalarr arr (/ arg w) n) (cgammaImpl vp1))
         (expt (/ z 2.0) v)))
  (t (BesselJRecur v z)))))))
```

———-

Asymptotic functions for large values of z. See Luke [Luke69a] p. 204 where **mu** is $4v^2$, **zsqr** is z^2, and **zfth** is z^4

40.0.78 defun Asymptotic series for BesselJ

Asymptotic series only works when $|v| < |z|$.

— **defun BesselJAsympt** —

```
(defun BesselJAsympt (v z)
 (let (zfth zsqr mu)
   (setq mu (* (* 4.0 v) v))
   (setq zsqr (* z z))
   (setq zfth (* zsqr zsqr))
   (* (* (sqrt (/ 2.0 (* Pi z))) (exp (BesselasymptA mu zsqr zfth)))
      (cos (- (- (BesselasymptB mu z zsqr zfth)
                 (/ (* Pi v) 2.0))
              (/ Pi 4.0))))))
```

———-

40.0.79 defun BesselasymptA

— defun BesselasymptA 0 —

```
(defun BesselasymptA (mu zsqr zfth)
  (* (/ (- mu 1) (* 16.0 zsqr))
     (+ (+ 1 (/ (- mu 13.0) (* 8.0 zsqr)))
        (/ (+ (- (* mu mu) (* 53.0 mu)) 412.0)
           (* 48.0 zfth)))))
```

———-

40.0.80 defun BesselasymptB

— defun BesselasymptB 0 —

```
(defun BesselasymptB (mu z zsqr zfth)
 (let ((musqr (* mu mu)))
   (+ z
      (* (/ (- mu 1.0) (* 8.0 z))
         (+ (+ (+ 1.0 (/ (- mu 25.0) (* 48.0 zsqr)))
               (/ (+ (- musqr (* 114.0 mu)) 1073.0) (* 640.0 zfth)))
            (/ (- (+ (- (* (* 5.0 mu) musqr) (* 1535.0 musqr))
                     (* 54703.0 mu))
                  375733.0)
               (* (* 128.0 zsqr) zfth)))))))
```

———-

40.0.81 defun BesselJRecur

— defun BesselJRecur —

```
(defun BesselJRecur (v z)
 (let (w m so)
  ; boost order. Numerical.Recipes. suggest so:=v+sqrt(n.s.f.^2*v)
  (setq so (* 15.0 z))
  ; reduce order until non-zero
  (loop while (not (zerop (abs (BesselJAsymptOrder so z)))) do
    (setq so (/ so 2.0)))
  (when (< (abs so) (abs z))
    (setq so (+ v (* 18.0 (sqrt v)))))
  (setq m (+ (floor (abs (- so v))) 1))
  (setq w (make-array m))
  (setf (aref w (- m 1)) (BesselJAsymptOrder (- (+ v m) 1) z))
  (setf (aref w (- m 2)) (BesselJAsymptOrder (- (+ v m) 2) z))
  (loop for i from (- m 3) downto 0 by 1 do
    (setf (aref w i)
      (- (/ (* (* 2.0 (+ (+ v i) 1.0)) (aref w (+ i 1))) z)
         (aref w (+ i 2)))))
```

```
  (aref w 0)))
```

———-

40.0.82 defun BesselJAsymptOrder

Asymptotic formula for BesselJ when order is large comes from Debye (1909). See Olver, Asymptotics and Special Functions, p. 134.

Expansion good for $0 <= phase(v) < \pi$

A&S recommend "uniform expansion" with complicated coefficients and Airy function.

Debye's Formula is in 9.3.7,9.3.9,9.3.10 of A&S

AXIOM recurrence for u_k

```
  f(0)==1::EXPR INT
  f(n)== (t^2)*(1-t^2)*D(f(n-1),t)/2 + (1/8)*integrate( (1-5*t^2)*f(n-1),t)
```

— **defun BesselJAsymptOrder** —

```
(defun BesselJAsymptOrder (v z)
 (let (ca8 ca4 ca2 ca tanhalpha alpha sechalpha)
  (setq sechalpha (/ z v))
  (setq alpha (acosh (/ 1.0 sechalpha)))
  (setq tanhalpha (sqrt (- 1.0 (* sechalpha sechalpha))))
  (setq ca (/ 1.0 tanhalpha))
  (setq ca2 (* ca ca))
  (setq ca4 (* ca2 ca2))
  (setq ca8 (* ca4 ca4))
  (* (/ (exp (- (* v (- alpha tanhalpha))))
              (sqrt (* (* (* 2.0 Pi) v) tanhalpha)))
     (+ (+ (+ (+ (+ (+ 1.0 (/ (* (horner '(-5.0 3.0) ca2) ca)
                               (* v 24.0)))
                    (/ (* (horner '(385.0 -462.0 81.0) ca2) ca2)
                       (* (* 1152.0 v) v)))
                 (/ (* (* (horner
                           '(-425425.0 765765.0 -369603.0 30375.0)
                           ca2)
                          ca2)
                       ca)
                    (* (* (* 414720.0 v) v) v)))
              (/ (* (horner
                      '(1.85910725E8 -4.4618574E8 3.4992243E8
                        -9.4121676E7 4465125.0)
                      ca2)
                    ca4)
                 (* (* (* (* 3.981312E7 v) v) v) v)))
           (/ (* (* (horner
                      '(-1.88699385875E11 5.66098157625E11
                        -6.1413587235E11  2.84499769554E11
                        -4.9286948607E10  1.519035525E9)
                      ca2)
                    ca4)
```

```
                ca)
             (* (* (* (* (* 6.68860416E9 v) v) v) v) v)))
          (/ (* (* (horner
                     '(1.023694168371875E15 -3.6852990061387505E15
                       5.104696716244125E15 -3.36903206826186E15
                       1.050760774457901E15 -1.2757729835475E14
                       2.757049477875E12)
                     ca2)
                   ca4)
                ca2)
             (* (* (* (* (* (* 4.8157949952E12 v) v) v) v) v) v))))))
```

———-

40.0.83 defun chebf01

Where:

c parameter to 0F1, possibly complex
z argument to 0F1
w scale factor so that $0 < \frac{z}{w} < 1$
n, $n+2$ coefficients will be produced stored in an array indexed from 0 to $n+1$.

Program transcribed from Fortran,, p. 80 [Luke77]

— **defun chebf01** —

```
(defun chebf01 (c z)
 (let (cc temp b p sum rho divfac c1 x1 arr
       ncount z1 a1 a2 a3 n2 n1 start four w n)
  (setq n 75)                          ; ad hoc decision
  (setq w (* 2.0 z))
  ; arr will be used to store the Cheb. series coefficients
  (setq four 4.0)
  (setq start (expt 10.0 -200))
  (setq n1 (+ n 1))
  (setq n2 (+ n 2))
  (setq a3 0.0)
  (setq a2 0.0)
  (setq a1 start)                      ; arbitrary starting value
  (setq z1 (/ four w))
  (setq ncount n1)
  (setq arr (make-array n2))
  (setf (aref arr ncount) start)  ; start off
  (setq x1 n2)
  (setq c1 (- 1.0 c))
  (loop for ncount from n downto 0 by 1 do
    (setq divfac (/ 1.0 x1))
    (setq x1 (- x1 1.0))
    (setf (aref arr ncount)
      (* x1 (- (+ (* (+ divfac (* z1 (- x1 c1))) a1)
                  (* (+ (/ 1.0 x1) (* z1 (+ (+ x1 c1) 1.0))) a2))
               (* divfac a3))))
```

```
  (setq a3 a2)
  (setq a2 a1)
  (setq a1 (aref arr ncount)))
(setf (aref arr 0) (/ (aref arr 0) 2.0))
; compute scale factor
(setq rho (aref arr 0))
(setq sum rho)
(setq p 1.0)
(loop for i from 1 to n1 do
  (setq rho (- rho (* p (aref arr i))))
  (setq sum (+ sum (aref arr i)))
  (setq p (- p)))
(loop for m from 0 to n1 do
  (setf (aref arr m) (/ (aref arr m) rho)))
(setq sum (/ sum rho))
; Now evaluate array at argument
(setq b 0.0)
(setq temp 0.0)
(loop for i from (+ n 1) downto 0 by 1 do
  (setq cc b)
  (setq b temp)
  (setq temp (+ (- cc) (aref arr i))))
temp))
```

Chapter 41

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.

41.1 AlgebraicFunction

41.1.1 defun retract

[objMode p462]
[objVal p462]
[isWrapped p1485]
[qcar p??]
[retract1 p??]
[mkObj p460]
[$EmptyMode p629]

— **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 |$PositiveInteger|) (mkObj val |$NonNegativeInteger|))
       ((equal type |$NonNegativeInteger|) (mkObj val |$Integer|))
       ((and (equal type |$Integer|) (typep (|unwrap| val) 'fixnum))
         (mkObj val |$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 (mkObj val type))) '|failed|)
        ans)
      (t
```

```
        (mkObj (|objVal| ans) (|objMode| ans)))))))))))
```

———-

41.2 Any

41.2.1 defun spad2BootCoerce

— defun spad2BootCoerce —

```
(defun |spad2BootCoerce| (x source target)
 (let (xp)
  (cond
   ((null (|isValidType| source))
    (|throwKeyedMsg| "%1p is not a valid type." (list source)))
   ((null (|isValidType| target))
    (|throwKeyedMsg| "%1p is not a valid type." (list target)))
   ((setq xp (|coerceInteractive| (mkObjWrap x source) target))
     (|objValUnwrap| xp))
   (t
    (|throwKeyedMsgCannotCoerceWithValue| (|wrap| x) source target)))))
```

———-

41.3 ApplicationProgramInterface

41.3.1 defun Report what domains get instantiated

[$reportinstantiations p1139]

— defun reportinstantiations —

```
(defun reportinstantiations (b)
 (setq |$reportInstantiations| b))
```

———-

41.4 Boolean

41.4.1 defun The Boolean = function support

— defun BooleanEquality 0 —

```
(defun |BooleanEquality| (x y) (if x y (null y)))
```

———-

41.5 Char

41.5.1 defun upcase

[identp p1107]
[upcase p1140]

— **defun upcase** —

```
(defun upcase (l)
  (cond ((stringp l) (string-upcase l))
        ((identp l) (intern (string-upcase (symbol-name l))))
        ((characterp l) (char-upcase l))
        ((atom l) l)
        (t (mapcar #'upcase l))))
```

————-

41.5.2 defun downcase

[identp p1107]
[downcase p1140]

— **defun downcase** —

```
(defun downcase (l)
  (cond ((stringp l) (string-downcase l))
        ((identp l) (intern (string-downcase (symbol-name l))))
        ((characterp l) (char-downcase L))
        ((atom l) l)
        (t (mapcar #'downcase l))))
```

————-

41.6 ComplexDoubleFloatMatrix

41.6.1 defmacro make-cdouble-matrix

ComplexDoubleFloatMatrix function support

— **defmacro make-cdouble-matrix 0** —

```
(defmacro make-cdouble-matrix (n m)
 `(make-array (list ,n (* 2 ,m)) :element-type 'double-float))
```

————-

41.6.2 defmacro cdaref2

ComplexDoubleFloatMatrix function support
— **defmacro cdaref2 0** —

```
(defmacro cdaref2 (ov oi oj)
   (let ((v (gensym))
         (i (gensym))
         (j (gensym)))
   `(let ((,v ,ov)
          (,i ,oi)
          (,j ,oj))
        (cons
            (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
            (aref (the (simple-array double-float (* *)) ,v)
                  ,i (+ (* 2 ,j) 1))))))
```

———-

41.6.3 defmacro cdsetaref2

ComplexDoubleFloatMatrix function support
— **defmacro cdsetaref2 0** —

```
(defmacro cdsetaref2 (ov oi oj os)
   (let ((v (gensym))
         (i (gensym))
         (j (gensym))
         (s (gensym)))
   `(let ((,v ,ov)
          (,i ,oi)
          (,j ,oj)
          (,s ,os))
         (setf (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
               (car ,s))
         (setf (aref (the (simple-array double-float (* *)) ,v)
                     ,i (+ (* 2 ,j) 1))
               (cdr ,s))
         ,s)))
```

———-

41.6.4 defmacro cdanrows

ComplexDoubleFloatMatrix function support
— **defmacro cdanrows 0** —

```
(defmacro cdanrows (v)
    `(array-dimension (the (simple-array double-float (* *)) ,v) 0))
```

———-

41.6.5 defmacro cdancols

ComplexDoubleFloatMatrix function support
— **defmacro cdancols 0** —

```
(defmacro cdancols (v)
 `(truncate
    (array-dimension (the (simple-array double-float (* *)) ,v) 1) 2))
```

———-

41.7 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.

41.7.1 defmacro make-cdouble-vector

ComplexDoubleFloatVector Qnew function support
— **defmacro make-cdouble-vector 0** —

```
(defmacro make-cdouble-vector (n)
   `(make-array (list (* 2 ,n)) :element-type 'double-float))
```

———-

41.7.2 defmacro cdelt

ComplexDoubleFloatVector Qelt1 function support
— **defmacro cdelt 0** —

```
(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))))))
```

———-

41.7.3 defmacro cdsetelt

ComplexDoubleFloatVector Qsetelt1 function support
— **defmacro cdsetelt 0** —

```
(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)))
```

———-

41.7.4 defmacro cdlen

ComplexDoubleFloatVector Qsize function support

— **defmacro cdlen 0** —

```
(defmacro cdlen(v)
  `(truncate (length (the (simple-array double-float (*)) ,v)) 2))
```

———-

41.8 Database

41.8.1 defun Database elt function support

[basicMatch? p**??**]

— **defun stringMatches?** —

```
(defun |stringMatches?| (pattern subject)
 (when (integerp (|basicMatch?| pattern subject)) t))
```

———-

41.9 DirectProduct

41.9.1 defun vec2list

— **defun vec2list 0** —

```
(defun vec2list (vec)
 (coerce vec 'list))
```

———-

41.10 DoubleFloat

These macros wrap their arguments with strong type information in order to optimize doublefloat computatations. They are used directly in the DoubleFloat domain (see Volume 10.3).

41.10.1 defmacro DFLessThan

Compute a strongly typed doublefloat comparison See Steele Common Lisp 1990 p293

— **defmacro DFLessThan 0** —

```
(defmacro DFLessThan (x y)
 `(< (the double-float ,x) (the double-float ,y)))
```

———-

41.11 DoubleFloatSpecialFunctions

41.11.1 defun Real Gamma $\Gamma(x)$

[rgammaImpl p1115]

— **defun rgamma** —

```
(defun rgamma (x)
  (rgammaImpl x))
```

———-

41.11.2 defun Complex Gamma $\Gamma(z)$

The cgamma(z) function is the Γ function for complex arguments implemented by Bruce Char, April-May, 1990.

Our text for complex gamma is H. Kuki's paper Complex Gamma Function with Error Control"[kuki72a]

It uses the reflection formula and the basic $z+1$ recurrence to transform the argument into something that Stirling's asymptotic formula can handle.

However along the way it does a few tricky things to reduce problems due to round-off/cancellation error for particular values.

Small float implementation of Gamma function. Valid for real arguments. Maximum error (relative) seems to be 2-4 ulps for real x $2 < x < 9$, and up to ten ulps for larger x up to overflow. See Hart and Cheney. (Bruce Char, April, 1990).

[cgammaImpl p1117]
[c-to-s p1113]
[s-to-c p1113]

— **defun cgamma** —

```
(defun cgamma (z)
  (c-to-s (cgammaImpl (s-to-c z)) ))
```

————-

41.11.3 defun The complex logGamma function

[lncgamma p1125]
[c-to-s p1113]
[s-to-c p1113]

— **defun clngamma** —

```
(defun clngamma (z)
  (c-to-s (lncgamma (s-to-c z)) ))
```

————-

41.11.4 defun The real logGamma function

[lnrgamma p1115]

— **defun rlngamma** —

```
(defun rlngamma (x)
  (lnrgamma x))
```

————-

41.11.5 defun The real Psi function

[rPsiImpl p1125]

— **defun rpsi** —

```
(defun rpsi (n x)
  (rPsiImpl n x))
```

————-

41.11.6 defun The complex Psi function

[c-to-s p1113]
[cPsi p**??**]
[s-to-c p1113]

— **defun cpsi** —

```
(defun cpsi (n z)
  (c-to-s (cPsiImpl n (s-to-c z)) ))
```

———-

41.11.7 defun The real BesselJ function

[c-to-r p1114]
[BesselJ p1130]

— **defun rbesselj** —

```
(defun rbesselj (n x)
  (c-to-r (BesselJ n x)) ))
```

———-

41.11.8 defun The complex BesselJ function

[c-to-s p1113]
[BesselJ p1130]
[s-to-c p1113]

— **defun cbesselj** —

```
(defun cbesselj (v z)
  (c-to-s (BesselJ (s-to-c v) (s-to-c z)) ))
```

———-

41.11.9 defun The real BesselI function

[c-to-r p1114]
[BesselI p1121]

— **defun rbesseli** —

```
(defun rbesseli (n x)
  (c-to-r (BesselI n x)) ))
```

———-

41.11.10 defun The complex BesselI function

[c-to-s p1113]
[BesselI p1121]
[s-to-c p1113]

— **defun cbesseli** —

```
(defun cbesseli (v z)
  (c-to-s (BesselI (s-to-c v) (s-to-c z)) ))
```

———-

41.11.11 defun The complex hypergeometric function

[c-to-s p1113]
[chebf01 p1134]
[s-to-c p1113]

— **defun chyper0f1** —

```
(defun chyper0f1 (a z)
  (c-to-s (chebf01 (s-to-c a) (s-to-c z)) ))
```

———-

41.11.12 defmacro DFUnaryMinus

Compute a strongly typed unary doublefloat minus See Steele Common Lisp 1990 p295

— **defmacro DFUnaryMinus 0** —

```
(defmacro DFUnaryMinus (x)
 `(the double-float (- (the double-float ,x))))
```

———-

41.11.13 defmacro DFMinusp

Compute a strongly typed unary doublefloat test for negative See Steele Common Lisp 1990 p292

— **defmacro DFMinusp 0** —

```
(defmacro DFMinusp (x)
 `(minusp (the double-float ,x)))
```

———-

41.11.14 defmacro DFZerop

Compute a strongly typed unary doublefloat test for zero See Steele Common Lisp 1990 p292

— **defmacro DFZerop 0** —

```
(defmacro DFZerop (x)
 `(zerop (the double-float ,x)))
```

———-

41.11.15 defmacro DFAdd

Compute a strongly typed doublefloat addition See Steele Common Lisp 1990 p295

— **defmacro DFAdd 0** —

```
(defmacro DFAdd (x y)
 `(the double-float (+ (the double-float ,x) (the double-float ,y))))
```

———-

41.11.16 defmacro DFSubtract

Compute a strongly typed doublefloat subtraction See Steele Common Lisp 1990 p295

— **defmacro DFSubtract 0** —

```
(defmacro DFSubtract (x y)
 `(the double-float (- (the double-float ,x) (the double-float ,y))))
```

———-

41.11.17 defmacro DFMultiply

Compute a strongly typed doublefloat multiplication See Steele Common Lisp 1990 p296

— **defmacro DFMultiply 0** —

```
(defmacro DFMultiply (x y)
 `(the double-float (* (the double-float ,x) (the double-float ,y))))
```

———-

41.11.18 defmacro DFIntegerMultiply

Compute a strongly typed doublefloat multiplication by an integer. See Steele Common Lisp 1990 p296

— **defmacro DFIntegerMultiply 0** —

```
(defmacro DFIntegerMultiply (i y)
 `(the double-float (* (the integer ,i) (the double-float ,y))))
```

———-

41.11.19 defmacro DFMax

Choose the maximum of two doublefloats. See Steele Common Lisp 1990 p294

— **defmacro DFMax 0** —

```
(defmacro DFMax (x y)
 `(the double-float (max (the double-float ,x) (the double-float ,y))))
```

———-

41.11.20 defmacro DFMin

Choose the minimum of two doublefloats. See Steele Common Lisp 1990 p294

— **defmacro DFMin 0** —

```
(defmacro DFMin (x y)
 `(the double-float (min (the double-float ,x) (the double-float ,y))))
```

———-

41.11.21 defmacro DFEql

Compare two doublefloats for equality, where equality is eq, or numbers of the same type with the same value. See Steele Common Lisp 1990 p105

— **defmacro DFEql 0** —

```
(defmacro DFEql (x y)
 `(eql (the double-float ,x) (the double-float ,y)))
```

———-

41.11.22 defmacro DFDivide

Divide a doublefloat by a a doublefloat See Steele Common Lisp 1990 p296

— **defmacro DFDivide 0** —

```
(defmacro DFDivide (x y)
 `(the double-float (/ (the double-float ,x) (the double-float ,y))))
```

———-

41.11.23 defmacro DFIntegerDivide

Divide a doublefloat by an integer See Steele Common Lisp 1990 p296

— **defmacro DFIntegerDivide 0** —

```
(defmacro DFIntegerDivide (x i)
 `(the double-float (/ (the double-float ,x) (the integer ,i))))
```

———-

41.11.24 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 0** —

```
(defmacro DFSqrt (x)
 `(sqrt (the double-float ,x)))
```

———-

41.11.25 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 0** —

```
(defmacro DFLogE (x)
 `(log (the double-float ,x)))
```

———-

41.11.26 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 0** —

```
(defmacro DFLog (x b)
 `(log (the double-float ,x) (the fixnum ,b)))
```

———-

41.11.27 defmacro DFIntegerExpt

Compute the doublefloat expt of x with a given integer power i See Steele Common Lisp 1990 p300

— **defmacro DFIntegerExpt 0** —

```
(defmacro DFIntegerExpt (x i)
 `(the double-float (expt (the double-float ,x) (the integer ,i))))
```

———-

41.11.28 defmacro DFExpt

Compute the doublefloat expt of x with a given power p. The result could be complex if the base is negative and the power is not an integer. See Steele Common Lisp 1990 p300

— **defmacro DFExpt 0** —

```
(defmacro DFExpt (x p)
 `(expt (the double-float ,x) (the double-float ,p)))
```

———-

41.11.29 defmacro DFExp

Compute the doublefloat exp with power e See Steele Common Lisp 1990 p300

— **defmacro DFExp 0** —

```
(defmacro DFExp (x)
 `(the double-float (exp (the double-float ,x))))
```

————-

41.11.30 defmacro DFSin

Compute a strongly typed doublefloat sin See Steele Common Lisp 1990 p304

— **defmacro DFSin 0** —

```
(defmacro DFSin (x)
 `(the double-float (sin (the double-float ,x))))
```

————-

41.11.31 defmacro DFCos

Compute a strongly typed doublefloat cos See Steele Common Lisp 1990 p304

— **defmacro DFCos 0** —

```
(defmacro DFCos (x)
 `(the double-float (cos (the double-float ,x))))
```

————-

41.11.32 defmacro DFTan

Compute a strongly typed doublefloat tan See Steele Common Lisp 1990 p304

— **defmacro DFTan 0** —

```
(defmacro DFTan (x)
 `(the double-float (tan (the double-float ,x))))
```

————-

41.11.33 defmacro DFAsin

Compute a strongly typed doublefloat asin. The result is complex if the absolute value of the argument is greater than 1. See Steele Common Lisp 1990 p305

— **defmacro DFAsin 0** —

```
(defmacro DFAsin (x)
 `(asin (the double-float ,x)))
```

————-

41.11.34 defmacro DFAcos

Compute a strongly typed doublefloat acos. The result is complex if the absolute value of the argument is greater than 1. See Steele Common Lisp 1990 p305

— **defmacro DFAcos 0** —

```
(defmacro DFAcos (x)
 `(acos (the double-float ,x)))
```

———-

41.11.35 defmacro DFAtan

Compute a strongly typed doublefloat atan See Steele Common Lisp 1990 p305

— **defmacro DFAtan 0** —

```
(defmacro DFAtan (x)
 `(the double-float (atan (the double-float ,x))))
```

———-

41.11.36 defmacro DFAtan2

Compute a strongly typed doublefloat atan with 2 arguments

$y = 0$	$x > 0$	Positive x-axis	0
$y > 0$	$x > 0$	Quadrant I	$0 <$ result $< \pi/2$
$y > 0$	$x = 0$	Positive y-axis	$\pi/2$
$y > 0$	$x < 0$	Quadrant II	$\pi/2 <$ result $< \pi$
$y = 0$	$x < 0$	Negative x-axis	π
$y < 0$	$x < 0$	Quadrant III	$-\pi <$ result $< -\pi/2$
$y < 0$	$x = 0$	Negative y-axis	$-\pi/2$
$y < 0$	$x > 0$	Quadrant IV	$-\pi/2 <$ result < 0
$y = 0$	$x = 0$	Origin	error

See Steele Common Lisp 1990 p306

— **defmacro DFAtan2 0** —

```
(defmacro DFAtan2 (y x)
 `(the double-float (atan (the double-float ,x) (the double-float ,y))))
```

———-

41.11.37 defmacro DFSinh

Compute a strongly typed doublefloat sinh

$$(e^z - e^{-z})/2$$

See Steele Common Lisp 1990 p308

— **defmacro DFSinh 0** —

```
(defmacro DFSinh (x)
 `(the double-float (sinh (the double-float ,x))))
```

———-

41.11.38 defmacro DFCosh

Compute a strongly typed doublefloat cosh

$$(e^z + e^{-z})/2$$

See Steele Common Lisp 1990 p308

— **defmacro DFCosh 0** —

```
(defmacro DFCosh (x)
 `(the double-float (cosh (the double-float ,x))))
```

———-

41.11.39 defmacro DFTanh

Compute a strongly typed doublefloat tanh

$$(e^z - e^{-z})/(e^z + e^{-z})$$

See Steele Common Lisp 1990 p308

— **defmacro DFTanh 0** —

```
(defmacro DFTanh (x)
 `(the double-float (tanh (the double-float ,x))))
```

———-

41.11.40 defmacro DFAsinh

Compute the inverse hyperbolic sin.

$$log\left(z + \sqrt{1 + z^2}\right)$$

See Steele Common Lisp 1990 p308

— **defmacro DFAsinh 0** —

```
(defmacro DFAsinh (x)
 `(the double-float (asinh (the double-float ,x))))
```

———-

41.11.41 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 0** —

```
(defmacro DFAcosh (x)
 `(acosh (the double-float ,x)))
```

———-

41.11.42 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 0** —

```
(defmacro DFAtanh (x)
 `(atanh (the double-float ,x)))
```

———-

41.11.43 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))
```

———-

41.11.44 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))))
```

———-

41.11.45 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))
```

———-

41.11.46 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))
```

———-

41.11.47 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)))
```

———-

41.11.48 defun The cotangent routine

The cotangent function is defined as

$$cot(z) = \frac{1}{tan(z)}$$

— **defun cot 0** —

```
(defun cot (a)
  (if (or (> a 1000.0) (< a -1000.0))
    (/ (cos a) (sin a))
    (/ 1.0 (tan a))))
```

———-

41.11.49 defun The inverse cotangent function

The inverse cotangent (arc-cotangent) function is defined as

$$acot(z) = cot^{-1}(z) = tan^{-1}(\frac{1}{z})$$

See Steele Common Lisp 1990 pp305-307

— **defun acot 0** —

```
(defun acot (a)
  (if (> a 0.0)
    (if (> a 1.0)
       (atan (/ 1.0 a))
       (- (/ pi 2.0) (atan a)))
    (if (< a -1.0)
       (- pi (atan (/ -1.0 a)))
       (+ (/ pi 2.0) (atan (- a))))))
```

————-

41.11.50 defun The secant function

$$sec(x) = \frac{1}{cos(x)}$$

— **defun sec 0** —

```
(defun sec (x) (/ 1 (cos x)))
```

————-

41.11.51 defun The inverse secant function

$$asec(x) = acos\left(\frac{1}{x}\right)$$

— **defun asec 0** —

```
(defun asec (x) (acos (/ 1 x)))
```

————-

41.11.52 defun The cosecant function

$$csc(x) = \frac{1}{sin(x)}$$

— **defun csc 0** —

```
(defun csc (x) (/ 1 (sin x)))
```

————-

41.11.53 defun The inverse cosecant function

$$acsc(x) = \frac{1}{asin(x)}$$

— **defun acsc 0** —

```
(defun acsc (x) (asin (/ 1 x)))
```

———-

41.11.54 defun The hyperbolic cosecant function

$$csch(x) = \frac{1}{sinh(x)}$$

— **defun csch 0** —

```
(defun csch (x) (/ 1 (sinh x)))
```

———-

41.11.55 defun The hyperbolic cotangent function

$$coth(x) = cosh(x)csch(x)$$

— **defun coth 0** —

```
(defun coth (x) (* (cosh x) (csch x)))
```

———-

41.11.56 defun The hyperbolic secant function

$$sech(x) = \frac{1}{cosh(x)}$$

— **defun sech 0** —

```
(defun sech (x) (/ 1 (cosh x)))
```

———-

41.11.57 defun The inverse hyperbolic cosecant function

$$acsch(x) = asinh\left(\frac{1}{x}\right)$$

— **defun acsch 0** —

```
(defun acsch (x) (asinh (/ 1 x)))
```

———-

41.11.58 defun The inverse hyperbolic cotangent function

$$acoth(x) = atanh\left(\frac{1}{x}\right)$$

— **defun acoth 0** —

```
(defun acoth (x) (atanh (/ 1 x)))
```

———-

41.11.59 defun The inverse hyperbolic secant function

$$asech(x) = acosh\left(\frac{1}{x}\right)$$

— **defun asech 0** —

```
(defun asech (x) (acosh (/ 1 x)))
```

———-

41.12 DoubleFloatMatrix

41.12.1 defmacro make-double-matrix

DoubleFloatMatrix qnew function support

— **defmacro make-double-matrix 0** —

```
(defmacro make-double-matrix (n m)
   `(make-array (list ,n ,m) :element-type 'double-float))
```

———-

41.12.2 defmacro make-double-matrix1

DoubleFloatMatrix new function support

— **defmacro make-double-matrix1 0** —

```
(defmacro make-double-matrix1 (n m s)
 `(make-array (list ,n ,m) :element-type 'double-float :initial-element ,s))
```

———-

41.12.3 defmacro daref2

DoubleFloatMatrix qelt function support
— **defmacro daref2 0** —

```
(defmacro daref2 (v i j)
 `(aref (the (simple-array double-float (* *)) ,v) ,i ,j))
```

———-

41.12.4 defmacro dsetaref2

DoubleFloatMatrix qsetelt! function support
— **defmacro dsetaref2 0** —

```
(defmacro dsetaref2 (v i j s)
  `(setf (aref (the (simple-array double-float (* *)) ,v) ,i ,j) ,s))
```

———-

41.12.5 defmacro danrows

DoubleFloatMatrix nrows function support
— **defmacro danrows 0** —

```
(defmacro danrows (v)
  `(array-dimension (the (simple-array double-float (* *)) ,v) 0))
```

———-

41.12.6 defmacro dancols

DoubleFloatMatrix ncols function support
— **defmacro dancols 0** —

```
(defmacro dancols (v)
  `(array-dimension (the (simple-array double-float (* *)) ,v) 1))
```

———-

41.13 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.

41.13.1 defmacro dlen

DoubleFloatVector Qsize function support
— **defmacro dlen 0** —

```
(defmacro dlen (v)
 `(length (the (simple-array double-float (*)) ,v)))
```

———-

41.13.2 defmacro make-double-vector

DoubleFloatVector Qnew function support
— **defmacro make-double-vector 0** —

```
(defmacro make-double-vector (n)
 `(make-array (list ,n) :element-type 'double-float))
```

———-

41.13.3 defmacro make-double-vector1

DoubleFloatVector Qnew1 function support
— **defmacro make-double-vector1 0** —

```
(defmacro make-double-vector1 (n s)
 `(make-array (list ,n) :element-type 'double-float :initial-element ,s))
```

———-

41.13.4 defmacro delt

DoubleFloatVector Qelt1 function support
— **defmacro delt 0** —

```
(defmacro delt (v i)
 `(aref (the (simple-array double-float (*)) ,v) ,i))
```

———-

41.13.5 defmacro dsetelt

DoubleFloatVector Qsetelt1 function support
— **defmacro dsetelt 0** —

```
(defmacro dsetelt (v i s)
 `(setf (aref (the (simple-array double-float (*)) ,v) ,i) ,s))
```

———-

41.14 File

41.14.1 defvar *read-place-holder*

— initvars —

```
(defvar *read-place-holder* (make-symbol "%.EOF")
   "default value returned by read and read-line at end-of-file")
```

———-

41.14.2 defun placep

[*read-place-holder* p1161]

— defun placep 0 —

```
(defun placep (item)
 (declare (special *read-place-holder*))
 (eq item *read-place-holder*))
```

———-

41.14.3 defun vmread

— defun vmread 0 —

```
(defun vmread (&optional (st *standard-input*) (eofval *read-place-holder*))
  (read st nil eofval))
```

———-

41.15 FileName

41.15.1 defun FileName filename function implementation

[StringToDir p1162]

— defun fnameMake —

```
(defun |fnameMake| (d n e)
  (if (string= e "") (setq e nil))
  (make-pathname :directory (|StringToDir| d) :name  n :type e))
```

———-

41.15.2 defun FileName filename support function

[lastc p??]

— **defun StringToDir** —

```
(defun |StringToDir| (s)
  (cond
    ((string= s "/") '(:root))
    ((string= s "")  nil)
    (t
      (let ((lastc (aref s (- (length s) 1))))
        (if (char= lastc #\/)
          (pathname-directory (concat s "name.type"))
          (pathname-directory (concat s "/name.type")) ))) ))
```

———-

41.15.3 defun FileName directory function implementation

[DirToString p1162]

— **defun fnameDirectory** —

```
(defun |fnameDirectory| (f)
  (|DirToString| (pathname-directory (string f))))
```

———-

41.15.4 defun FileName directory function support

For example, "/" "/u/smwatt" "../src"

— **defun DirToString 0** —

```
(defun |DirToString| (d)
  (cond
    ((equal d '(:root)) "/")
    ((null d) "")
    ('t (string-right-trim "/" (namestring (make-pathname :directory d)))) ))
```

———-

41.15.5 defun FileName name function implementation

— **defun fnameName 0** —

```
(defun |fnameName| (f)
  (let ((s (pathname-name (string f))))
    (if s s "") ))
```

———-

41.15.6 defun FileName extension function implementation

— defun fnameType 0 —

```
(defun |fnameType| (f)
  (let ((s (pathname-type (string f))))
    (if s s "") ))
```

———-

41.15.7 defun FileName exists? function implementation

— defun fnameExists? 0 —

```
(defun |fnameExists?| (f)
  (if (probe-file (namestring f)) 't nil))
```

———-

41.15.8 defun FileName readable? function implementation

— defun fnameReadable? 0 —

```
(defun |fnameReadable?| (f)
 (let ((s (open f :direction :input :if-does-not-exist nil)))
  (cond (s (close s) t) (t nil)) ))
```

———-

41.15.9 defun FileName writeable? function implementation

[myWriteable? p??]

— defun fnameWritable? —

```
(defun |fnameWritable?| (f)
  (|myWritable?| (namestring f)) )
```

———-

41.15.10 defun FileName writeable? function support

[error p??]
[fnameExists? p1163]

[fnameDirectory p1162]
[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) )
```

———-

41.15.11 defun FileName new function implementation

[fnameMake p1161]

— 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)) )))
```

———-

41.16 IndexedBits

41.16.1 defmacro truth-to-bit

IndexedBits new function support

— defmacro truth-to-bit —

```
(defmacro truth-to-bit (x) `(cond (,x 1) ('else 0)))
```

———-

41.16.2 defun IndexedBits new function support

— defun bvec-make-full 0 —

```
(defun bvec-make-full (n x)
 (make-array (list n) :element-type 'bit :initial-element x))
```

———-

41.16.3 defmacro bit-to-truth

IndexedBits elt function support
— **defmacro bit-to-truth 0** —

```
(defmacro bit-to-truth (b) `(eq ,b 1))
```

———-

41.16.4 defmacro bvec-elt

IndexedBits elt function support
— **defmacro bvec-elt 0** —

```
(defmacro bvec-elt (bv i) `(sbit ,bv ,i))
```

———-

41.16.5 defmacro bvec-setelt

IndexedBits setelt function support
— **defmacro bvec-setelt** —

```
(defmacro bvec-setelt (bv i x) `(setf (sbit ,bv ,i) ,x))
```

———-

41.16.6 defmacro bvec-size

IndexedBits length function support
— **defmacro bvec-size** —

```
(defmacro bvec-size (bv) `(size ,bv))
```

———-

41.16.7 defun IndexedBits concat function support

— **defun bvec-concat 0** —

```
(defun bvec-concat (bv1 bv2) (concatenate '(vector bit) bv1 bv2))
```

———-

41.16.8 defun IndexedBits copy function support

— defun bvec-copy 0 —

```
(defun bvec-copy (bv) (copy-seq bv))
```

————-

41.16.9 defun IndexedBits = function support

— defun bvec-equal 0 —

```
(defun bvec-equal (bv1 bv2) (equal bv1 bv2))
```

————-

41.16.10 defun IndexedBits < function support

— defun bvec-greater 0 —

```
(defun bvec-greater (bv1 bv2)
  (let ((pos (mismatch bv1 bv2)))
    (cond ((or (null pos) (>= pos (length bv1))) nil)
          ((< pos (length bv2)) (> (bit bv1 pos) (bit bv2 pos)))
          ((find 1 bv1 :start pos) t)
          (t nil))))
```

————-

41.16.11 defun IndexedBits And function support

— defun bvec-and 0 —

```
(defun bvec-and (bv1 bv2) (bit-and  bv1 bv2))
```

————-

41.16.12 defun IndexedBits Or function support

— defun bvec-or 0 —

```
(defun bvec-or (bv1 bv2) (bit-ior  bv1 bv2))
```

————-

41.16.13 defun IndexedBits xor function support

— **defun bvec-xor 0** —

```
(defun bvec-xor (bv1 bv2) (bit-xor  bv1 bv2))
```

————-

41.16.14 defun IndexedBits nand function support

— **defun bvec-nand 0** —

```
(defun bvec-nand (bv1 bv2) (bit-nand bv1 bv2))
```

————-

41.16.15 defun IndexedBits nor function support

— **defun bvec-nor 0** —

```
(defun bvec-nor (bv1 bv2) (bit-nor  bv1 bv2))
```

————-

41.16.16 defun IndexedBits not function support

— **defun bvec-not 0** —

```
(defun bvec-not (bv) (bit-not  bv))
```

————-

41.17 IndexCard

41.17.1 defun IndexCard origin function support

[dbPart p**??**]
[charPosition p**??**]
[substring p293]

— **defun alqlGetOrigin** —

```
(defun |alqlGetOrigin| (x)
 (let (field k)
  (setq field (|dbPart| x 5 1))
```

```
  (setq k (|charPosition| #\( field 2))
  (substring field 1 (1- k))))
```

———-

41.17.2 defun IndexCard origin function support

[dbPart p**??**]
[charPosition p**??**]
[substring p293]

— **defun alqlGetParams** —

```
(defun |alqlGetParams| (x)
 (let (field k)
  (setq field (|dbPart| x 5 1))
  (setq k (|charPosition| #\( field 2))
  (substring field k nil)))
```

———-

41.17.3 defun IndexCard elt function support

[dbPart p**??**]
[substring p293]

— **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)))
```

———-

41.18 IndexedString

41.18.1 defun qenum

This is also used in bookvol10.3 in the CHAR domain.

— **defun qenum 0** —

```
(defun qenum (cvec ind)
 (char-code (char cvec ind)))
```

———-

— **defmacro qcsize 0** —

```
(defmacro qcsize (x)
 `(the fixnum (length (the simple-string ,x))))
```

———-

— **defun qeset 0** —

```
(defun qeset (cvec ind charnum)
  (setf (char cvec ind) (code-char charnum)))
```

———-

— **defmacro qsgreaterp 0** —

```
(defmacro qsgreaterp (a b)
 `(> (the fixnum ,a) (the fixnum ,b)))
```

———-

41.19 InputForm

41.19.1 defun called by interpret function

— **defun mkObjFn 0** —

```
(defun |mkObjFn| (val mode)
 (cons mode val))
```

———-

41.19.2 defun called by interpret function

— **defun objValFn 0** —

```
(defun |objValFn| (obj)
 (cdr obj))
```

———-

41.19.3 defun called by interpret function

— **defun objModeFn 0** —

```
(defun |objModeFn| (obj)
 (car obj))
```

———-

41.19.4 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.

[$InteractiveMode p284]
[$formatSigAsTex p**??**]

— **defun unparseInputForm** —

```
(defun |unparseInputForm| (u)
 (let (|$formatSigAsTeX| |$InteractiveMode|)
 (declare (special |$formatSigAsTeX| |$InteractiveMode|))
  (setq |$formatSigAsTeX| 1)
  (setq |$InteractiveMode| nil)
  (|form2StringLocal| u)))
```

———-

41.20 Integer

41.20.1 defun Integer divide function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer divide function actually inlines a call to this code. The Integer domain contains the line:

```
(PUT (QUOTE |INT;divide;2$R;44|) (QUOTE |SPADreplace|) (QUOTE DIVIDE2))
```

— **defun divide2 0** —

```
(defun divide2 (x y)
 (multiple-value-call #'cons (truncate x y)))
```

———-

41.20.2 defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer quo function actually inlines a call to this code. The Integer domain contains the line:

```
(PUT (QUOTE |INT;rem;3$;46|) (QUOTE |SPADreplace|) (QUOTE REMAINDER2))
```

Because these are identical except for name we make the symbol-functions equivalent. This was done in the original code for efficiency.

— **defun remainder2 0** —

```
(setf (symbol-function 'remainder2) #'rem)
```

———-

41.20.3 defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer quo function actually inlines a call to this code. The Integer domain contains the line:

```
(PUT (QUOTE |INT;quo;3$;45|) (QUOTE |SPADreplace|) (QUOTE QUOTIENT2))
```

— **defun quotient2 0** —

```
(defun quotient2 (x y)
 (values (truncate x y)))
```

———-

41.20.4 defun Integer random function support

This is used for calls to random with no arguments. If an argument is supplied to random then the common lisp random function is called directly. This could be lifted up into the spad code.

— **defun random 0** —

```
(defun |random| () (random (expt 2 26)))
```

———-

41.21 KeyedAccessFile

41.21.1 defun KeyedAccessFile defstream function support

This is a simpler interpface to RDEFIOSTREAM [rdefiostream p**??**]

— **defun rdefinstream** —

```
(defun rdefinstream (&rest fn)
  ;; following line prevents rdefiostream from adding a default filetype
  (unless (rest fn) (setq fn (list (pathname (car fn)))))
  (rdefiostream (list (cons 'file fn) '(mode . input))))
```

———-

41.21.2 defun KeyedAccessFile defstream function support

[rdefiostream p**??**]

— defun rdefoutstream —

```
(defun rdefoutstream (&rest fn)
  ;; following line prevents rdefiostream from adding a default filetype
  (unless (rest fn) (setq fn (list (pathname (car fn)))))
  (rdefiostream (list (cons 'FILE fn) '(mode . OUTPUT))))
```

———-

41.22 NumberFormats

41.22.1 defun ncParseFromString

— defun ncParseFromString —

```
(defun |ncParseFromString| (s)
  (|zeroOneTran| (catch 'SPAD_READER (|parseFromString| s))))
```

———-

41.23 OperationsQuery

41.23.1 defun OperationQuery getDatabase function support

This function, called as `getBrowseDatabase(arg)` returns a list of appropriate entries in the browser database. The legal values for arg are

- "o" (operations)
- "k" (constructors)
- "d" (domains)
- "c" (categories)
- "p" (packages)

[member p1108]
[grepConstruct p??]
[$includeUnexposed? p??]

— defun getBrowseDatabase —

```
(defun |getBrowseDatabase| (kind)
 (let (|$includeUnexposed?|)
 (declare (special |$includeUnexposed?|))
  (setq |$includeUnexposed?| t)
  (when (|member| kind '("o" "k" "c" "d" "p"))
   (|grepConstruct| "*" (intern kind)))))
```

———-

41.24 ParametricLinearEquations

41.24.1 defun algCoerceInteractive

— **defun algCoerceInteractive** —

```
(defun |algCoerceInteractive| (p source target)
 (let (|$useConvertForCoercions| u)
 (declare (special |$useConvertForCoercions|))
  (setq |$useConvertForCoercions| t)
  (setq source (|devaluate| source))
  (setq target (|devaluate| target))
  (setq u (|coerceInteractive| (mkObjWrap p source) target))
  (if u
   (|objValUnwrap| u)
   (|error| (list "can't convert" p "of mode" source "to mode" target)))))
```

———-

41.25 Plot3d

We catch numeric errors and throw a different failure than normal. The trapNumericErrors macro will return a pair of the the form `Union(type-of-form, "failed")`. This pair is tested for eq-ness so it has to be unique. It lives in the defvar `$numericFailure`. The old value of the `$BreakMode` variable is saved in a defvar named `$oldBreakMode`.

41.25.1 defvar $numericFailure

This is a failed union branch which is the value returned for numeric failure.

— **initvars** —

```
(defvar |$numericFailure| (cons 1 "failed"))
```

———-

41.25.2 defvar $oldBreakMode

— **initvars** —

```
(defvar |$oldBreakMode| nil "the old value of the $BreakMode variable")
```

———-

41.25.3 defmacro trapNumericErrors

The following macro evaluates form returning Union(type-of form, "failed"). It is used in the `myTrap` local function in Plot3d.

— **defmacro trapNumericErrors** —

```
(defmacro |trapNumericErrors| (form)
 `(let ((|$oldBreakMode| |$BreakMode|) (|$BreakMode| '|trapNumerics|) (val))
  (declare (special |$BreakMode| |$numericFailure| |$oldBreakMode|))
   (setq val (catch '|trapNumerics| ,form))
   (if (eq val |$numericFailure|) val (cons 0 val))))
```

———-

41.26 SingleInteger

41.26.1 defun qsquotient

— **defun qsquotient 0** —

```
(defun qsquotient (a b)
 (the fixnum (truncate (the fixnum a) (the fixnum b))))
```

———-

41.26.2 defun qsremainder

— **defun qsremainder 0** —

```
(defun qsremainder (a b)
 (the fixnum (rem (the fixnum a) (the fixnum b))))
```

———-

41.26.3 defmacro qsdifference

— **defmacro qsdifference 0** —

```
(defmacro qsdifference (x y)
 `(the fixnum (- (the fixnum ,x) (the fixnum ,y))))
```

———-

41.26.4 defmacro qslessp

This is also used in IndexedString

— **defmacro qslessp 0** —

```
(defmacro qslessp (a b)
 `(< (the fixnum ,a) (the fixnum ,b)))
```

———-

41.26.5 defmacro qsadd1

— **defmacro qsadd1 0** —

```
(defmacro qsadd1 (x)
 `(the fixnum (1+ (the fixnum ,x))))
```

———-

41.26.6 defmacro qssub1

— **defmacro qssub1 0** —

```
(defmacro qssub1 (x)
 `(the fixnum (1- (the fixnum ,x))))
```

———-

41.26.7 defmacro qsminus

— **defmacro qsminus 0** —

```
(defmacro qsminus (x)
 `(the fixnum (minus (the fixnum ,x))))
```

———-

41.26.8 defmacro qsplus

— **defmacro qsplus 0** —

```
(defmacro qsplus (x y)
 `(the fixnum (+ (the fixnum ,x) (the fixnum ,y))))
```

———-

41.26.9 defmacro qstimes

— **defmacro qstimes 0** —

```
(defmacro qstimes (x y)
 `(the fixnum (* (the fixnum ,x) (the fixnum ,y))))
```

———-

41.26.10 defmacro qsabsval

— defmacro qsabsval 0 —

```
(defmacro qsabsval (x)
  `(the fixnum (abs (the fixnum ,x))))
```

———-

41.26.11 defmacro qsoddp

— defmacro qsoddp 0 —

```
(defmacro qsoddp (x)
 `(oddp (the fixnum ,x)))
```

———-

41.26.12 defmacro qszerop

— defmacro qszerop 0 —

```
(defmacro qszerop (x)
 `(zerop (the fixnum ,x)))
```

———-

41.26.13 defmacro qsmax

— defmacro qsmax 0 —

```
(defmacro qsmax (x y)
 `(the fixnum (max (the fixnum ,x) (the fixnum ,y))))
```

———-

41.26.14 defmacro qsmin

— defmacro qsmin 0 —

```
(defmacro qsmin (x y)
 `(the fixnum (min (the fixnum ,x) (the fixnum ,y))))
```

———-

41.27 Table

41.27.1 defun Table InnerTable support

We look inside the Key domain given to Table and find if there is an equality predicate associated with the domain. If found then Table will use a HashTable representation, otherwise it will use an AssociationList representation.

[knownEqualPred p**??**]
[compiledLookup p1097]
[Boolean p630]
[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))))
```

———-

41.28 U8Vector

41.28.1 defmacro qvlenU8

— **defmacro qvlenU8** —

```
(defmacro qvlenU8 (v)
 `(length (the (simple-array (unsigned-byte 8) (*)) ,v)))
```

———-

41.28.2 defmacro eltU8

— **defmacro eltU8** —

```
(defmacro eltU8 (v i)
 `(aref (the (simple-array (unsigned-byte 8) (*)) ,v) ,i))
```

———-

41.28.3 defmacro seteltU8

— defmacro seteltU8 —

```
(defmacro seteltU8 (v i s)
 `(setf (aref (the (simple-array (unsigned-byte 8) (*)) ,v) ,i), s))
```

———-

41.28.4 defun getRefvU8

— defun getRefvU8 —

```
(defun getRefvU8 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 8)))
```

———-

41.29 U16Vector

41.29.1 defmacro qvlenU16

— defmacro qvlenU16 —

```
(defmacro qvlenU16 (v)
 `(length (the (simple-array (unsigned-byte 16) (*)) ,v)))
```

———-

41.29.2 defmacro eltU16

— defmacro eltU16 —

```
(defmacro eltU16 (v i)
 `(aref (the (simple-array (unsigned-byte 16) (*)) ,v) ,i))
```

———-

41.29.3 defmacro seteltU16

— defmacro seteltU16 —

```
(defmacro seteltU16 (v i s)
 `(setf (aref (the (simple-array (unsigned-byte 16) (*)) ,v) ,i), s))
```

———-

41.29.4 defun getRefvU16

— **defun getRefvU16** —

```
(defun getRefvU16 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 16)))
```

———-

41.30 U32Vector

41.30.1 defmacro qvlenU32

— **defmacro qvlenU32** —

```
(defmacro qvlenU32 (v)
 `(length (the (simple-array (unsigned-byte 32) (*)) ,v)))
```

———-

41.30.2 defmacro eltU32

— **defmacro eltU32** —

```
(defmacro eltU32 (v i)
 `(aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i))
```

———-

41.30.3 defmacro seteltU32

— **defmacro seteltU32** —

```
(defmacro seteltU32 (v i s)
 `(setf (aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i), s))
```

———-

41.30.4 defun getRefvU32

— defun getRefvU32 —

```
(defun getRefvU32 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 32)))
```

———-

41.31 U8Matrix

41.31.1 defmacro aref2U8

— defmacro aref2U8 —

```
(defmacro aref2U8 (v i j)
 `(aref (the (simple-array (unsigned-byte 8) (* *)) ,v) ,i ,j))
```

———-

41.31.2 defmacro setAref2U8

— defmacro setAref2U8 —

```
(defmacro setAref2U8 (v i j s)
 `(setf (aref (the (simple-array (unsigned-byte 8) (* *)) ,v) ,i ,j), s))
```

———-

41.31.3 defmacro anrowsU8

— defmacro anrowsU8 —

```
(defmacro anrowsU8 (v)
 `(array-dimension (the (simple-array (unsigned-byte 8) (* *)) ,v) 0))
```

———-

41.31.4 defmacro ancolsU8

— defmacro ancolsU8 —

```
(defmacro ancolsU8 (v)
 `(array-dimension (the (simple-array (unsigned-byte 8) (* *)) ,v) 1))
```

———-

41.31.5 defmacro makeMatrixU8

— defmacro makeMatrixU8 —

```
(defmacro makeMatrixU8 (n m)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 8)
                           :initial-element 0))
```

———-

41.31.6 defmacro makeMatrix1U8

— defmacro makeMatrix1U8 —

```
(defmacro makeMatrix1U8 (n m s)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 8)
                           :initial-element ,s))
```

———-

41.32 U16Matrix

41.32.1 defmacro aref2U16

— defmacro aref2U16 —

```
(defmacro aref2U16 (v i j)
 `(aref (the (simple-array (unsigned-byte 16) (* *)) ,v) ,i ,j))
```

———-

41.32.2 defmacro setAref2U16

— defmacro setAref2U16 —

```
(defmacro setAref2U16 (v i j s)
 `(setf (aref (the (simple-array (unsigned-byte 16) (* *)) ,v) ,i ,j), s))
```

———-

41.32.3 defmacro anrowsU16

— defmacro anrowsU16 —

```
(defmacro anrowsU16 (v)
 `(array-dimension (the (simple-array (unsigned-byte 16) (* *)) ,v) 0))
```

———-

41.32.4 defmacro ancolsU16

— defmacro ancolsU16 —

```
(defmacro ancolsU16 (v)
 `(array-dimension (the (simple-array (unsigned-byte 16) (* *)) ,v) 1))
```

———-

41.32.5 defmacro makeMatrixU16

— defmacro makeMatrixU16 —

```
(defmacro makeMatrixU16 (n m)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 16)
                           :initial-element 0))
```

———-

41.32.6 defmacro makeMatrix1U16

— defmacro makeMatrix1U16 —

```
(defmacro makeMatrix1U16 (n m s)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 16)
                           :initial-element ,s))
```

———-

41.33 U32Matrix

41.33.1 defmacro aref2U32

— defmacro aref2U32 —

```
(defmacro aref2U32 (v i j)
 `(aref (the (simple-array (unsigned-byte 32) (* *)) ,v) ,i ,j))
```

———-

41.33.2 defmacro setAref2U32

— **defmacro setAref2U32** —

```
(defmacro setAref2U32 (v i j s)
 `(setf (aref (the (simple-array (unsigned-byte 32) (* *)) ,v) ,i ,j), s))
```

———-

41.33.3 defmacro anrowsU32

— **defmacro anrowsU32** —

```
(defmacro anrowsU32 (v)
 `(array-dimension (the (simple-array (unsigned-byte 32) (* *)) ,v) 0))
```

———-

41.33.4 defmacro ancolsU32

— **defmacro ancolsU32** —

```
(defmacro ancolsU32 (v)
 `(array-dimension (the (simple-array (unsigned-byte 32) (* *)) ,v) 1))
```

———-

41.33.5 defmacro makeMatrixU32

— **defmacro makeMatrixU32** —

```
(defmacro makeMatrixU32 (n m)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 32)
                           :initial-element 0))
```

———-

41.33.6 defmacro makeMatrix1U32

— **defmacro makeMatrix1U32** —

```
(defmacro makeMatrix1U32 (n m s)
 `(make-array (list ,n ,m) :element-type '(unsigned-byte 32)
                           :initial-element ,s))
```

———-

41.34 U32VectorPolynomialOperations

41.34.1 defmacro qsMulAdd6432

— **defmacro qsMulAdd6432** —

```
(defmacro qsMulAdd6432 (x y z)
  `(the (unsigned-byte 64)
     (+ (the (unsigned-byte 64)
          (* (the (unsigned-byte 32) ,x)
             (the (unsigned-byte 32) ,y)))
        (the (unsigned-byte 64) ,z))))
```

———-

41.34.2 defmacro qsMulMod32

— **defmacro qsMulMod32** —

```
(defmacro qsMulMod32 (x y &optional z)
  (declare (ignore z))
  `(the (unsigned-byte 64)
     (* (the (unsigned-byte 32) ,x)
        (the (unsigned-byte 32) ,y))))
```

———-

41.34.3 defmacro qsMod6432

— **defmacro qsMod6432** —

```
(defmacro qsMod6432 (x p)
  `(the (unsigned-byte 32)
     (rem (the (unsigned-byte 64) ,x) (the (unsigned-byte 32) ,p))))
```

———-

41.34.4 defmacro qsMulAddMod6432

— defmacro qsMulAddMod6432 —

```
(defmacro qsMulAddMod6432 (x y z p)
  `(qsMod6432 (qsMulAdd6432 ,x ,y ,z) ,p))
```

———-

41.34.5 defmacro qsMul6432

— defmacro qsMul6432 —

```
(defmacro qsMul6432 (x y)
  `(the (unsigned-byte 64)
     (* (the (unsigned-byte 32) ,x)
        (the (unsigned-byte 32) ,y))))
```

———-

41.34.6 defmacro qsDot26432

— defmacro qsDot26432 —

```
(defmacro qsDot26432 (a1 b1 a2 b2)
  `(qsMulAdd6432 ,a1 ,b1 (qsMul6432 ,a2 ,b2)))
```

———-

41.34.7 defmacro qsDot2Mod6432

— defmacro qsDot2Mod6432 —

```
(defmacro qsDot2Mod6432 (a1 b1 a2 b2 p)
  `(qsMod6432 (qsDot26432 ,a1 ,b1 ,a2 ,b2) ,p))
```

———-

41.35 Void

41.35.1 defun voidValue

— defun voidValue —

```
(defun |voidValue| () "()")
```

———-

Chapter 42

OpenMath

42.1 A Technical Overview

OpenMath[Dewa] 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.

42.1.1 The OpenMath Architecture

The OpenMath representation of a mathematical structure is referred to as an OpenMath object. This is an abstract structure which is represented concretely via an OpenMath encoding. These encoded objects are what an OpenMath application would read and write, and in practice the OpenMath objects themselves almost never exist, except on paper. The advantage of this is that OpenMath is not tied to any one underlying mechanism: in the past we have used functional, SGML and binary encodings. The current favourite is XML, as described below, and we will tend to use XML notation when describing OpenMath objects

(even though strictly speaking the XML representation is an encoding). OpenMath Objects

Formally, an OpenMath object is a labelled tree whose leaves are the basic OpenMath objects integers, IEEE double precision floats, unicode strings, byte arrays, variables or symbols. Of these, symbols are the most interesting since they consist of a name and a reference to a definition in an external document called a content dictionary (or CD). Using XML notation where the element name OMS indicates an OpenMath symbol, the following:

```
<OMS name="sin" cd="transc1"/>
```

represents the usual sine function, as defined in the CD "transc1". A basic OpenMath object is an OpenMath object, although its XML representation will be:

```
<OMOBJ>
  <OMS name="sin" cd="transc1"/>
</OMOBJ>
```

OpenMath objects can be built up recursively in a number of ways. The simplest is function application, for example the expression sin(x) can be represented by the XML:

```
<OMOBJ>
  <OMA>
    <OMS name="sin" cd="transc1"/>
    <OMV name="x"/>
  </OMA>
</OMOBJ>
```

where OMV introduces a variable and OMA is the application element. Another straightforward method is attribution which as the name suggests can be used to add additional information (for example "the AXIOM command which generated me was ...") to an object without altering its fundamental meaning. More interesting are binding objects which are used to represent an expression containing bound variables, for example:

```
<OMOBJ>
  <OMA>
    <OMS cd="calculus1" name="int"/>
    <OMS cd="transc1" name="sin"/>
  </OMA>
</OMOBJ>
```

represents the integral of the sin function, but the encoding:

```
<OMOBJ>
  <OMA>
    <OMS cd="calculus1" name="int"/>
    <OMBIND>
      <OMS cd="fns1" name="lambda"/>
      <OMBVAR> <OMV name="x"/> </OMBVAR>
      <OMA>
        <OMS name="sin" cd="transc1"/>
        <OMV name="x"/>
      </OMA>
    </OMBIND>
  </OMA>
</OMOBJ>
```

represents sin(x)dx. This may appear overly complicated but it is useful, for example when searching in a database for expressions which match sin(y)dy . The definition of a symbol in the CD specifies whether or not it may be used to bind variables, which is why

```
<OMS cd="calculus1" name="int"/>
```

cannot be used as a binding symbol.

The final kind of OpenMath object is an error which is built up from a symbol describing the error and a sequence of OpenMath objects. For example:

```
<OMOBJ>
  <OME>
    <OMS name="unexpected_symbol" cd="error1">
    <OMS name="sine" cd="transc1">
  </OME>
</OMOBJ>
```

represents the error which might be generated when an application sees a symbol it doesn't recognise from a CD it thought it knew about.

42.1.2 OpenMath Encodings

We have already seen some examples of the XML encoding, but it is by no means the only encoding. In the past there was a functional encoding (which looked like Lisp) and an SGML encoding which evolved into the current XML. Both of these are now obsolete, but there is still a binary encoding described in the standard , which is much more compact than the XML one.

In fact the XML encoding is not 100% XML. When XML was in its infancy the developers of OpenMath realised that it might become significant and decided to add some XML-like features to the SGML encoding so that an an OpenMath object could be encoded as valid XML. Thus it is currently the case that any well-formed OpenMath object encoded using the XML encoding as described in the standard is a valid XML document. However, if one uses standard XML tools to generate an OpenMath object in the XML encoding from the DTD given in chapter 4 of the standard, it is possible that the result will not be valid OpenMath, although in practice this is highly unlikely. To cover all the possibilities allowed by XML would make it much more complicated to write an application to read any OpenMath object from scratch. Whether to adopt XML completely remains a hot topic of debate within the OpenMath community!

Generally speaking, it is not intended that the existing encodings should be readable by a human user or writable by hand. It is desirable that they be compact and it is also desirable that they be linear, but neither of these is a requirement. It is a property of encodings that it is possible to convert between them with no loss of information.

42.1.3 Content Dictionaries

Content Dictionaries (or CDs for short) are the most important, and the most interesting, aspect of OpenMath because they define the meaning of the objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in an XML format. Defining the meaning of a symbol is not a trivial task, and even referring to well-known references can be fraught with pitfalls Formal definitions and properties can be very useful but time-consuming to produce and verbose, not to mention difficult to get right. A symbol definition in an OpenMath CD consists of the following pieces of information:

```
the symbol name;
a description in plain text;
```

```
    optionally, a set of this symbol's properties in plain text
       (Commented Mathematical Properties, or CMPs);
    optionally, a set of this symbol's properties encoded in OpenMath
       (Formal Mathematical Properties, or FMPs);
    optionally, one or more examples of its use (encoded in OpenMath).
```

In practice the CMPs and FMPs can come as pairs, and often serve in the place of examples.

A very simple instance of a CD definition is:

```
<CDDefinition>
<Name> log </Name>

<Description>
This symbol represents a binary log function; the first argument is
the base, to which the second argument is log'ed.
It is defined in Abramowitz and Stegun, Handbook of Mathematical
Functions, section 4.1
</Description>
<CMP>
  a^b = c implies log_a c = b
</CMP>
<FMP>
  <OMOBJ>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
        <OMS cd="relation1" name="eq"/>

        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMV name="a"/>
          <OMV name="b"/>
        </OMA>
        <OMV name="c"/>
      </OMA>
      <OMA>
        <OMS cd="relation1" name="eq"/>

        <OMA>
          <OMS cd="transc1" name="log"/>
          <OMV name="a"/>
          <OMV name="c"/>
        </OMA>
        <OMV name="b"/>
      </OMA>
    </OMA>
  </OMOBJ>

</FMP>

<Example>
log 100 to base 10 (which is 2).
<OMOBJ>
  <OMA>
    <OMS cd="transc1" name="log"/>
```

```
    <OMF dec="10"/>
    <OMF dec="100"/>
  </OMA>
</OMOBJ>
</Example>

</CDDefinition>
```

Another example would be to print the list

```
   [ 1, 1/2 ]
```

as

```
<OMOBJ>
 <OMA>
  <OMS cd="list1" name="list"/>
  <OMI>1</OMI>
  <OMA>
   <OMS cd="nums1" name="rational"/>
   <OMI>1</OMI>
   <OMI>2</OMI>
  </OMA>
 </OMA>
</OMOBJ>
```

This provides a symbol to represent the log function by giving a pointer to a standard reference book. It provides the property that:

$$a^b = c \rightarrow log_a(c) = b$$

both as plain text and as OpenMath, and also gives an example of how the symbol is used.

CDs usually consist of related symbols and collections of related CDs can be grouped together, for convenience, as CD Groups. One very important CD Group is that corresponding to the content part of MathML.

It is possible to associate extra information with CDs, in particular type information. Since there are many type systems available, each of which has its own strengths and advocates, the OpenMath community does not mandate any single system. Simple signatures can be encoded using the Simple Type System, while more formal definitions are possible using the Extended Calculus of Constructorss. Other associated information can include style sheets for rendering OpenMath symbols in MathML, and mathematical definitions to be used by formal logic systems.

Given the evolutionary nature of mathematics, it is clear that the set of CDs should be forever growing and never complete. Currently there are CDs for high-school mathematics, linear algebra, polynomials and group theory to name a few, and new contributions are always welcome. There is no requirement that applications use the standard set of CDs and it is often very useful to design a "private" CD for a specific purpose.

42.1.4 OpenMath in Action

There is no definitive way in which OpenMath should be used, as the protocol has been designed to be as flexible as possible. Nevertheless many OpenMath applications share common characteristics which we shall discuss here.

Suppose that we wish to have two applications communicating by sending OpenMath objects to each other, e.g. a client program and a computational server. It is unlikely that the internal data structures used by the applications will be OpenMath, and so translation between the internal representations and OpenMath (almost certainly OpenMath encodings rather than objects) will have to take place. The piece of software which does this is usually referred to as a phrase-book.

It is possible to write a generic phrase-book which can handle any piece of OpenMath, but applications where this makes sense are few and far between. In practice an OpenMath phrase book will usually only handle a fixed set of CDs (and hence a fixed set of symbols). What "handle" means will vary from case to case: a computer algebra system will usually try and evaluate its input and return a result or an error, while a typesetter will print its input according to some rendering rules and not return anything. OpenMath carefully avoids defining what the "right" behaviour is in a given circumstance, and leaves that up to the phrase-book writer. Indeed it is quite possible that a piece of software could have multiple phrase-books associated with it for different purposes. OpenMath symbols should not be regarded as verbs since they are used to construct objects rather than to send commands, and the presence of both nouns and verbs in a CD (e.g. "integral" and "integrate") is strongly discouraged.

Writing a phrase-book may be non-trivial, and requires an understanding of the semantics of the underlying software. An OpenMath object may not map directly into a private object and vice-versa, for example in some systems a rational number might have to be represented by a float, or a sparse matrix by a dense one.

The OpenMath standard includes a section on compliance, which describes the behaviour of an OpenMath application when certain errors occur. It also insists that all compliant software has the capability to use the XML encoding, to guarantee a degree of interoperability. This is an area where the standard is expected to evolve as more OpenMath applications become available.

42.2 Technical Details

This chapter describes the Axiom implementation of the OpenMath project[Dalm97] at INRIA. 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.

42.3 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.

42.4 OpenMath Expressions

42.4.1 Expressions

The library understands the following kinds of basic OpenMath expressions:

- integers
- double precision floating-point numbers (64 bits, following IEEE 754)
- byte arrays
- character strings
- symbols
- variables

and the four kinds of constructions:

- applications $e_0(e_1, \ldots e_n)$
- errors $s(e_1, \ldots e_n)$
- binders $e_1, (v_1, \ldots v_n), e_2$
- attributed expressions $[s_1 e_1, \ldots s_n e_n]e$

where e_i are OpenMath expressions, v_i are OpenMath variables and s and s_i are OpenMath symbols.

42.4.2 Symbols

Symbols are constructed from a content dictionary (abbreviated as CD in the sequel) and a name. A content dictionary is identified by its name. The API permits the creation of any symbol in any content dictionary: there is nothing that prevents creating symbols that do not belong to a known CD.

42.4.3 Encoding and Decoding OpenMath Expressions

An OpenMath object is encoded as a sequence of bytes that is read and written sequentially. The library views this sequence as a stream of tokens. Expressions are linearized in a way that looks like Lisp with typed parenthesis. For example, the linearization of the application of S to $E_1 \ldots E_n$ is:

- indicating that this is an application (a "begin application" token)
- linearizing S
- linearizing E_1,... E_n
- indicating that all arguments have been given (an "end application" token)

The other constructions are linearized the same way (each one with its own begin and end tokens). Note that there is no explicit arity indication so that we don't have to introduce a special mechanism when we don't know beforehand how many arguments there are.

To give attributes to an expression, the attributes and their associated values are put before the expression. To give the attributes a_i with values v_i (where a_i are symbols and v_i are OpenMath expressions) to an expression E the process is:

- put a "begin attributed expression" token
- put a "begin attribute pairs" token
- put the symbol a_1 followed by the linearization of v_1 *etc*
- put an "end attribute pairs" token
- linearize E
- put an "end attributed expression" token

Decoding is done by first querying the type of the next OpenMath token and then invoking the right function to get this particular kind of token.

42.5 Big Integers

The library supports big integers that can potentially be given in various formats. The `OMBigIntType` describes the different possible formats.

```
typedef enum OMbigIntType {
  OMBIunknown = 0,  /* this is base 10, digits in normal order */
  OMBIbase10        /* this is base 16, digits in normal order (MSB) */
  OMbigIntBase16
} OMbigIntType;
```

42.6 Functions Dealing with OpenMath Devices

OpenMath expressions are read and written through *devices*. Basically, an OpenMath device has an associated encoding and an I/O method. There are basically two encodings defined and implemented. The first one is a human readable and writable one that can be used for example as the encoding for sending OpenMath objects via e-mail or storing OpenMath objects to files. This encoding is SGML compatible in the sense that it can be used to represent OpenMath objects in SGML texts. I has an XML variant. The second encoding is a binary one that can be used when compactness and speed of encoding and decoding is important. The encodings are defined by the `OMencodingType` type which is an enumerated type defined as

```
typedef enum OMencodingType
   { OMencodingUnknown,
     OMencodingBinary,
     OMencodingSGML,
     OMencodingXML} OMencodingType;
```

`OMencodingUnknown` is to be used when creating a device that does not know which kind of encoding will be used. It must be used only for input devices.

A device is created with the following function, given an encoding and an appropriate I/O method:

- `OMdev OMmakeDevice(OMencodingType encoding, OMIO IO)`

Devices are closed with the following function

- `void OMcloseDevice(OMdev dev)`

Whether a device could be used both for reading and writing is entirely dependent on its I/O method.

The user can define its own I/O method as a function returning an `OMIO` object. This could enable him, for example, to use an existing transport protocol to exchange OpenMath expressions or to implement cut-and-paste of OpenMath expression by writing I/O structures that input and output to strings. The I/O section describes the available I/O structures in the library.

An `OMdev` object is a pointer to a structure that contains a lot of state. Almost all functions taking an `OMdev` object modify it. Likewise, an `OMIO` object carries a lot of state.

42.7 Functions to Write OpenMath Expressions to Devices

42.7.1 Beginning and Ending Objects

The following two functions mark the beginning and end of an OpenMath object.

- `OMstatus OMputObject(OMdev dev)`
- `OMstatus OMputEndObject(OMdev dev)`

These functions should be called before and after an OpenMath object in constructed in a device. In particular, the `OMputEndObject` function insures that the object has been completely written if any buffering was used.

42.7.2 Writing Basic Objects

Basic OpenMath objects are written using these functions:

- `OMstatus OMputInt32(OMdev dev, int n)`
- `OMstatus OMputBigInt(OMdev dev, const char *data, int len, int sign, OMbigIntType format)`
- `OMstatus OMputFloat64(OMdev dev, double *f)`
- `OMstatus OMputByteArray(OMdev dev, const char *data, int len)`
- `OMstatus OMputString(OMdev dev, const char *s)`
- `OMstatus OMputVar(OMdev dev, const char *name)`
- `OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name)`

The `char *` arguments of `OMputString`, `OMputVar` and `OMputSymbol` are null-terminated strings. There are other functions that accept non null-terminated arrays of characters with their length. These are

- `OMstatus OMputStringN(OMdev dev, const char *str, int len)`
- `OMstatus OMputVarN(OMdev dev, const char *var, int len)`
- `OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen,`

```
                                const char *name, int nlen)
```

The format for the `data` argument of the `OMputBigInt` function is given by `format`. When `format` is `OMBIbase10`, it is the sequence of character of its base 10 representation without sign (most significant digit first). The sign of the big integer is given by the `sign` argument that should be an integer greater or equal to zero for a positive integer and less than zero for a negative one. For example, the following line outputs the value of 20! to `dev`:

```
OMputBigInt(dev, "2652528598121910586363084800000000", 33, 1, OMBIbase10);
```

42.7.3 Writing Structured Objects

The following functions are used to mark the beginning and end of the structured objects. They should be called in nested pairs, correctly bracketed:

- `OMstatus OMputApp(OMdev dev)`
- `OMstatus OMputEndApp(OMdev dev)`
- `OMstatus OMputAttr(OMdev dev)`
- `OMstatus OMputEndAttr(OMdev dev)`
- `OMstatus OMputBind(OMdev dev)`
- `OMstatus OMputEndBind(OMdev dev)`
- `OMstatus OMputBVar(OMdev dev)`
- `OMstatus OMputEndBVar(OMdev dev)`
- `OMstatus OMputAtp(OMdev dev)`
- `OMstatus OMputEndAtp(OMdev dev)`
- `OMstatus OMputError(OMdev dev)`
- `OMstatus OMputEndError(OMdev dev)`

Here is an example showing how to use these functions to output $\sin x + y$, where x and y are represented as variables and sin is the symbol whose name is `sin` in the `Basic` content dictionary. This can be done using the following sequence:

```
OMputObject(dev);
OMputApp(dev);
  OMputSymbol(dev, "Basic", "sin");
  OMputApp(dev)
    OMputSymbol(dev, "Basic", "+");
    OMputVar(dev, "x");
    OMputVar(dev, "y");
  OMputEndApp(dev);
OMputEndApp(dev);
OMputEndObject(dev);
```

42.8 Functions to Extract OpenMath Expressions from Devices

42.8.1 Testing the type of the current token

The first step in decoding an expression from a device is to call the `OMgetType` function

- `OMstatus OMgetType(OMdev dev, OMtokenType *type)`

so that the correct function can be called to recover the current token.

`OMgetType` returns via its `type` argument an `OMtokenType` object indicating the type of the next object to be read from the device. `OMtokenType` is an enumerated type defined as

```
typedef enum OMtokenType {
  OMtokenUnknown, /* error catching trick */
  OMtokenInt32,
  OMtokenBigInt,
  OMtokenFloat64,
  OMtokenByteArray,
  OMtokenVar,
  OMtokenString,
  OMtokenSymbol,
  OMtokenComment,
  OMtokenApp,    OMtokenEndApp,
  OMtokenAttr,   OMtokenEndAttr,
  OMtokenAtp,    OMtokenEndAtp,
  OMtokenError,  OMtokenEndError,
  OMtokenObject, OMtokenEndObject,
  OMtokenBind,   OMtokenEndBind,
  OMtokenBVar,   OMtokenEndBVar,
} OMtokenType;
```

Note that the type of the current token can be tested multiple times. Two successive calls to `OMgetType` will always return the same result if no other `OMget...` function was called in between.

42.8.2 Extracting the current token

The following functions are used to read the basic OpenMath objects from devices:

- `OMstatus OMgetInt32(OMdev dev, int *i)`
- `OMstatus OMgetFloat64(OMdev dev, double *d)`
- `OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)`
- `OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)`
- `OMstatus OMgetByteArray(OMdev dev, char **data, int *len)`
- `OMstatus OMgetByteArrayN(OMdev dev, char *data, int len)`
- `OMstatus OMgetString(OMdev dev, char **str)`
- `OMstatus OMgetStringN(OMdev dev, char *str, int len)`

- `OMstatus OMgetVar(OMdev dev, char **var)`
- `OMstatus OMgetVarN(OMdev dev, char *var, int len)`
- `OMstatus OMgetSymbol(OMdev dev, char **cd, char **name)`
- `OMstatus OMgetSymbolN(OMdev dev,char *cd,int clen,char *name,int nlen)`

The functions that return variable size data exist in two versions. A simple version that does the necessary memory allocation itself (using `OMmalloc`) and a version (suffixed with `N`) that lets the user do the allocation itself. The size of the needed area can be determined with the following function:

- `int OMgetLength(OMdev dev)` returns the length of the next object.

that works for big integers, byte arrays, strings and variables. For symbols, the following function returns both the length of the content dictionary name and the length of the symbol name:

- `OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen)`

When the current token does not carry any data i.e. when `OMgetType` returns a marker, i.e. one of:

- `OMtokenApp`,
- `OMtokenEndApp`,
- `OMtokenAttr`,
- `OM tokenEndAttr`,
- `OMtokenAtp`,
- `OMtokenEndAtp`,
- `OMtokenError`,
- `OMtokenEndError`,
- `OMtokenObject`,
- `OMtokenEndObject`,
- `OMtokenBind`,
- `OMtokenEndBind`,
- `OMtokenBVar`
- `OMtokenEndBVar`

it is necessary to call the correct function to remove the marker. The available functions are

- `OMstatus OMgetObject(OMdev dev)`
- `OMstatus OMgetEndObject(OMdev dev)`
- `OMstatus OMgetApp(OMdev dev)`
- `OMstatus OMgetEndApp(OMdev dev)`
- `OMstatus OMgetAttr(OMdev dev)`
- `OMstatus OMgetEndAttr(OMdev dev)`
- `OMstatus OMgetAtp(OMdev dev)`
- `OMstatus OMgetEndAtp(OMdev dev)`

- `OMstatus OMgetBind(OMdev dev)`
- `OMstatus OMgetEndBind(OMdev dev)`
- `OMstatus OMgetBVar(OMdev dev)`
- `OMstatus OMgetEndBVar(OMdev dev)`
- `OMstatus OMgetError(OMdev dev)`
- `OMstatus OMgetEndError(OMdev dev)`

All the previous functions return `OMsuccess` when they succeed. When they return something else, there has been a problem such as calling the wrong function (`OMgetApp` when there is not a "beginning of application" mark) or a system error.

The sequence of calls to read an expression is thus completely similar (if we omit the calls to `OMgetType`) to the sequence of calls to write the expression. For example, the previous expression ($\sin x + y$) can be recovered via the sequence:

```
OMgetObject(dev);
OMgetApp(dev);
  OMgetSymbol(dev, ...);
  OMgetApp(dev);
    OMgetSymbol(dev, ...);
    OMgetVar(dev, ...);
    OMgetVar(dev, ...);
  OMgetEndApp(dev);
OMgetEndApp(dev);
OMgetEndObject(dev);
```

`OMgetInt32(OMdev dev, int *i)` returns the integer through its `i` argument.

`OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)`

returns the data corresponding to the big integer in `data`, its length in `len`, its sign in `sign` and its format in `fmt`.

`OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)`

copies the data corresponding to the big integer in `data` buffer that should be (at least) `len` characters long. The sign and format are returned in the `sign` and `fmt` arguments.

`OMgetByteArray(OMdev dev, char **data, int *len)` returns the byte array through its `data` argument. Its length is returned via the `len` argument.

`OMgetByteArrayN(OMdev dev, char *data, int len)` copies the byte array in the `data` buffer that should be (at least) `len` characters long.

`OMgetString(OMdev dev, char **str)` returns the string trough its `str` argument.

`OMgetStringN(OMdev dev, char *str, int len)` copies the string in the `str` buffer whose length should be (at least) `len`. If `len` is greater than the actual length of the string, a null character is added at the end of `str`.

`OMgetVar(OMdev dev, char **var)` returns the name of the variable (as a null-terminated string) in its `var` argument

`OMgetVarN(OMdev dev, char *var, int len)` copies the name of the variable in the `var` buffer, whose length should be (at least) `len`. If `len` is greater than the actual length of the variable name, a null character is added at the end of `var`.

`OMgetSymbol(OMdev dev, char **cd, char **name)` returns the content dictionary and the name of the symbol through the `cd` and `name` arguments.

`OMgetSymbolN(OMdev dev, char *cd, int clen, char *name, int nlen)` copies the content dictionary and the name of the symbols in the `cd` and `name` buffers. `cd` should be at least `clen` character long and `name` should be at least `nlen` long. When there is enough room (based on `clen` or `nlen`) a null character is added after the last character of the name (`cd` or `name`).

42.9 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.

42.10 I/O Functions for Devices

We provide four functions that produce `OMIO` objects for devices. These functions provide I/O through the `stdio` library (on `FILE` object), file descriptors and character strings.

- `OMIO OMmakeIOFile(FILE *f)` associates the device with the file pointer `f`.
- `OMIO OMmakeIOfd(int fd)` associates the device with the file descriptor `fd`.
- `OMIO OMmakeIOHandle(HANDLE handle)` associates the device with a file handle *Windows specific version of `OMmakeIOfd()`.`fd`.
- `OMIO OMmakeIOString(char **s)` associates the device with a string.

For example, the following code opens a device that reads from standard input:

```
dev = OMmakeDevice(OMencodingSGML, OMmakeIOFile(stdin));
```

The `OMmakeIOString` builds an input device that reads from a string or an output device that writes to a string. For input, `s` must point to a character string (null terminated). For output, `s` will point to a string allocated by the library (note that the string `s` points to can be reallocated by the library).

42.11 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.

42.11.1 Functions to Initiate an `OMconn`

The functions we provide can be divided in two classes. The first one simply establishes an interprocess communication using IP addresses. The second one provides functions that can be used to launch a server. The addresses used are then generated by the library.

Simple Connections Functions

The following functions allow a client OpenMath application to contact an OpenMath server at a specified address:

- `OMstatus OMconnTCP(OMconn conn, char *machine, int port)`
- `OMstatus OMconnUnix(OMconn conn, char *file)`

These functions first physically establish the connection. Then, they enter negotiation with the server (they send the first message). When they return, the negotiation is finished and the devices in the `conn` argument are ready.

On the server side, the following functions provide bindings at specified addresses and take care of the negotiation:

- `OMstatus OMbindTCP(OMconn conn, int port)`
- `OMstatus OMbindUnix(OMconn conn, char *file)`

All four the previous functions block until the connection is established (and negotiation is over) or the timeout of the `conn` argument is reached.

The following function returns the file descriptor associated with a device. This is intended to be used when there is a need to poll the device (through the `select` or `poll` system calls).

- `OMdeviceFd(OMdev dev)`

Functions that Launch Servers

These functions provide the same functionalities for launching a server that were provided in the ASAP library.

In this model, the client calls `OMlaunch` with a machine name `mach` and a string `cmd` that is executed via `rsh` on machine `mach` as a shell command line. This command is supposed to launch the server program. The command is executed in an environment (in the UNIX sense) where some variables are associated with an address on the machine that runs the client. The server can then connect to the client with the `OMserveClient` function.

If the machine name is `localhost`, the command is started on the same machine (without calling `rsh`).

- `OMstatus OMlaunchEnv(OMconn conn, char *machine, char *command, char *env)`
- `OMstatus OMlaunch(OMconn conn, char *machine, char *command)`
- `OMstatus OMserveClient(OMconn conn)`

The environment variables sent to the server (launched program) are `OM_CALLER_UNIX_SOCKET` (when a local connection is required) and `OM_CALLER_MACHINE` and `OM_CALLER_PORT` (for internet connections).

The `OMlaunchEnv` function enables the command to be run with a particular environment (in the UNIX sense). For example to run a `plot` server on the `kama` machine, we could use a piece of code such as

```
conn = OMmakeConn(2000);
OMlaunchEnv(conn, "kama", "plot", "DISPLAY=rati:0 PATH=/users/bin");
```

Termination

- `OMstatus OMconnClose(OMconn conn)`

42.12 Parameters

The library internally uses three functions that can be supplied by the user.

- `extern void *(*OMmalloc) (size_t size)`
- `extern void *(*OMrealloc) (void *ptr, size_t size)`
- `extern void (*OMfree) (void *ptr)`
- `OMmalloc` is used for all memory allocations in the library. The default value is the `malloc` function.
- `OMfree` is used for deallocations. The default value is the `free` function.
- `OMfatal` is invoked when a fatal error is detected in the library (for example when memory allocation failed or when an inconsistency is detected in the library code data structures). The default value just does an `exit`.

`OMfatal` is declared as `extern void (*OMfatal)(OMstatus status)`. All memory allocations and deallocations in the library are done through the `OMmalloc` and `OMfree` functions.

42.13 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.

42.14 The OM.h header file

```
#ifndef __OM_h__
#define __OM_h__

/*
 *                      All types used through API.
 */

/* These types are anonymized by the mean of a generic pointer.
 * You should not allocate or dereference objects of these types.
 * API (hopefully) provides you with all needed methods.
 * If you find any that are not included, please refer to
 * us rather than using private structures.
 * ie: If you need to do something like
 *   malloc(sizeof(OMdevStruct));
 * or
 *   OMdevStruct * pDev;
 *   pDev->anyField = something;
 * this probably means we need to discuss your problem.
 */

/* A device is an abstraction for put/get of OpenMath tokens */
typedef struct OMdevStruct *OMdev;

/* IO is a device field, (the physical IO channel) */
typedef struct OMIOStruct *OMIO;


/* Error status that may be returned
 */
typedef enum OMstatus {
  /* Last call was successful. */
  OMsuccess = 0,
  /* Last call failed for some undetermined reason. */
  OMfailed = 1,
  /* Last call failed for memory reasons. */
  OMnoMem,
```

```
  /* Last call failed during some system call. */
  OMerrorSys,
  /* Last call to some OMget* function failed due to an unexpected EOF
     on input IO. */
  OMemptyIO,
  /* Last call to some OMget* function failed because there is no more
     token on device. */
  OMnoMoreToken,
  /* Last call to some OMget* function timeouted. */
  OMtimeoutedRead,
  /* Last call to some OMget* function failed due to malformed input.
    (this error covers all low level lexical or syntactic problems). */
  OMmalformedInput,
  /* Last call to OMbindTCP failed because address is already in use
     (EADDRINUSE). */
  OMaddrInUse,
  /* Last call to OMconnTCP failed to set connection. */
  OMconnectFailed,
  /* Last call trigered some not (yet) implemented code in this lib. */
  OMnotImplemented,
  /* Last call caused some internal trouble. */
  OMinternalError
} OMstatus;

/* All OpenMath token kinds are identified by one of these types.
 * Values given in this enum have been chosen to:
 * - avoid  conflicts whith specific XML characters
 *   to help automatic detection of encoding type.
 *   (no: '\t'(9) '\r'(13) '\n'(10) '<'(60) or ' '(32))
 * - keep some bits (3) available for special encodings purpose
 *   (eg: sharing or big len flags in binary encoding)
 */
typedef enum OMtokenType {
  OMtokenUnknown = 0, /* error catching trick */
  OMtokenInt32 = 1,
  OMtokenBigInt = 2,
  OMtokenFloat64 = 3,
  OMtokenByteArray = 4,
  OMtokenVar = 5,
  OMtokenString = 6,
  OMtokenWCString = 7,
  OMtokenSymbol = 8,
  OMtokenComment = 15,
  OMtokenApp = 16, OMtokenEndApp = 17,
  OMtokenAttr = 18, OMtokenEndAttr = 19,
  OMtokenAtp = 20, OMtokenEndAtp = 21,
  OMtokenError = 22, OMtokenEndError = 23,
  OMtokenObject = 24, OMtokenEndObject = 25,
  OMtokenBind = 26,   OMtokenEndBind = 27,
  OMtokenBVar = 28,   OMtokenEndBVar = 29
} OMtokenType;

typedef enum OMbigIntType {
  OMbigIntUnknown = 0,
```

```
  /* this is base 10, digits in normal order (MSB) */
  OMbigIntBase10,
  /* this is base 16, digits in normal order (MSB) */
  OMbigIntBase16
} OMbigIntType;

/* Encodings should not be "user visible"
 * We thus refer to encoding as "symbolic constants" from this enum type. */
typedef enum OMencodingType {
  /* You may set an input stream to "unknown encoding".
   * By doing this, you let library auto detect the
   * encoding type of the device during first token input.*/
  OMencodingUnknown = 0,
  /* Binary encoding, more compact than XML one. */
  OMencodingBinary,
  /* XML-like encoding, human readable. */
  OMencodingXML,
} OMencodingType;

/* This is a portable equivalent to wchar_t for unicode strings */
typedef unsigned short OMUCS2;

/* Replacment for lacking C bools */
typedef unsigned char OMbool;
#define OMfalse (0)
#define OMtrue  (1)

/*
 *                        Some global variables
 */

/* Version of this lib (eg: "1.0") */
extern const char *OMlibVersion;

/* Some textual information about this lib (eg: "debug is on" */
extern const char *OMlibInfo;

/* These pointers allow you to redefine memory managment functions
   used in lib. */
extern void *(*OMmalloc) (size_t size);
extern void *(*OMrealloc) (void *ptr, size_t size);
extern void (*OMfree) (void *ptr);

/* If set, this function will be called by OMfatal, thus you may use it for
   error handling (by default it is set to exit()) */
extern void (*OMfatal) (OMstatus status);



/* for C++ includes */
#ifdef  __cplusplus
#define OMbeginPrototypes   extern "C" {
#define OMendPrototypes     }
#else /*__cplusplus */
```

```
#define OMbeginPrototypes
#define OMendPrototypes
#endif /*__cplusplus */

/*
 *                       Prototypes of OpenMath API
 */

/* Prototypes that are spread along all headers are repeated here.
 * - This should ease the API users.
 *   (docs are fine but source is always the ultimate help)
 * - This allow a cleaner embeding of library
 *   (no need to install all .h! just take this one and the .a)
 */
OMbeginPrototypes
#ifndef OM_DEV
/* this part is automaticaly updated, do NOT edit below */
/** Prototypes */
/* OMPut* functions.
 *    They all take a device <dev> to put token to.
 *    Some of them need more parameters to define the token content.
 *    They are thoroughly documented in OpenMath Specification shiped
 *    with the library.
 * return: a status that reflect the operation success.
 */
extern OMstatus OMputInt32(OMdev dev, int n);
extern OMstatus OMputFloat64(OMdev dev, double *d);
extern OMstatus OMputBigInt(OMdev dev, const char *data, int len,
                            int sign, OMbigIntType format);
extern OMstatus OMputByteArray(OMdev dev, const char *data, int len);
/* OMputString*
 *   If you want to output plain 8bits C like strings there is no need
 *   to use the OMputWCString* functions. This one is more efficient
 *   (faster and more compact output for some encodings)
 */
extern OMstatus OMputString(OMdev dev, const char *str);
extern OMstatus OMputStringN(OMdev dev, const char *str, int len);
/* OMputWCString
 *   If you are using wide char strings you need to output them
 *   with that function rather than with OMputString.
 *   (It takes endianess into account)
 */
extern OMstatus OMputWCString(OMdev dev, const OMUCS2 * wcstr);
extern OMstatus OMputVar(OMdev dev, const char *var);
extern OMstatus OMputVarN(OMdev dev, const char *var, int len);
extern OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name);
extern OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen,
                             const char *name, int nlen);
extern OMstatus OMputApp(OMdev dev);
extern OMstatus OMputEndApp(OMdev dev);
extern OMstatus OMputAttr(OMdev dev);
extern OMstatus OMputEndAttr(OMdev dev);
extern OMstatus OMputAtp(OMdev dev);
extern OMstatus OMputEndAtp(OMdev dev);
```

```
extern OMstatus OMputBind(OMdev dev);
extern OMstatus OMputEndBind(OMdev dev);
extern OMstatus OMputBVar(OMdev dev);
extern OMstatus OMputEndBVar(OMdev dev);
extern OMstatus OMputObject(OMdev dev);
extern OMstatus OMputEndObject(OMdev dev);
extern OMstatus OMputError(OMdev dev);
extern OMstatus OMputEndError(OMdev dev);
extern OMstatus OMputComment(OMdev dev, const char *comment);
extern OMstatus OMputCommentN(OMdev dev, const char *comment, int len);
/* OMgetType
 *   Get the type of the current token on device <dev>/
 * dev: device to look at.
 * type: where to store returned type.
 * return: 0 or some error code
 */
extern OMstatus OMgetType(OMdev dev, OMtokenType * type);
/* OMgetLength
 *   Get the current token length.
 * dev: device to read from
 * len: where to put the token length
 *      the last '\0' for string like tokens is not counted
 *      (rem: for WCString it is the number of bytes not the number of
 *       wide chars)
 * return: 0 or some error code
 */
extern OMstatus OMgetLength(OMdev dev, int *len);
/* OMgetSymbolLength
 *   Get the current token (wich is assumed to be a symbol) lengths.
 * dev: device to read from
 * clen: where to put the cd length (not counting the last '\0')
 * nlen: where to put the name length (not counting the last '\0')
 * return: 0 or some error code
 */
extern OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen);
/* OMGet* functions.
 *    They all take a device <dev> to get token from.
 *    Some of them need more parameters to fill with the token content.
 *    They are thoroughly documented in OpenMath Specification shiped with
 *    the library.
 * return: a status that reflect the operation success.
 */
extern OMstatus OMgetInt32(OMdev dev, int *i);
extern OMstatus OMgetFloat64(OMdev dev, double *d);
extern OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign,
                            OMbigIntType * format);
extern OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign,
                             OMbigIntType * format);
extern OMstatus OMgetByteArray(OMdev dev, char **data, int *len);
extern OMstatus OMgetByteArrayN(OMdev dev, char *data, int len);
/* OMgetString*
 *   Beware! You are not suposed to use these functions unless you know
 *   for sure you are reading plain 8bits strings.
 *   Thus it is here only for speed/space consideration in very
```

```
 *   specific applications.
 *   If input is a 16 bit char string and you read it with these
 *   functions you will lose the 8 most significant bits of each char.
 *   You should rather refer to OMgetWCString* functions.
 */
extern OMstatus OMgetString(OMdev dev, char **str);
extern OMstatus OMgetStringN(OMdev dev, char *str, int len);
/* OMgetWCString*
 *   These functions return 16 bits wide strings. (regardless input
 *   was done in 8 or 16 bits mode).
 *   Thus, most if not all applications should use these functions
 *   preferably to OMgetString*.
 */
extern OMstatus OMgetWCString(OMdev dev, OMUCS2 ** wcstr);
/* BEWARE: the <len> is suposed to be the length in bytes for the
 * preallocated buffer <wcstr> (not the length in number of wide chars)
 */
extern OMstatus OMgetWCStringN(OMdev dev, OMUCS2 * wcstr, int len);
extern OMstatus OMgetVar(OMdev dev, char **var);
extern OMstatus OMgetVarN(OMdev dev, char *var, int len);
extern OMstatus OMgetSymbol(OMdev dev, char **cd, char **name);
extern OMstatus OMgetSymbolN(OMdev dev, char *cd, int clen, char *name,
                             int nlen);
extern OMstatus OMgetApp(OMdev dev);
extern OMstatus OMgetEndApp(OMdev dev);
extern OMstatus OMgetAttr(OMdev dev);
extern OMstatus OMgetEndAttr(OMdev dev);
extern OMstatus OMgetAtp(OMdev dev);
extern OMstatus OMgetEndAtp(OMdev dev);
extern OMstatus OMgetBind(OMdev dev);
extern OMstatus OMgetEndBind(OMdev dev);
extern OMstatus OMgetBVar(OMdev dev);
extern OMstatus OMgetEndBVar(OMdev dev);
extern OMstatus OMgetObject(OMdev dev);
extern OMstatus OMgetEndObject(OMdev dev);
extern OMstatus OMgetError(OMdev dev);
extern OMstatus OMgetEndError(OMdev dev);
extern OMstatus OMgetComment(OMdev dev, char **comment);
extern OMstatus OMgetCommentN(OMdev dev, char *comment, int len);
/* OMbeginObject
 *   Must be called before every new OpenMath object put.
 *   (Not before every token!)
 * dev: device where new object is to be put.
 * return: status describing operation success
 */
extern OMstatus OMbeginObject(OMdev dev);
/* OMendObject
 *   Must be called after every OpenMath object put.
 *   (Not after every token!)
 * dev: device where object has been put.
 * return: status describing operation success
 */
extern OMstatus OMendObject(OMdev dev);
/* OMignoreComment
```

```
 *   Set behavior of a device concerning comments.
 *   (Comments on an input device may safely be ignored.)
 * dev: device to modify
 * set: If set == OMtrue then device will ignore incoming comments
 *      If set == OMfalse then device will process incoming comments
 *                        like other tokens.
 *      By default comments are ignored.
 *      Whatever is <set> value, output of comments is always done.
 * return: previous value
 */
extern OMbool OMignoreComment(OMdev dev, OMbool set);
/* OMtokenCount
 *   Reports the numbe rof tokens that have been in/output on a device
 * dev: devive to examine
 * inTokenNb: where to store number of input tokens (if not NULL)
 * outTokenNb: where to store number of output tokens (if not NULL)
 */
extern void OMtokenCount(OMdev dev, int *inTokenNb, int *outTokenNb);
/* OMgetDeviceEncoding
 *   Get the currnet encoding used by a device
 * dev: device to examine
 * return: current encoding
 */
extern OMencodingType OMgetDeviceEncoding(OMdev dev);
/* OMsetDeviceEncoding
 *   Set the encoding that will be used on a device
 *   BEWARE: changing encoding on a device that has already been used
 *   for IO is unsafe.
 *   but seting encoding on a new device is safe.
 *   (in some occasions, it is not easy to know which encoding to
 *    use at device creation)
 * dev: device to modify
 * encoding: encoding to use
 */
extern void OMsetDeviceEncoding(OMdev dev, OMencodingType encoding);
/* OMmakeDevice
 *   Create a device from a low level IO
 *   Warning: "IO" should be a "generated" (new) structure as it contains some
 *   state that is private to the device. It is very dangerous for two devices
 *   to share the same "IO" structure.
 * encoding: encoding scheme used by device
 * IO: low level I/O suport for device
 * return: a newly allocated device
 */
extern OMdev OMmakeDevice(OMencodingType encoding, OMIO IO);
/* OMcloseDevice
 *   Close a device previously created with OMmakeDevice
 *   (embedded IO is closed too)
 * dev: device to close
 */
extern void OMcloseDevice(OMdev dev);
/* OMmakeIOFd
 *   Create a low level IO object from a file descriptor.
 *   (May be used on socket for instance.)
```

```
 * fd: file descriptor to wrap into the OpenMath IO object.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOFd(int fd);
/* OMmakeIOFile
 *   Create a low level IO object from a FILE*.
 *   (May be used on stdin for instance.)
 * fd: FILE* to wrap into the OpenMath IO object.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOFile(FILE * f);
/* OMmakeIOString
 *   Create a low level IO object from a string (NUL terminator is not needed).
 *   (May be used for copy/paste for instance.)
 * s: pointer to string to use into the OpenMath IO object.
 *    - In case of input device the string must be NUL terminated.
 *    - In case of output device string may be reallocated
 *      to fit size of outcoming objects.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOString(char **s);
/* OMstatusToString
 *   Convert a status to a human readable string that explain its meaning
 * status: status to explain
 * return: corresponding string
 */
extern char *OMstatusToString(OMstatus status);
/* OMtokenTypeToString
 *   Convert a tokenType to a human readable string
 * ttype: type to convert
 * return: corresponding string
 */
extern char *OMtokenTypeToString(OMtokenType ttype);
/* OMsetVerbosityLevel
 *   When using API some infos may be loged.
 *   This set the required verbosity level.
 * level: level of verbosity.
 *        0 means nothing is nether printed
 *        1 everything is printed (default)
 *        2,... less verbose
 * return: last verbosity level
 */
extern int OMsetVerbosityLevel(int level);
/* OMsetVerbosityOutput
 *   When using API some infos may be loged.
 *   This set the destination for logs.
 * logFile: where to output logs (default is stderr)
 * return: last output
 */
extern FILE *OMsetVerbosityOutput(FILE * logFile);
/* OMlibDynamicInfo
 *   Gather some informations about lib that can't be statically determined.
 *   Complete them with some relevant static infornation too.
 * return: a newly allocated string
```

```
 */
extern char *OMlibDynamicInfo(void);
/*** End Prototypes */
/* end of automaticaly updated part */

#ifdef WIN32
#include "windows.h"

/* OMmakeIOHandle
 *   Create a low level IO object from a widows handle.
 * handle: windows handle to wrap into the OpenMath IO object.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOHandle(HANDLE handle);
extern void OMfreeIOHandle(OMIO io);
#endif

#else /* OM_DEV */
/* The prototypes above are in fact collected from all these .h files */
#include "OMbase.h"
#include "OMdev.h"
#include "OMdevFd.h"
#include "OMdevFile.h"
#include "OMdevString.h"
#include "OMdevHandle.h"
#include "OMencBin.h"
#include "OMencXml.h"
#include "OMmisc.h"
#include "OMutf7.h"
#endif /* OM_DEV */

OMendPrototypes

#endif /* __OM_h__ */
```

42.15 Axiom OpenMath stub functions

These stub functions will eventually be expanded to handle OpenMath. See the OpenMathDevice 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

42.15.1 Axiom specific functions

This is used in OpenMathPackage in Volume 10.4.

```
(read OMdev)              -> LispObject
(listCDs)                 -> List(String)
(listSymbols)             -> List(String)
(supportsCD cd)           -> Boolean
(supportsSymbol cd name)  -> Boolean
```

42.15.2 defun om-Read

Read an OpenMath object from dev.

— **defun om-Read** —

```
(defun om-Read (dev)
 (declare (ignore dev)))
```

———-

42.15.3 defun om-listCDs

Lists all of the CDs supported by Axiom.

— **defun om-listCDs** —

```
(defun om-listCDs ())
```

———-

42.15.4 defun om-listSymbols

Lists all the symbols in CD

— **defun om-listSymbols** —

```
(defun om-listSymbols ())
```

———-

42.15.5 defun om-supportsCD

Return true if Axiom supports this CD.

— **defun om-supportsCD** —

```
(defun om-supportsCD (cd)
 (declare (ignore cd)))
```

———-

42.15.6 defun om-supportsSymbol

— **defun om-supportsSymbol** —

```
(defun om-supportsSymbol (cd name)
 (declare (ignore cd name)))
```

———-

42.15.7 Lisp conversion functions

The lisp conversion functions are:

```
(toDev                 LispObject)                   -> OMdev
(fromDev               OMdev)                        -> LispObject
(toStatus              LispObject)                   -> LispObject
(fromStatus            OMstatus)                     -> LispObject
(toEncodingType        LispObject)                   -> OMencodingType
(fromEncodingType      OMencodingType)               -> LispObject
(toBigNumStr           LispObject)                   -> char *
(fromBigNumStr         char *,int,int, OMbigIntType) -> LispObject
(toConn                LispObject)                   -> OMconn
(fromConn              OMconn)                       -> LispObject
(toCString             LispObject)                   -> char **
(fromCString           char **)                      -> LispObject
(lispStringFromCString LispObject)                   -> LispObject
(cStringFromLispString LispObject)                   -> LispObject
```

42.15.8 defun om-setDevEncoding

This sets the encoding used for reading or writeing OpenMath objects to or from dev to enc.

— **defun om-setDevEncoding** —

```
(defun om-setDevEncoding (dev enc)
 (declare (ignore dev enc)))
```

———-

42.15.9 Device manipulation functions

```
(openFileDev LispObject, ints, ...)              -> LispObject
(openStrDev  LispObject, LispObject, LispObject) -> LispObject
(closeDev    LispObject, LispObject)             -> LispObject
```

42.15.10 defun om-openFileDev

This opens file fname for reading or writing OpenMath objects. The mode can be “r” for read, “w” for write, or ”a” for append.

— **defun om-openFileDev** —

```
(defun om-openFileDev (fname fmode enc)
 (declare (ignore fname fmode enc)))
```

———-

42.15.11 defun om-openStringDev

This opens the string str for reading and writing OpenMath objects in encoding enc.

— **defun om-openStringDev** —

```
(defun om-openStringDev (str enc)
 (declare (ignore str enc)))
```

———-

42.15.12 defun om-closeDev

This closes dev, flushing output if necessary.

— **defun om-closeDev** —

```
(defun om-closeDev (dev)
 (declare (ignore dev)))
```

———-

42.15.13 Connection manipulation functions

These are covered in the OpenMathConnection domain in Volume 10.3.

```
(makeConn      LispObject, LispObject) -> LispObject
(closeConn     LispObject, LispObject) -> LispObject
(getConnInDev  LispObject, LispObject) -> LispObject
(getConnOutDev LispObject, LispObject) -> LispObject
```

42.15.14 defun om-makeConn

— **defun om-makeConn** —

```
(defun om-makeConn (conn)
 (declare (ignore conn)))
```

———-

42.15.15 defun om-closeConn

— **defun om-closeConn** —

```
(defun om-closeConn (conn)
 (declare (ignore conn)))
```

———-

42.15.16 defun om-getConnInDev

— defun om-getConnInDev —

```
(defun om-getConnInDev (conn)
 (declare (ignore conn)))
```

———-

42.15.17 defun om-getConnOutDev

— defun om-getConnOutDev —

```
(defun om-getConnOutDev (conn)
 (declare (ignore conn)))
```

———-

42.15.18 Client/Server functions

These are covered in the OpenMathConnection domain in Volume 10.3. See OMconn.h

```
(bindTCP    LispObject, LispObject, LispObject) -> LispObject
(connectTCP LispObject, int, ...)               -> LispObject
```

42.15.19 defun om-bindTCP

— defun om-bindTCP —

```
(defun om-bindTCP (conn port)
 (declare (ignore conn port)))
```

———-

42.15.20 defun om-connectTCP

— defun om-connectTCP —

```
(defun om-connectTCP (conn host port)
 (declare (ignore conn host port)))
```

42.15.21 Device input/output functions

Most of these functions are in the OpenMathDevice domain in Volume 10.3. The only exception seems to be the om-stringPtrToString and om-stringToStringPtr functions which are called in the domains that export primitives. Currently these are:

- Complex (10.3)
- DoubleFloat (10.3)
- Float (10.3)
- Fraction (10.3)
- Integer (10.3)
- List (10.3)
- SingleInteger (10.3)
- String (10.3)
- Symbol (10.3)
- ExpressionToOpenMath (10.4)
- OpenMathPackage (10.4)

Note that putSymbol2 is not implemented.

```
(getApp           LispObject, LispObject) -> LispObject
(getAtp           LispObject, LispObject) -> LispObject
(getAttr          LispObject, LispObject) -> LispObject
(getBind          LispObject, LispObject) -> LispObject
(getBVar          LispObject, LispObject) -> LispObject
(getByteArray     LispObject, LispObject) -> LispObject
(getEndApp        LispObject, LispObject) -> LispObject
(getEndAtp        LispObject, LispObject) -> LispObject
(getEndAttr       LispObject, LispObject) -> LispObject
(getEndBind       LispObject, LispObject) -> LispObject
(getEndBVar       LispObject, LispObject) -> LispObject
(getEndError      LispObject, LispObject) -> LispObject
(getEndObject     LispObject, LispObject) -> LispObject
(getError         LispObject, LispObject) -> LispObject
(getFloat         LispObject, LispObject) -> LispObject
(getInt           LispObject, LispObject) -> LispObject
(getObject        LispObject, LispObject) -> LispObject
(getString        LispObject, LispObject) -> LispObject
(getSymbol        LispObject, LispObject) -> LispObject
(getType          LispObject, LispObject) -> LispObject
(getVar           LispObject, LispObject) -> LispObject
(putApp           LispObject, LispObject) -> LispObject
(putAtp           LispObject, LispObject) -> LispObject
(putAttr          LispObject, LispObject) -> LispObject
(putBind          LispObject, LispObject) -> LispObject
(putBVar          LispObject, LispObject) -> LispObject
(putByteArray     LispObject, LispObject, LispObject) -> LispObject
(putEndApp        LispObject, LispObject) -> LispObject
```

```
(putEndAtp         LispObject, LispObject) -> LispObject
(putEndAttr        LispObject, LispObject) -> LispObject
(putEndBind        LispObject, LispObject) -> LispObject
(putEndBVar        LispObject, LispObject) -> LispObject
(putEndError       LispObject, LispObject) -> LispObject
(putEndObject      LispObject, LispObject) -> LispObject
(putError          LispObject, LispObject) -> LispObject
(putFloat          LispObject, LispObject, LispObject) -> LispObject
(putInt            LispObject, LispObject, LispObject) -> LispObject
(putObject         LispObject, LispObject) -> LispObject
(putString         LispObject, LispObject, LispObject) -> LispObject
(putSymbol         LispObject, LispObject, LispObject) -> LispObject
(putSymbol2        LispObject, int nargs, ...) -> LispObject
(putVar            LispObject, LispObject, LispObject) -> LispObject
(stringPtrToString LispObject, LispObject) -> LispObject
(stringToStringPtr LispObject, LispObject) -> LispObject
```

42.15.22 defun om-getApp

Reads a begin application token from dev.

— **defun om-getApp** —

```
(defun om-getApp (dev)
 (declare (ignore dev)))
```

———-

42.15.23 defun om-getAtp

Reads a begin attribute pair token from dev.

— **defun om-getAtp** —

```
(defun om-getAtp (dev)
 (declare (ignore dev)))
```

———-

42.15.24 defun om-getAttr

Reads a begin attribute token from dev

— **defun om-getAttr** —

```
(defun om-getAttr (dev)
 (declare (ignore dev)))
```

———-

42.15.25 defun om-getBind

Reads a begin binder token from dev.

— **defun om-getBind** —

```
(defun om-getBind (dev)
 (declare (ignore dev)))
```

———-

42.15.26 defun om-getBVar

Reads a begin bound variable list token from dev.

— **defun om-getBVar** —

```
(defun om-getBVar (dev)
 (declare (ignore dev)))
```

———-

42.15.27 defun om-getByteArray

Reads a byte array from dev.

— **defun om-getByteArray** —

```
(defun om-getByteArray (dev))
```

———-

42.15.28 defun om-getEndApp

Reads an end application token from dev

— **defun om-getEndApp** —

```
(defun om-getEndApp (dev)
 (declare (ignore dev)))
```

———-

42.15.29 defun om-getEndAtp

Reads an end attribute pair token from dev.

— **defun om-getEndAtp** —

```
(defun om-getEndAtp (dev)
 (declare (ignore dev)))
```

———-

42.15.30 defun om-getEndAttr

Reads an end attribute token from dev.

— **defun om-getEndAttr** —

```
(defun om-getEndAttr (dev)
 (declare (ignore dev)))
```

———-

42.15.31 defun om-getEndBind

Reads an end binder token from dev.

— **defun om-getEndBind** —

```
(defun om-getEndBind (dev)
 (declare (ignore dev)))
```

———-

42.15.32 defun om-getEndBVar

Reads an end bound variable list token from dev.

— **defun om-getEndBVar** —

```
(defun om-getEndBVar (dev)
 (declare (ignore dev)))
```

———-

42.15.33 defun om-getEndError

Reads an end error token from dev.

— **defun om-getEndError** —

```
(defun om-getEndError (dev)
 (declare (ignore dev)))
```

———-

42.15.34 defun om-getEndObject

Reads an end object token from dev.

— **defun om-getEndObject** —

```
(defun om-getEndObject (dev)
 (declare (ignore dev)))
```

———-

42.15.35 defun om-getError

Reads a begin error token from dev.

— **defun om-getError** —

```
(defun om-getError (dev)
 (declare (ignore dev)))
```

———-

42.15.36 defun om-getFloat

Reads a float from dev.

— **defun om-getFloat** —

```
(defun om-getFloat (dev)
 (declare (ignore dev)))
```

———-

42.15.37 defun om-getInt

Reads an integer from dev.

— **defun om-getInt** —

```
(defun om-getInt (dev)
 (declare (ignore dev)))
```

———-

42.15.38 defun om-getObject

Reads a begin object token from dev.

— **defun om-getObject** —

```
(defun om-getObject (dev)
 (declare (ignore dev)))
```

———-

42.15.39 defun om-getString

Reads a string from dev.

— **defun om-getString** —

```
(defun om-getString (dev)
 (declare (ignore dev)))
```

———-

42.15.40 defun om-getSymbol

Reads a symbol from dev.

— **defun om-getSymbol** —

```
(defun om-getSymbol (dev)
 (declare (ignore dev)))
```

———-

42.15.41 defun om-getType

Returns the type of the next object on dev.

— **defun om-getType** —

```
(defun om-getType (dev)
 (declare (ignore dev)))
```

———-

42.15.42 defun om-getVar

Reads a variable from dev.

— **defun om-getVar** —

```
(defun om-getVar (dev)
 (declare (ignore dev)))
```

———-

42.15.43 defun om-putApp

Writes a begin application token to dev.

— **defun om-putApp** —

```
(defun om-putApp (dev)
 (declare (ignore dev)))
```

———-

42.15.44 defun om-putAtp

This writea a begin application pair token to dev.

— **defun om-putAtp** —

```
(defun om-putAtp (dev)
 (declare (ignore dev)))
```

———-

42.15.45 defun om-putAttr

This writes a begin attribute token to dev.

— **defun om-putAttr** —

```
(defun om-putAttr (dev)
 (declare (ignore dev)))
```

———-

42.15.46 defun om-putBind

This writes a begin binder token to dev.

— **defun om-putBind** —

```
(defun om-putBind (dev)
 (declare (ignore dev)))
```

———-

42.15.47 defun om-putBVar

This writes a begin bound variable list token to dev.

— **defun om-putBVar** —

```
(defun om-putBVar (dev)
 (declare (ignore dev)))
```

———-

42.15.48 defun om-putByteArray

This writes a byte array to dev.

— **defun om-putByteArray** —

```
(defun om-putByteArray (dev b)
 (declare (ignore dev b)))
```

———-

42.15.49 defun om-putEndApp

This writes an end application token to dev.

— **defun om-putEndApp** —

```
(defun om-putEndApp (dev)
 (declare (ignore dev)))
```

———-

42.15.50 defun om-putEndAtp

This writes an end attribute pair to dev.

— **defun om-putEndAtp** —

```
(defun om-putEndAtp (dev)
 (declare (ignore dev)))
```

———-

42.15.51 defun om-putEndAttr

This writes an end attribute token to dev.

— **defun om-putEndAttr** —

```
(defun om-putEndAttr (dev)
 (declare (ignore dev)))
```

———-

42.15.52 defun om-putEndBind

This writes an end binder token to dev.

— **defun om-putEndBind** —

```
(defun om-putEndBind (dev)
 (declare (ignore dev)))
```

———-

42.15.53 defun om-putEndBVar

This writes and end bound variable list token to dev

— **defun om-putEndBVar** —

```
(defun om-putEndBVar (dev)
 (declare (ignore dev)))
```

———-

42.15.54 defun om-putEndError

This writes an end error token to dev

— **defun om-putEndError** —

```
(defun om-putEndError (dev)
 (declare (ignore dev)))
```

———-

42.15.55 defun om-putEndObject

This writes an end object token to dev.

— **defun om-putEndObject** —

```
(defun om-putEndObject (dev)
 (declare (ignore dev)))
```

———-

42.15.56 defun om-putError

This writes a begin error token to dev.

— **defun om-putError** —

```
(defun om-putError (dev)
 (declare (ignore dev)))
```

———-

42.15.57 defun om-putFloat

This writes the float f to dev.

— **defun om-putFloat** —

```
(defun om-putFloat (dev f)
 (declare (ignore dev f)))
```

———-

42.15.58 defun om-putInt

This writes the integer i to dev

— **defun om-putInt** —

```
(defun om-putInt (dev i)
 (declare (ignore dev i)))
```

———-

42.15.59 defun om-putObject

This writes a begin object token to dev.

— **defun om-putObject** —

```
(defun om-putObject (dev)
 (declare (ignore dev)))
```

———-

42.15.60 defun om-putString

This writes the string s to dev.

— **defun om-putString** —

```
(defun om-putString (dev s)
 (declare (ignore dev s)))
```

———-

42.15.61 defun om-putSymbol

This writes the symbol nm using semantics from cd to dev.

— **defun om-putSymbol** —

```
(defun om-putSymbol (dev cd nm)
 (declare (ignore dev cd nm)))
```

———-

42.15.62 defun om-putVar

This writes the variable v to dev.

— **defun om-putVar** —

```
(defun om-putVar (dev v)
 (declare (ignore dev v)))
```

———-

42.15.63 defun om-stringToStringPtr

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the string from its address? It would appear to be a nop in lisp.

— **defun om-stringToStringPtr** —

```
(defun om-stringToStringPtr (str)
 (declare (ignore str)))
```

———-

42.15.64 defun om-stringPtrToString

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the string address from a string? It would appear to be a nop in lisp.

— **defun om-stringPtrToString** —

```
(defun om-stringPtrToString (str)
 (declare (ignore str)))
```

———-

Chapter 43

NRLIB code.lisp support code

43.0.65 defun makeByteWordVec2

— defun makeByteWordVec2 0 —

```
(defun |makeByteWordVec2| (maxelement initialvalue)
  (let ((n (cond ((null initialvalue) 7) ('t maxelement))))
    (make-array (length initialvalue)
      :element-type (list 'mod (1+ n))
      :initial-contents initialvalue)))
```

———-

43.0.66 defmacro spadConstant

— defmacro spadConstant 0 —

```
(defmacro |spadConstant| (dollar n)
 `(spadcall (svref ,dollar (the fixnum ,n))))
```

———-

Chapter 44

Monitoring execution

```
MONITOR

This file contains a set of function for monitoring the execution
of the functions in a file. It constructs a hash table that contains
the function name as the key and monitor-data structures as the value

The technique is to use a :cond parameter on trace to call the
monitor-incr function to incr the count every time a function is called

*monitor-table*                                 HASH TABLE
   is the monitor table containing the hash entries
*monitor-nrlibs*                                LIST of STRING
   list of nrlib filenames that are monitored
*monitor-domains*                               LIST of STRING
   list of domains to monitor-report (default is all exposed domains)
monitor-data                                    STRUCTURE
   is the defstruct name of records in the table
   name is the first field and is the name of the monitored function
   count contains a count of times the function was called
   monitorp is a flag that skips counting if nil, counts otherwise
   sourcefile is the name of the file that contains the source code

 ***** SETUP, SHUTDOWN ****

monitor-inittable ()                            FUNCTION
   creates the hashtable and sets *monitor-table*
   note that it is called every time this file is loaded
monitor-end ()                                  FUNCTION
   unhooks all of the trace hooks

 ***** TRACE, UNTRACE *****

monitor-add (name &optional sourcefile)         FUNCTION
   sets up the trace and adds the function to the table
monitor-delete (fn)                             FUNCTION
   untraces a function and removes it from the table
monitor-enable (&optional fn)                   FUNCTION
```

```
   starts tracing for all (or optionally one) functions that
   are in the table
monitor-disable (&optional fn)                  FUNCTION
   stops tracing for all (or optionally one) functions that
   are in the table

***** COUNTING, RECORDING  *****

monitor-reset (&optional fn)                    FUNCTION
   reset the table count for the table (or optionally, for a function)
monitor-incr (fn)                               FUNCTION
   increments the count information for a function
   it is called by trace to increment the count
monitor-decr (fn)                               FUNCTION
   decrements the count information for a function
monitor-info (fn)                               FUNCTION
   returns the monitor-data structure for a function

***** FILE IO *****

monitor-write (items file)                      FUNCTION
   writes a list of symbols or structures to a file
monitor-file (file)                             FUNCTION
   will read a file, scan for defuns, monitor each defun
   NOTE: monitor-file assumes that the file has been loaded

***** RESULTS *****

monitor-results ()                              FUNCTION
   returns a list of the monitor-data structures
monitor-untested ()                             FUNCTION
   returns a list of files that have zero counts
monitor-tested (&optional delete)               FUNCTION
   returns a list of files that have nonzero counts
   optionally calling monitor-delete on those functions

***** CHECKPOINT/RESTORE *****
monitor-checkpoint (file)                      FUNCTION
   save the *monitor-table* in a loadable form
monitor-restore (file)                         FUNCTION
  restore a checkpointed file so that everything is monitored

***** ALGEBRA *****
monitor-autoload ()                             FUNCTION
  traces autoload of algebra to monitor corresponding source files
  NOTE: this requires the /spad/int/algebra directory
monitor-dirname (args)                          FUNCTION
  expects a list of 1 libstream (loadvol's arglist) and monitors the source
  this is a function called by monitor-autoload
monitor-nrlib (nrlib)                           FUNCTION
  takes an nrlib name as a string (eg POLY) and returns a list of
  monitor-data structures from that source file
monitor-report ()                               FUNCTION
  generate a report of the monitored activity for domains in
```

```
  *monitor-domains*
monitor-spadfile (name)                        FUNCTION
  given a spad file, report all nrlibs it creates
  this adds each nrlib name to *monitor-domains* but does not
  trace the functions from those domains
monitor-percent ()                             FUNCTION
  ratio of (functions executed)/(functions traced)
monitor-apropos (str)                          FUNCTION
  given a string, find all monitored symbols containing the string
  the search is case-insensitive. returns a list of monitor-data items

for example:
 suppose we have a file "/u/daly/testmon.lisp" that contains:
 (defun foo1 () (print 'foo1))
 (defun foo2 () (print 'foo2))
 (defun foo3 () (foo1) (foo2) (print 'foo3))
 (defun foo4 () (print 'foo4))

 an example session is:

 ; FIRST WE LOAD THE FILE (WHICH INITS *monitor-table*)

 >(load "/u/daly/monitor.lisp")
 Loading /u/daly/monitor.lisp
 Finished loading /u/daly/monitor.lisp
 T

 ; SECOND WE LOAD THE TESTMON FILE
 >(load "/u/daly/testmon.lisp")
 T

 ; THIRD WE MONITOR THE FILE
 >(monitor-file "/u/daly/testmon.lisp")
 monitoring "/u/daly/testmon.lisp"
 NIL

 ; FOURTH WE CALL A FUNCTION FROM THE FILE (BUMP ITS COUNT)
 >(foo1)

 FOO1
 FOO1

 ; AND ANOTHER FUNCTION (BUMP ITS COUNT)
 >(foo2)

 FOO2
 FOO2

 ; AND A THIRD FUNCTION THAT CALLS THE OTHER TWO (BUMP ALL THREE)
 >(foo3)

 FOO1
 FOO2
 FOO3
```

```
FOO3

; CHECK THAT THE RESULTS ARE CORRECT

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 2 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 1 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; STOP COUNTING CALLS TO FOO2

>(monitor-disable 'foo2)
NIL

; INVOKE FOO2 THRU FOO3

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NOTICE THAT FOO1 AND FOO3 WERE BUMPED BUT NOT FOO2
>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; TEMPORARILY STOP ALL MONITORING

>(monitor-disable)
NIL

; CHECK THAT NOTHING CHANGES

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NO COUNT HAS CHANGED
```

```
>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; MONITOR ONLY CALLS TO FOO1

>(monitor-enable 'foo1)
T

; FOO3 CALLS FOO1

>(foo3)

FOO1
FOO2
FOO3
FOO3

; FOO1 HAS CHANGED BUT NOT FOO2 OR FOO3

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 4 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; MONITOR EVERYBODY

>(monitor-enable)
NIL

; CHECK THAT EVERYBODY CHANGES

>(foo3)

FOO1
FOO2
FOO3
FOO3

; EVERYBODY WAS BUMPED

>(monitor-results)
```

```
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; WHAT FUNCTIONS WERE TESTED?

>(monitor-tested)
(FOO1 FOO2 FOO3)

; WHAT FUNCTIONS WERE NOT TESTED?

>(monitor-untested)
(FOO4)

; UNTRACE THE WHOLE WORLD, MONITORING CANNOT RESTART

>(monitor-end)
NIL

; CHECK THE RESULTS

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; CHECK THAT THE FUNCTIONS STILL WORK

>(foo3)

FOO1
FOO2
FOO3
FOO3

; CHECK THAT MONITORING IS NOT OCCURING

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
```

```
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
      "/u/daly/testmon.lisp"))
```

44.0.67 defvar *monitor-domains*

— **initvars** —

```
(defvar *monitor-domains* nil "a list of domains to report")
```

———-

44.0.68 defvar *monitor-nrlibs*

— **initvars** —

```
(defvar *monitor-nrlibs* nil "a list of nrlibs that have been traced")
```

———-

44.0.69 defvar *monitor-table*

— **initvars** —

```
(defvar *monitor-table* nil "a table of all of the monitored data")
```

———-

— **postvars** —

```
(eval-when (eval load)
 (unless *monitor-table* (monitor-inittable)))
```

———-

44.0.70 defstruct monitor-data

— **initvars** —

```
(defstruct monitor-data name count monitorp sourcefile)
```

———-

44.0.71 defstruct libstream

— **initvars** —

```
(defstruct libstream  mode dirname (indextable nil)  (indexstream nil))
```

————-

44.0.72 defun Initialize the monitor statistics hashtable

[*monitor-table* p1235]

— **defun monitor-inittable 0** —

```
(defun monitor-inittable ()
 "initialize the monitor statistics hashtable"
 (declare (special *monitor-table*))
 (setq *monitor-table* (make-hash-table)))
```

————-

44.0.73 defun End the monitoring process, we cannot restart

[*monitor-table* p1235]

— **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*))
```

————-

44.0.74 defun Return a list of the monitor-data structures

[*monitor-table* p1235]

— **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))))
```

———-

44.0.75 defun Add a function to be monitored

[monitor-delete p1237]
[make-monitor-data p??]
[*monitor-table* p1235]

— **defun monitor-add 0** —

```
(defun monitor-add (name &optional sourcefile)
 "add a function to be monitored"
 (declare (special *monitor-table*))
 (unless (fboundp name) (load sourcefile))
 (when (gethash name *monitor-table*)
  (monitor-delete name))
 (eval `(trace (,name :cond (progn (monitor-incr ',name) nil))))
 (setf (gethash name *monitor-table*)
  (make-monitor-data
     :name name :count 0 :monitorp t :sourcefile sourcefile)))
```

———-

44.0.76 defun Remove a function being monitored

[*monitor-table* p1235]

— **defun monitor-delete 0** —

```
(defun monitor-delete (fn)
 "Remove a function being monitored"
 (declare (special *monitor-table*))
 (eval `(untrace ,fn))
 (remhash fn *monitor-table*))
```

———-

44.0.77 defun Enable all (or optionally one) function for monitoring

[*monitor-table* p1235]

— **defun monitor-enable 0** —

```
(defun monitor-enable (&optional fn)
 "enable all (or optionally one) function for monitoring"
```

```
 (declare (special *monitor-table*))
 (if fn
  (progn
   (eval `(trace (,fn :cond (progn (monitor-incr ',fn) nil))))
   (setf (monitor-data-monitorp (gethash fn *monitor-table*)) t))
  (maphash
   #'(lambda (key value)
      (declare (ignore value))
      (eval `(trace (,key :cond (progn (monitor-incr ',key) nil))))
      (setf (monitor-data-monitorp (gethash key *monitor-table*)) t))
   *monitor-table*)))
```

———-

44.0.78 defun Disable all (optionally one) function for monitoring

[*monitor-table* p1235]

— **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*)))
```

———-

44.0.79 defun Reset the table count for the table (or a function)

[*monitor-table* p1235]

— **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*)))
```

———-

44.0.80 defun Incr the count of fn by 1

[*monitor-table* p1235]

— **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))))
```

———-

44.0.81 defun Decr the count of fn by 1

[*monitor-table* p1235]

— **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))))
```

———-

44.0.82 defun Return the monitor information for a function

[*monitor-table* p1235]

— **defun monitor-info 0** —

```
(defun monitor-info (fn)
 "return the monitor information for a function"
 (declare (special *monitor-table*))
 (gethash fn *monitor-table*))
```

———-

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

[done p??]
[done p??]
[monitor-add p1237]

— **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))))))))
```

———-

44.0.84 defun Return a list of the functions with zero count fields

[*monitor-table* p1235]

— **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 )))
```

———-

44.0.85 defun Return a list of functions with non-zero counts

[monitor-delete p1237]
[*monitor-table* p1235]

— **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)))
```

———-

44.0.86 defun Write out a list of symbols or structures to a file

— **defun monitor-write 0** —

```
(defun monitor-write (items file)
 "write out a list of symbols or structures to a file"
 (with-open-file (out file :direction :output)
  (dolist (item items)
    (if (symbolp item)
     (format out "~s~%" item)
     (format out "~s~50t~s~100t~s~%"
       (monitor-data-sourcefile item)
       (monitor-data-name item)
       (monitor-data-count item))))))
```

———-

44.0.87 defun Save the *monitor-table* in loadable form

[*monitor-table* p1235]
[*print-package* p??]

— **defun monitor-checkpoint 0** —

```
(defun monitor-checkpoint (file)
 "save the *monitor-table* in loadable form"
 (let ((*print-package* t))
  (declare (special *print-package* *monitor-table*))
  (with-open-file (out file :direction :output)
   (format out "~&~%(IN-PACKAGE \"BOOT\")~%~%")
   (format out "(monitor-inittable)~%")
   (dolist (data (monitor-results))
    (format out "(monitor-add '~s ~s)~%"
     (monitor-data-name data)
     (monitor-data-sourcefile data))
    (format out "(setf (gethash '~s *monitor-table*)
```

```
                (make-monitor-data :name '~s :count ~s :monitorp ~s
                                   :sourcefile ~s))~%"
    (monitor-data-name data)
    (monitor-data-name data)
    (monitor-data-count data)
    (monitor-data-monitorp data)
    (monitor-data-sourcefile data))))))
```

———-

44.0.88 defun restore a checkpointed file

— **defun monitor-restore 0** —

```
(defun monitor-restore (file)
 "restore a checkpointed file"
 (load file))
```

———-

44.0.89 defun Printing help documentation

— **defun monitor-help 0** —

```
(defun monitor-help ()
 (format t "~%
;;; MONITOR
;;;
;;; This file contains a set of function for monitoring the execution
;;; of the functions in a file. It constructs a hash table that contains
;;; the function name as the key and monitor-data structures as the value
;;;
;;; The technique is to use a :cond parameter on trace to call the
;;; monitor-incr function to incr the count every time a function is called
;;;
;;; *monitor-table*                                HASH TABLE
;;;    is the monitor table containing the hash entries
;;; *monitor-nrlibs*                               LIST of STRING
;;;    list of nrlib filenames that are monitored
;;; *monitor-domains*                              LIST of STRING
;;;    list of domains to monitor-report (default is all exposed domains)
;;; monitor-data                                   STRUCTURE
;;;    is the defstruct name of records in the table
;;;    name is the first field and is the name of the monitored function
;;;    count contains a count of times the function was called
;;;    monitorp is a flag that skips counting if nil, counts otherwise
;;;    sourcefile is the name of the file that contains the source code
;;;
;;;  ***** SETUP, SHUTDOWN ****
;;;
```

```
;;; monitor-inittable ()                          FUNCTION
;;;    creates the hashtable and sets *monitor-table*
;;;    note that it is called every time this file is loaded
;;; monitor-end ()                                FUNCTION
;;;    unhooks all of the trace hooks
;;;
;;;  ***** TRACE, UNTRACE *****
;;;
;;; monitor-add (name &optional sourcefile)       FUNCTION
;;;    sets up the trace and adds the function to the table
;;; monitor-delete (fn)                           FUNCTION
;;;    untraces a function and removes it from the table
;;; monitor-enable (&optional fn)                 FUNCTION
;;;    starts tracing for all (or optionally one) functions that
;;;    are in the table
;;; monitor-disable (&optional fn)                FUNCTION
;;;    stops tracing for all (or optionally one) functions that
;;;    are in the table
;;;
;;; ***** COUNTING, RECORDING  *****
;;;
;;; monitor-reset (&optional fn)                  FUNCTION
;;;    reset the table count for the table (or optionally, for a function)
;;; monitor-incr (fn)                             FUNCTION
;;;    increments the count information for a function
;;;    it is called by trace to increment the count
;;; monitor-decr (fn)                             FUNCTION
;;;    decrements the count information for a function
;;; monitor-info (fn)                             FUNCTION
;;;    returns the monitor-data structure for a function
;;;
;;; ***** FILE IO *****
;;;
;;; monitor-write (items file)                    FUNCTION
;;;    writes a list of symbols or structures to a file
;;; monitor-file (file)                           FUNCTION
;;;    will read a file, scan for defuns, monitor each defun
;;;    NOTE: monitor-file assumes that the file has been loaded
;;;
;;; ***** RESULTS *****
;;;
;;; monitor-results ()                            FUNCTION
;;;    returns a list of the monitor-data structures
;;; monitor-untested ()                           FUNCTION
;;;    returns a list of files that have zero counts
;;; monitor-tested (&optional delete)             FUNCTION
;;;    returns a list of files that have nonzero counts
;;;    optionally calling monitor-delete on those functions
;;;
;;; ***** CHECKPOINT/RESTORE *****
;;;
;;; monitor-checkpoint (file)                    FUNCTION
;;;    save the *monitor-table* in a loadable form
;;; monitor-restore (file)                       FUNCTION
```

```
;;;   restore a checkpointed file so that everything is monitored
;;;
;;; ***** ALGEBRA *****
;;;
;;; monitor-autoload ()                          FUNCTION
;;;   traces autoload of algebra to monitor corresponding source files
;;;   NOTE: this requires the /spad/int/algebra directory
;;; monitor-dirname (args)                       FUNCTION
;;;   expects a list of 1 libstream (loadvol's arglist) and monitors the source
;;;   this is a function called by monitor-autoload
;;; monitor-nrlib (nrlib)                        FUNCTION
;;;   takes an nrlib name as a string (eg POLY) and returns a list of
;;;   monitor-data structures from that source file
;;; monitor-report ()                            FUNCTION
;;;   generate a report of the monitored activity for domains in
;;;   *monitor-domains*
;;; monitor-spadfile (name)                      FUNCTION
;;;   given a spad file, report all nrlibs it creates
;;;   this adds each nrlib name to *monitor-domains* but does not
;;;   trace the functions from those domains
;;; monitor-percent ()                           FUNCTION
;;;   ratio of (functions executed)/(functions traced)
;;; monitor-apropos (str)                        FUNCTION
;;;   given a string, find all monitored symbols containing the string
;;;   the search is case-insensitive. returns a list of monitor-data items
") nil)
```

———-

44.0.90 Monitoring algebra files

44.0.91 defun Monitoring algebra code.lsp files

[*monitor-nrlibs* p1235]

— **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))))
```

———-

44.0.92 defun Monitor autoloaded files

— defun monitor-autoload 0 —

```
(defun monitor-autoload ()
 "traces autoload of algebra to monitor corresponding source files"
 (trace (vmlisp::loadvol
          :entrycond nil
          :exitcond (progn (monitor-dirname system::arglist) nil))))
```

———-

44.0.93 defun Monitor an nrlib

[*monitor-table* p1235]

— 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))
```

———-

44.0.94 defun Given a monitor-data item, extract the nrlib name

— defun monitor-libname 0 —

```
(defun monitor-libname (item)
  "given a monitor-data item, extract the nrlib name"
  (pathname-name (car (last
   (pathname-directory (monitor-data-sourcefile item))))))
```

———-

44.0.95 defun Is this an exposed algebra function?

— **defun monitor-exposedp 0** —

```
(defun monitor-exposedp (fn)
 "exposed functions have more than 1 semicolon. given a symbol, count them"
   (> (count #\; (symbol-name fn)) 1))
```

———-

44.0.96 defun Monitor exposed domains

TPDHERE: note that the file interp.exposed no longer exists. The exposure information is now in this book. This needs to work off the internal exposure list, not the file.

[done p**??**]
[done p**??**]
[*monitor-domains* p1235]

— **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*)))))))))
```

———-

44.0.97 defun Generate a report of the monitored domains

[monitor-readinterp p1246]
[*monitor-domains* p1235]

— defun monitor-report 0 —

```
(defun monitor-report ()
 "generate a report of the monitored activity for domains in *monitor-domains*"
 (let (nrlibs nonzero total)
 (declare (special *monitor-domains*))
  (unless *monitor-domains* (monitor-readinterp))
  (setq nonzero 0)
  (setq total 0)
  (maphash
   #'(lambda (k v)
      (declare (ignore k))
      (let (nextlib point)
       (when (> (monitor-data-count v) 0) (incf nonzero))
       (incf total)
       (setq nextlib (monitor-libname v))
       (setq point (member nextlib nrlibs :test #'string= :key #'car))
       (if point
         (setf (cdr (first point)) (cons v (cdr (first point))))
         (push (cons nextlib (list v)) nrlibs))))
   *monitor-table*)
  (format t "~d of ~d (~d percent) tested~%" nonzero total
    (round (/ (* 100.0 nonzero) total)))
  (setq nrlibs (sort nrlibs #'string< :key #'car))
  (dolist (pair nrlibs)
   (let ((exposedcount 0) (testcount 0))
    (when (member (car pair) *monitor-domains* :test #'string=)
     (format t "for library ~s~%" (car pair))
     (dolist (item (sort (cdr pair)  #'> :key #'monitor-data-count))
      (when (monitor-exposedp (monitor-data-name item))
       (incf exposedcount)
       (when (> (monitor-data-count item) 0) (incf testcount))
       (format t "~5d ~s~%"
         (monitor-data-count item)
         (monitor-data-name item))))
     (if (= exposedcount testcount)
      (format t "~a has all exposed functions tested~%" (car pair))
      (format t "Daly bug:~a has untested exposed functions~%" (car pair))))))
 nil))
```

———-

44.0.98 defun Parse an)abbrev expression for the domain name

— defun monitor-parse 0 —

```
(defun monitor-parse (expr)
```

```
  (let (point1 point2)
   (setq point1 (position #\space expr :test #'char=))
   (setq point1 (position #\space expr :start point1 :test-not #'char=))
   (setq point1 (position #\space expr :start point1 :test #'char=))
   (setq point1 (position #\space expr :start point1 :test-not #'char=))
   (setq point2 (position #\space expr :start point1 :test #'char=))
   (subseq expr point1 point2)))
```

———-

44.0.99 defun Given a spad file, report all nrlibs it creates

[done p??]
[done p??]
[monitor-parse p1247]
[*monitor-domains* p1235]

— **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=))))))))
```

———-

44.0.100 defun Print percent of functions tested

[*monitor-table* p1235]

— **defun monitor-percent 0** —

```
(defun monitor-percent ()
 "Print percent of functions tested"
 (let (nonzero total)
 (declare (special *monitor-table*))
  (setq nonzero 0)
  (setq total 0)
  (maphash
   #'(lambda (k v)
      (declare (ignore k))
      (when (> (monitor-data-count v) 0) (incf nonzero))
      (incf total))
   *monitor-table*)
```

```
  (format t "~d of ~d (~d percent) tested~%" nonzero total
    (round (/ (* 100.0 nonzero) total)))))
```

———-

44.0.101 defun Find all monitored symbols containing the string

[*monitor-table* p1235]

— **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 45

HyperDoc

Hyperdoc works by building up a page "description" which consists of a list of items. Each item is itself a list whose first element is a tag. The htMakePage1 routine walks the list of items, dispatching on the tag (called itemType), to create the final page.

Pages are constructed with a latex-like syntax. The valid syntax values are in the primitive-HtCommands list.

Each page is an 8 part list, of which the description is the last item. See HTPage Layout for more details.

Pages can also be constructed by code. For example, the page found by

```
Basic Commands -> Matrix
```

is constructed by a call to bcReadMatrix. This routine sets the page title htInitPage, sets up the EXIT button handler htpSetProperty, constructs a page descriptions which is passes to htMakePage, and then calls htShowPage to display the result.

45.1 Hyperdoc macro handling and util.ht

All of the macros used in hyperdoc are in this hash table. User-defined macros are read from the file **doc/util.ht**

45.1.1 defvar $htMacroTable

— **initvars** —

```
(defvar |$htMacroTable| (make-hash-table :test #'equal))
```

————-

These are the primitive hyperdoc commands. They are directly implemented. The **buildHtMacroTable** function adds these to the `$htMacroTable` at startup.

45.1.2 defvar $primitiveHtCommands

— initvars —

```
(defvar |$primitiveHtCommands|
  '(("\\ContinueButton" . 1)
    ("\\andexample" . 1)
    ("\\autobutt" . 0)
    ("\\autobuttons" . 0)
    ("\\begin" . 1)
    ("\\beginscroll" . 0)
    ("\\bound" . 1)
    ("\\fbox" . 1)
    ("\\centerline" . 1)
    ("\\downlink" . 2)
    ("\\em" . 0)
    ("\\end" . 1)
    ("\\endscroll" . 0)
    ("\\example" . 1)
    ("\\free" . 1)
    ("\\graphpaste" . 1)
    ("\\helppage" . 1)
    ("\\htbmdir" . 0)
    ("\\htbmfile" . 1)
    ("\\indent" . 1)
    ("\\inputbitmap" . 1)
    ("\\inputstring" . 3)
    ("\\item" . 0)
    ("\\keyword" . 1)
    ("\\link" . 2)
    ("\\lispdownlink" . 2)
    ("\\lispmemolink" . 2)
    ("\\lispwindowlink" . 2)
    ("\\menudownlink" . 2)
    ("\\menuitemstyle" . 1)
    ("\\menulink" . 2)
    ("\\menulispdownlink" . 2)
    ("\\menulispmemolink" . 2)
    ("\\menulispwindowlink" . 2)
    ("\\menumemolink" . 2)
    ("\\menuwindowlink" . 2)
    ("\\newline" . 0)
    ("\\radioboxes" . 3)
    ("\\space" . 1)
    ("\\spadcommand" . 1)
    ("\\stringvalue" . 1)
    ("\\tab" . 1)
    ("\\table" . 1)
    ("\\vspace" . 1)
    ("\\windowlink" . 2)))
```

————-

45.1.3 defvar $newPage

— **initvars** —

```
(defvar |$newPage| nil "The new page being built")
```

————-

45.1.4 defun Build the table of hyperdoc macros

Hash user-defined macros from **doc/util.ht** into **htMacroTable**. Hash primitive hyperdoc macros into **htMacroTable**. [buildHtMacroTable util.ht (vol7.1)]
[getHtMacroItem p1253]
[sayBrightly p**??**]
[concat p1107]
[getenviron p291]
[hput p1105]
[$htMacroTable p1251]
[$primitiveHtCommands p1252]

— **defun buildHtMacroTable** —

```
(defun |buildHtMacroTable| ()
 (let (fn)
  (declare (special |$htMacroTable| |$primitiveHtCommands|))
  (setq fn (concat (getenviron "AXIOM") "/doc/util.ht"))
  (cond
   ((probe-file fn)
    (with-open-file (instream fn)
     (loop
       for line = (read-line instream nil :eof)
       until (eq line :eof)
       do
        (when
         (multiple-value-bind (command numOfArgs) (|getHtMacroItem| line)
          (hput |$htMacroTable| command numOfArgs))))
    (dolist (pair |$primitiveHtCommands|)
     (hput |$htMacroTable| (car pair) (cdr pair)))))
   (t (|sayBrightly| "Warning: macro table not found")))
  |$htMacroTable|))
```

————-

45.1.5 defun Get new command name and number of args

This processes **newcommand** lines read from **doc/util.ht** An example newcommand looks like

```
\newcommand{\menulink}[2]        {\menudownlink{#1}{#2}}
```

This function returns a pair whose CAR is the new command name and whose CDR is the number of arguments. If there are zero arguments the brackets and number will not appear.

However brackets can appear in the new command so we need to fix the original code to handle this new case. We set up a wall starting after the first closing brace.

```
\newcommand{\beginmenu}          {\beginitems[\MenuDotBitmap]}
```

[getHtMacroItem util.ht (vol7.1)]

getHtMacroItem : **String** → **Values (String NonNegativeInteger)**

— **defun getHtMacroItem** —

```
(defun |getHtMacroItem| (line)
 (let (k command m i j wall digitString)
  (when (|stringPrefix?| "\\newcommand{" line)
    (setq k (position #\} line :start 11))
    (setq wall (position #\{ line :start k)) ; wall off the body of command
    (setq command (substring line 12 (- k 12)))
    (setq m (length line))
    (setq i (position #\[ line :start k))
    (if (and i (< i m) wall (< i wall))
     (progn                 ; brackets. parse number of args
      (setq j (position #\] line :start (+ i 1)))
      (setq digitString (substring line (+ i 1) (- (- j i) 1)))
      (when (every #'digitp digitString)
       (values command (parse-integer digitString))))
     (values command 0)))))  ; no brackets
```

———-

We populate the htMacroTable at load time.

— **postvars** —

```
(eval-when (eval load)
 (|buildHtMacroTable|))
```

———-

45.1.6 defun Is the first string a prefix of the second?

— **defun stringPrefix? 0** —

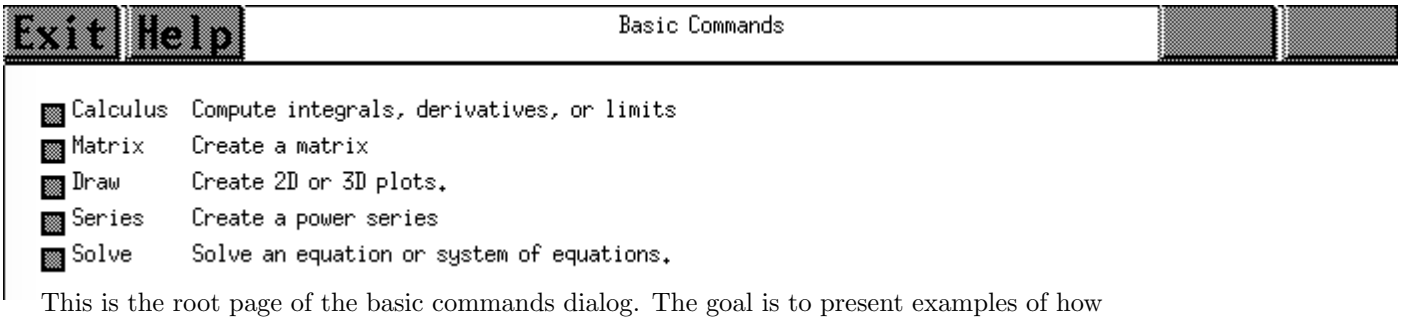
```
(defun |stringPrefix?| (pref str)
 (let (lp)
  (cond
   ((null (and (stringp pref) (stringp str))) nil)
   ((eql (setq lp (length pref)) 0) t)
   ((> lp (length str)) nil)
   (t (every #'char= pref str)))))
```

———-

Chapter 46

HyperDoc Basic Command support

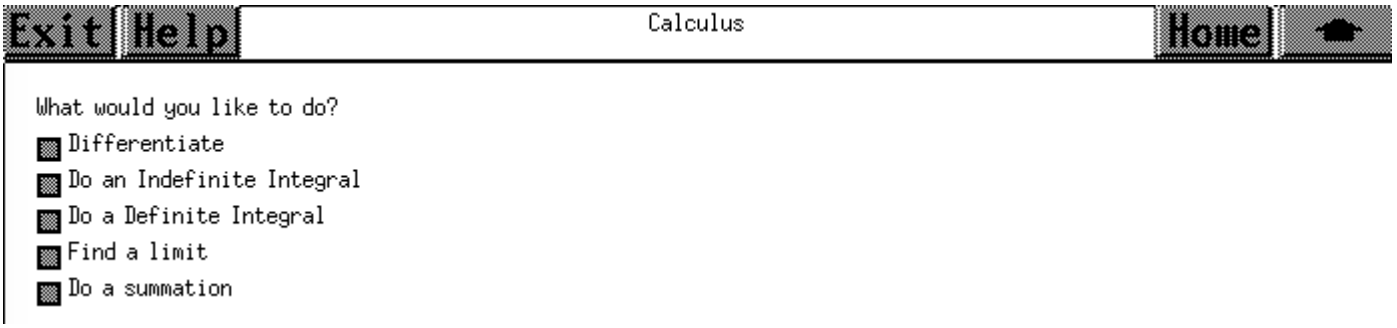
Basic Command



This is the root page of the basic commands dialog. The goal is to present examples of how to construct command lines which demonstrate using Axiom to solve problems.

46.1 Calculus

Calculus



46.1.1 defun Calculus - Differentiate

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]
[$EmptyMode p629]

Calculus → Differentiate

Enter the function you want to differentiate:

sin(x*y)_

List the variables you want to differentiate with respect to?

x y

List the number of times you want to differentiate with respect to each variable (leave blank if once for each)

1 2

Continue

Pressing the **Continue** button will call the function **bcDifferentiateGen** due to this line:

```
(|htMakeDoneButton| "Continue" '|bcDifferentiateGen|)
```

— **defun bcDifferentiate** —

```
(defun |bcDifferentiate| ()
 (declare (special |$EmptyMode|))
 (|htInitPage| '|Differentiate Basic Command| nil)
 (|htMakePage|
  '(((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em function} you want to differentiate:")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "sin(x*y)" |diffand| EM))
    (|text| . "\\blankline") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| .
"\\newline List the {\\em variables} you want to differentiate with respect to?")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "x y" |variables| S . |quoteString|))
    (|text| . "\\blankline") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| .
"\\newline List the number of {\\em times} you want to differentiate with respect to each variable (leave blank
```

```
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "1 2" |times| S . |quoteString|))))
 (|htMakeDoneButton| "Continue" '|bcDifferentiateGen|)
 (|htShowPage|))
```

———-

Pressing the **Continue** calls `bcDifferentiateGen`

46.1.2 defun bcDifferentiateGen

[htpLabelInputString p1342]
[bcString2WordList p1335]
[bcwords2liststring p1335]
[length p**??**]
[bcError p1336]
[concat p1107]
[bcGen p1334]

— **defun bcDifferentiateGen** —

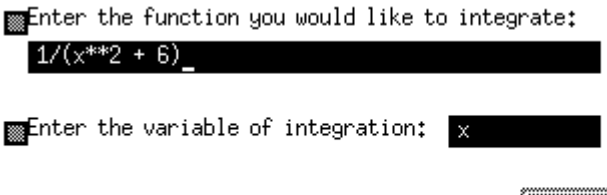
```
(defun |bcDifferentiateGen| (htPage)
 (let (mand varlist indexList varpart indexpart lastPart)
  (setq mand (|htpLabelInputString| htPage '|diffand|))
  (setq varlist
    (|bcString2WordList| (|htpLabelInputString| htPage '|variables|)))
  (setq indexList
    (|bcString2WordList| (|htpLabelInputString| htPage '|times|)))
  (setq varpart
    (if (> (|#| varlist) 1)
      (|bcwords2liststring| varlist)
      (car varlist)))
  (setq indexpart
   (cond
     ((null indexList) nil)
     ((null (cdr indexList)) (car indexList))
     ((= (|#| indexList) (|#| 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
  (setq lastPart (if indexpart (concat "," indexpart ")") ")"))
  (|bcGen| (concat "differentiate(" mand "," varpart lastPart))))
```

———-

46.1.3 defun Calculus - Do an Indefinite Integral

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Calculus → Indefinite Integral



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}")
      (|text| . "Enter the {\\em variable of integration}:")
      (|text| . "\\tab{37}") (|bcStrings| (10 |x| |symbol| SY))
      (|doneButton| "Continue" |bcIndefiniteIntegrateGen|)))
  (|htShowPage|))
```

———-

46.1.4 defun bcIndefiniteIntegrateGen

[htpLabelInputString p1342]
[concat p1107]
[bcGen p1334]

— **defun bcIndefiniteIntegrateGen** —

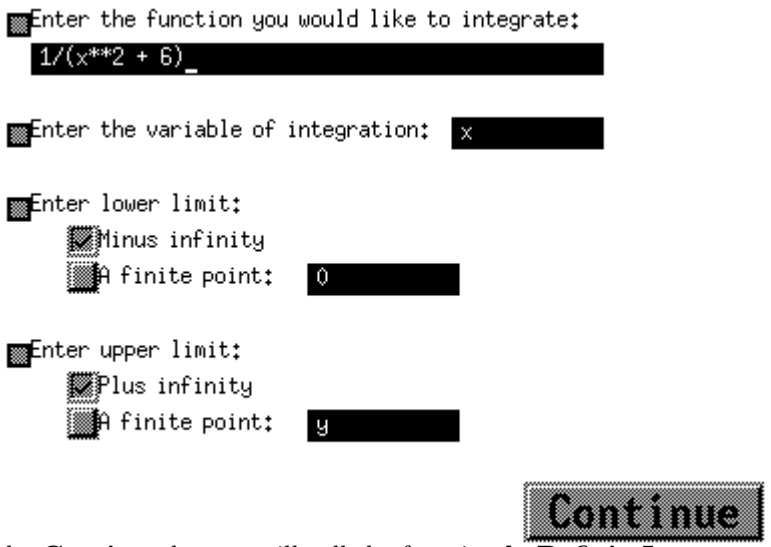
```
(defun |bcIndefiniteIntegrateGen| (htPage)
 (let (integrand var)
  (setq integrand (|htpLabelInputString| htPage '|integrand|))
  (setq var (|htpLabelInputString| htPage '|symbol|))
  (|bcGen| (concat "integrate(" integrand "," var ")"))))
```

———-

46.1.5 defun Calculus - Do a Definite Integral

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Calculus → Definite Integral



Pressing the **Continue** button will call the function **bcDefiniteIntegrateGen** due to this line:

```
(|doneButton| "Continue" |bcDefiniteIntegrateGen|)
```

— defun bcDefiniteIntegrate —

```
(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}")
    (|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}")
    (|text| . "Enter the {\\em variable of integration}:")
    (|text| . "\\tab{37}") (|bcStrings| (10 |x| |symbol| SY))
```

```
    (|text| . "\\blankline") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "\\newline Enter {\\em lower limit}:")
    (|radioButtons| |fromButton|
      ("" "Minus infinity" |minusInfinity|)
      (""
         ((|text| . "A finite point:\\tab{15}")
          (|bcStrings| (10 0 |from| EM . |bcOptional|)))
         |fromPoint|))
    (|text| . "\\blankline") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "\\indent{2}\\newline Enter {\\em upper limit}:")
    (|radioButtons| |toButton|
        ("" "Plus infinity" |plusInfinity|)
        (""
         ((|text| "A finite point:\\tab{15}")
          (|bcStrings| (10 |y| |to| EM . |bcOptional|)))
         |toPoint|))
     (|doneButton| "Continue" |bcDefiniteIntegrateGen|)))
 (|htShowPage|))
```

———-

46.1.6 defun bcDefiniteIntegrateGen

[htpLabelInputString p1342]
[htpButtonValue p1340]
[concat p1107]
[bcGen p1334]

— **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"))
  (setq upperLimit
   (if (eq (|htpButtonValue| htPage '|toButton|) '|toPoint|)
    (|htpLabelInputString| htPage '|to|)
    "%plusInfinity"))
  (setq varpart (concat var " = " lowerLimit ".." upperLimit))
  (|bcGen| (concat "integrate(" integrand "," varpart ")"))))
```

———-

46.1.7 defun Calculus - Find a limit

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Calculus → Limit

What kind of limit do you want to compute?

A real limit? The limit as the variable approaches a real value along the real axis

A complex limit? The limit as the variable approaches a complex value along any path in the complex plane

— defun bcLimit —

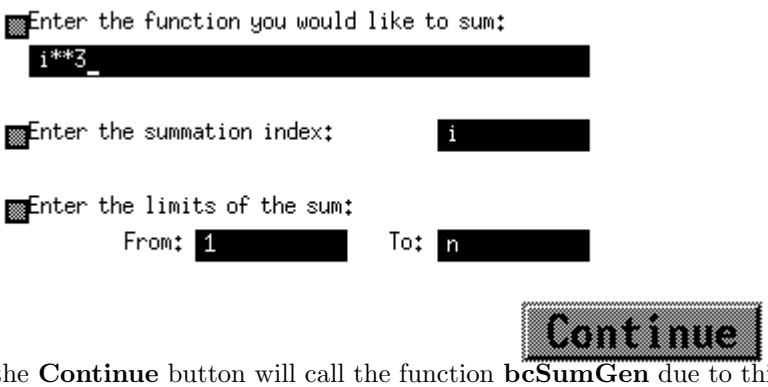
```
(defun |bcLimit| ()
 (declare (special |$EmptyMode|))
 (|htInitPage| "Limit Basic Command" NIL)
 (|htMakePage|
  '((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
    (|text| . "What kind of limit do you want to compute? ")
    (|text| . "\\blankline ")
    (|text| . "\\beginmenu")
    (|text| . "\\item ")
    (|bcLinks| ("\\menuitemstyle{A real limit?}" "" |bcRealLimit| |real|))
    (|text| . "\\indentrel{17}\\tab{0}")
    (|text| .
"The limit as the variable approaches a {\\em real} value along the real axis")
    (|text| . "\\indentrel{-17}") (|text| . "\\item ")
    (|text| . "\\blankline ")
    (|bcLinks|
      ("\\menuitemstyle{A complex limit?}" "" |bcComplexLimit| |complex|))
    (|text| . "\\indentrel{17}\\tab{0}")
    (|text| .
"The limit as the variable approaches a {\\em complex} value along any path in the complex plane")
    (|text| . "\\indentrel{-17}")
    (|text| . "\\endmenu")))
 (|htShowPage|))
```

———-

46.1.8 defun Calculus - Do a summation

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Calculus → Summation



Pressing the **Continue** button will call the function **bcSumGen** due to this line:

```
(|doneButton| "Continue" |bcSumGen|)
```

— **defun bcSum** —

```
(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| . "\\blankline ") (|text| . "\\newline ")
     (|text| . "\\menuitemstyle{}\\tab{2}")
     (|text| . "Enter the {\\em summation index}:")
     (|text| . "\\tab{36}") (|bcStrings| (10 |i| |index| SY))
     (|text| . "\\blankline ") (|text| . "\\newline ")
     (|text| . "\\menuitemstyle{}\\tab{2}")
     (|text| . "Enter the limits of the sum:")
     (|text| . "\\newline\\tab{10}{\\em From:}")
     (|bcStrings| (10 1 |first| S))
     (|text| . "\\tab{32}{\\em To:}") (|text| . "\\tab{36}")
     (|bcStrings| (10 |n| |last| S))
     (|doneButton| "Continue" |bcSumGen|)))
 (|htShowPage|))
```

———-

46.1.9 defun bcSumGen

[htpLabelInputString p1342]
[concat p1107]
[bcGen p1334]

— defun bcSumGen —

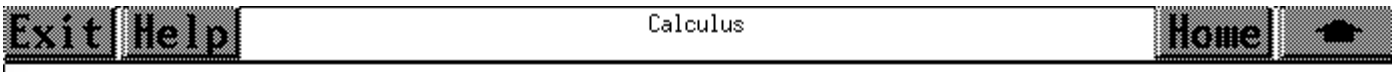
```
(defun |bcSumGen| (htPage)
 (let (mand index car last)
  (setq mand  (|htpLabelInputString| htPage '|summand|))
  (setq index (|htpLabelInputString| htPage '|index|))
  (setq car   (|htpLabelInputString| htPage '|first|))
  (setq last  (|htpLabelInputString| htPage '|last|))
  (|bcGen| (concat "sum(" mand "," index " = " car ".." last ")"))))
```

———-

46.2 Matrix

46.2.1 defun Basic Commands - Matrix

Matrix



What would you like to do?

- Differentiate
- Do an Indefinite Integral
- Do a Definite Integral
- Find a limit
- Do a summation

[bcReadMatrix p1318]

— defun bcMatrix —

```
(defun |bcMatrix| () (|bcReadMatrix| nil))
```

———-

46.3 Draw

46.3.1 defun Basic Commands - Draw

[htInitPage p1349]
[bcHt p1347]
[htShowPage p1350]

Draw Basic Command

What would you like to draw?

Two Dimensional Plots

A function of one variable y = f(x)
A parametrically defined curve (x(t), y(t))
A solution to a polynomial equation p(x,y) = 0

Three Dimensional Surfaces

A function of two variables z = f(x,y)
A parametrically defined tube (x(t), y(t), z(t))
A parameterically defined surface (x(u,v), y(u,v), z(u,v))

— **defun bcDraw** —

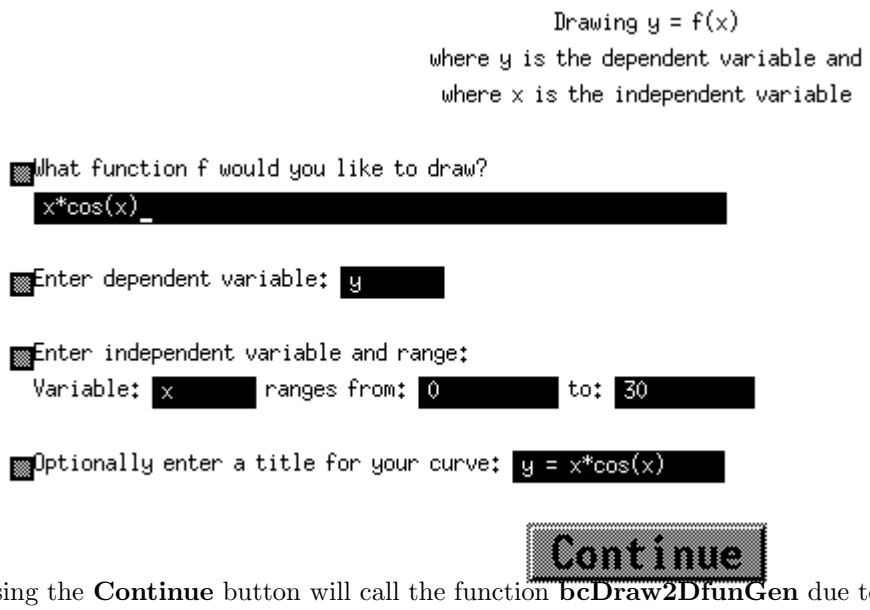
```
(defun |bcDraw| ()
 (|htInitPage| "Draw Basic Command" NIL)
 (|bcHt| "What would you like to draw?")
 (|bcHt| "\\newline\\centerline{{\\em Two Dimensional Plots}}\\newline")
 (|bcHt| "\\lispdownlink{A function of one variable}{(|bcDraw2Dfun|)}")
 (|bcHt| "\\space{2}y = f(x)\\newline")
 (|bcHt| "\\lispdownlink{A parametrically defined curve}{(|bcDraw2Dpar|)}")
 (|bcHt| "\\space{2}(x(t), y(t))\\newline")
 (|bcHt|
  "\\lispdownlink{A solution to a polynomial equation}{(|bcDraw2DSolve|)}")
 (|bcHt| "\\space{2} p(x,y) = 0\\newline")
 (|bcHt| "\\vspace{1}\\newline ")
 (|bcHt| "\\centerline{{\\em Three Dimensional Surfaces}}\\newline\\newline")
 (|bcHt| "\\lispdownlink{A function of two variables}{(|bcDraw3Dfun|)}")
 (|bcHt| "\\space{2} z = f(x,y)\\newline")
 (|bcHt| "\\lispdownlink{A parametrically defined tube}{(|bcDraw3Dpar|)}")
 (|bcHt| "\\space{2}(x(t), y(t), z(t))\\newline")
 (|bcHt| "\\lispdownlink{A parameterically defined surface}{(|bcDraw3Dpar1|)}")
 (|bcHt| "\\space{2}(x(u,v), y(u,v), z(u,v))\\newline")
 (|htShowPage|))
```

———-

46.3.2 defun Draw - Function of one variable

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Draw Basic Command by Function



Pressing the **Continue** button will call the function **bcDraw2DfunGen** due to this line:

```
(|doneButton| "Continue" |bcDraw2DfunGen|)
```

— defun bcDraw2Dfun —

```
(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}\\newline "
"\\centerline{where {\\em x} is the independent variable}\\vspace{1}\\newline "
"\\menuitemstyle{}\\tab{2}What {\\em function} f would you like to draw?\\newline\\tab{2}")
    (|bcStrings| (55 "x*cos(x)" |function| EM))
    (|text| .
"\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}Enter {\\em dependent} variable:")
    (|bcStrings| (6 |y| |dependent| SY))
    (|text| . "\\newline\\vspace{1}\\newline ")
    (|text| .
"\\menuitemstyle{}\\tab{2}Enter {\\em independent} variable and {\\em range}:\\newline\\tab{2} ")
    (|text| . "{\\em Variable:}") (|bcStrings| (6 |x| |ind| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 0 |from1| F)) (|text| . "{\\em to:}")
    (|bcStrings| (9 30 |to1| F))
    (|text| "\\indent{0}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
           "Optionally enter a {\\em title} for your curve:")
    (|bcStrings| (15 "y = x*cos(x)" |title| S))
    (|text| . "\\indent{0}")
    (|doneButton| "Continue" |bcDraw2DfunGen|) (|text| . "{}")))
 (|htShowPage|))
```

———-

46.3.3 defun bcDraw2DfunGen

[htpLabelInputString p1342]
[concat p1107]
[bcFinish p1333]
[bcDrawIt2 p1316]

— **defun bcDraw2DfunGen** —

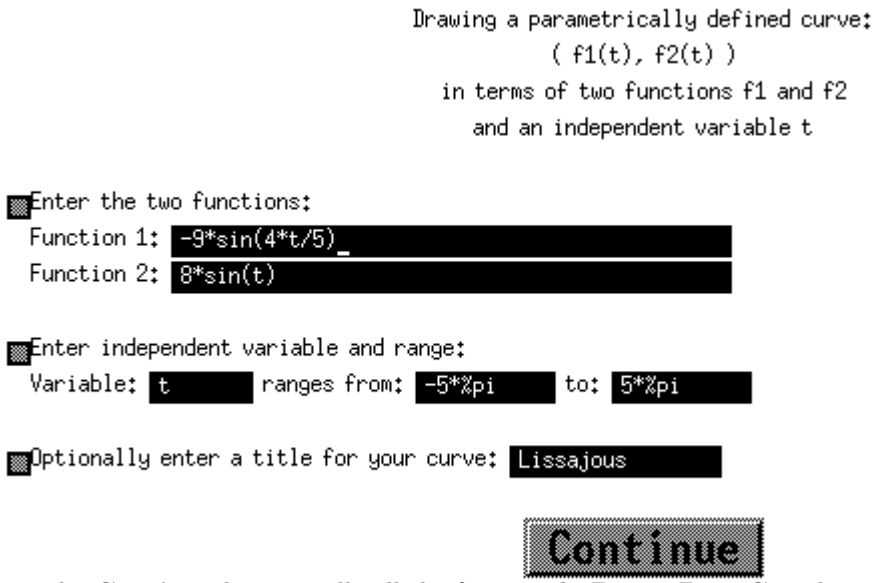
```
(defun |bcDraw2DfunGen| (htPage)
 (let (fun dep ind from1 to1 title titlePart)
  (setq fun   (|htpLabelInputString| htPage '|function|))
  (setq dep   (|htpLabelInputString| htPage '|dependent|))
  (setq ind   (|htpLabelInputString| htPage '|ind|))
  (setq from1 (|htpLabelInputString| htPage '|from1|))
  (setq to1   (|htpLabelInputString| htPage '|to1|))
  (setq title (|htpLabelInputString| htPage '|title|))
  (cond
   ((not (string-equal title ""))
     (setq titlePart (concat "{}" "title ==\"" title "\""))
     (|bcFinish| "draw" fun (|bcDrawIt2| ind from1 to1) titlePart))
   (t
     (|bcFinish| "draw" fun (|bcDrawIt2| ind from1 to1))))))
```

———-

46.3.4 defun Draw - Parametrically defined curve

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Draw Basic Command by Parameters



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|)
        (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
    (|text| "\\centerline{Drawing a parametrically defined curve:}\\newline "
            "\\centerline{{\\em ( f1(t), f2(t) )}}\\newline "
            "\\centerline{in terms of two functions {\\em f1} and {\\em f2}}"
       "\\centerline{and an independent variable {\\em t}}\\vspace{1}\\newline"
            "\\menuitemstyle{}\\tab{2}Enter the two {\\em functions:}")
    (|text| . "\\newline\\tab{2}{\\em Function 1:}")
    (|bcStrings| (44 "-9*sin(4*t/5)" |function1| EM))
    (|text| . "\\newline\\tab{2}{\\em Function 2:}")
    (|bcStrings| (44 "8*sin(t)" |function2| EM))
    (|text| .
"\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}Enter {\\em independent} variable and range:\\newline\\tab{2} ")
    (|text| . "{\\em Variable:}") (|bcStrings| (6 |t| |ind| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 "-5*\\%pi" |from1| F))
    (|text| . "{\\em to:}") (|bcStrings| (9 "5*\\%pi" |to1| F))
    (|text| "\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}"
            "Optionally enter a {\\em title} for your curve:")
    (|bcStrings| (15 "Lissajous" |title| S))
    (|text| . "\\indent{0}")
    (|doneButton| "Continue" |bcDraw2DparGen|)))
 (|htShowPage|))
```

———-

46.3.5 defun bcDraw2DparGen

[htpLabelInputString p1342]
[concat p1107]
[bcFinish p1333]
[bcDrawIt2 p1316]

— **defun bcDraw2DparGen** —

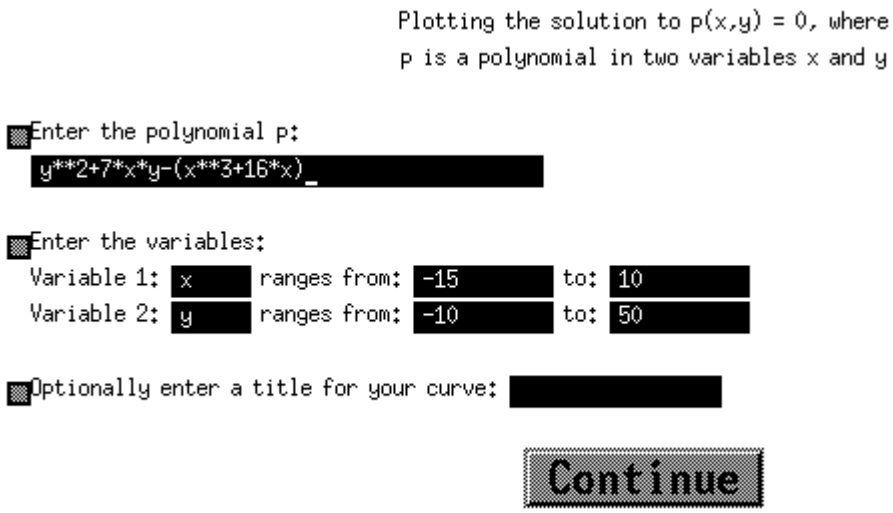
```
(defun |bcDraw2DparGen| (htPage)
 (let (fun1 fun2 ind from1 to1 title curvePart titlePart)
  (setq fun1  (|htpLabelInputString| htPage '|function1|))
  (setq fun2  (|htpLabelInputString| htPage '|function2|))
  (setq ind   (|htpLabelInputString| htPage '|ind|))
  (setq from1 (|htpLabelInputString| htPage '|from1|))
  (setq to1   (|htpLabelInputString| htPage '|to1|))
  (setq title (|htpLabelInputString| htPage '|title|))
  (setq curvePart (concat "curve(" "{}" fun1 ",{}" fun2 ")"))
  (cond
   ((not (string-equal title ""))
     (setq titlePart (concat "{}" "title ==\"" title "\""))
     (|bcFinish| "draw" curvePart (|bcDrawIt2| ind from1 to1) titlePart))
    (t
     (|bcFinish| "draw" curvePart (|bcDrawIt2| ind from1 to1))))))
```

———-

46.3.6 defun Draw - Solution to a polynomial equation

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]
[$EmptyMode p629]

Draw Basic Command by Equation Solution



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| "\\centerline{Plotting the solution to {\\em p(x,y) = 0}, where} "
"\\centerline{{\\em p} is a polynomial in two variables {\\em x} and {\\em y}}"
"\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}Enter the {\\em polynomial} p:"
           "\\newline\\tab{2}")
    (|bcStrings| (40 "y**2+7*x*y-(x**3+16*x)" |function| EM))
    (|text| .
    "\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}Enter the {\\em variables}:")
    (|text| . "\\newline\\tab{2}{\\em Variable 1:} ")
    (|bcStrings| (4 |x| |independent1| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 -15 |from1| F)) (|text| . "{\\em to:}")
    (|bcStrings| (9 10 |to1| F))
    (|text| . "\\newline\\tab{2}{\\em Variable 2:} ")
    (|bcStrings| (4 |y| |independent2| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 -10 |from2| F)) (|text| . "{\\em to:}")
    (|bcStrings| (9 50 |to2| F))
    (|text| "\\indent{0}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
           "Optionally enter a {\\em title} for your curve:")
    (|bcStrings| (15 "" |title| S)) (|text| . "\\indent{0}")))
 (|htMakeDoneButton| "Continue" '|bcDraw2DSolveGen|)
 (|htShowPage|))
```

———-

46.3.7 defun bcDraw2DSolveGen

[htpLabelInputString p1342]
[concat p1107]
[bcFinish p1333]

— **defun bcDraw2DSolveGen** —

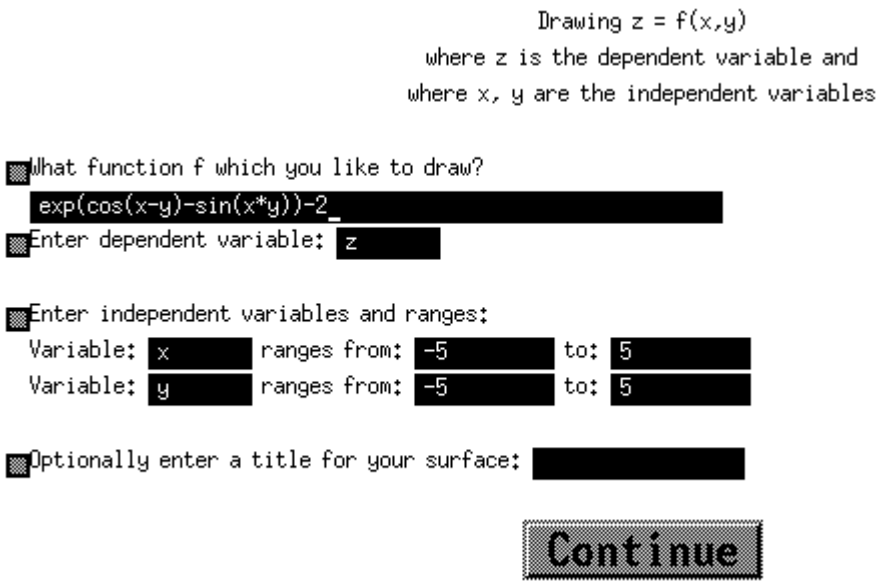
```
(defun |bcDraw2DSolveGen| (htPage)
 (let (fun ind1 from1 to1 ind2 from2 to2 title clipPart titlePart)
  (setq fun   (|htpLabelInputString| htPage '|function|))
  (setq ind1  (|htpLabelInputString| htPage '|independent1|))
  (setq from1 (|htpLabelInputString| htPage '|from1|))
  (setq to1   (|htpLabelInputString| htPage '|to1|))
  (setq ind2  (|htpLabelInputString| htPage '|independent2|))
  (setq from2 (|htpLabelInputString| htPage '|from2|))
  (setq to2   (|htpLabelInputString| htPage '|to2|))
  (setq title (|htpLabelInputString| htPage '|title|))
  (setq clipPart (concat "{}" "range==[{}"
                    from1 ".." to1 '|,{}|
                    from2 ".." to2 "]"))
  (cond
   ((not (string-equal title ""))
    (setq titlePart (concat "{}" "title ==\"" title "\""))
    (|bcFinish| "draw" (concat fun " = 0 ") ind1 ind2 clipPart titlePart))
   (t
     (|bcFinish| "draw" (concat fun " = 0 ") ind1 ind2 clipPart)))))
```

———-

46.3.8 defun Draw - Function of two variables

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Draw Basic Command by 3D function



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 "
"\\centerline{where {\\em x, y} are the independent variables}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
        "What {\\em function} f which you like to draw?\\newline\\tab{2}")
    (|bcStrings| (55 "exp(cos(x-y)-sin(x*y))-2" |function| EM))
    (|text| .
       "\\newline\\menuitemstyle{}\\tab{2}Enter {\\em dependent} variable:")
    (|bcStrings| (6 |z| |dependent| SY))
    (|text| "\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}"
            "Enter {\\em independent} variables and ranges:\\newline\\tab{2} "
            "{\\em Variable:}")
    (|bcStrings| (6 |x| |independent1| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 -5 |from1| F)) (|text| . "{\\em to:}")
    (|bcStrings| (9 5 |to1| F))
    (|text| . "\\newline\\tab{2}{\\em Variable:}")
    (|bcStrings| (6 |y| |independent2| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 -5 |from2| F)) (|text| . "{\\em to:}")
    (|bcStrings| (9 5 |to2| F))
    (|text| "\\indent{0}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
            "Optionally enter a {\\em title} for your surface:")
```

```
   (|bcStrings| (15 "" |title| S)) (|text| . "\\indent{0}")
   (|doneButton| "Continue" |bcDraw3DfunGen|)))
 (|htShowPage|))
```

———-

46.3.9 defun bcDraw3DfunGen

[htpLabelInputString p1342]
[concat p1107]
[bcFinish p1333]
[bcDrawIt2 p1316]

— **defun bcDraw3DfunGen** —

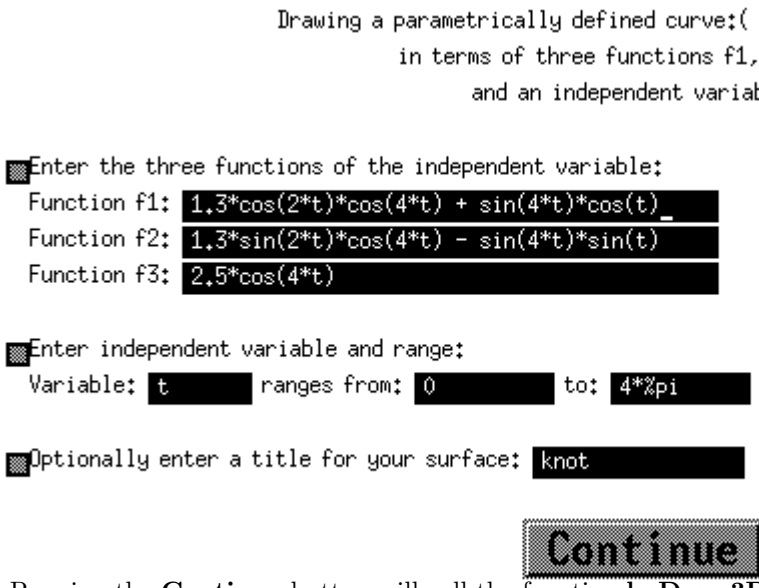
```
(defun |bcDraw3DfunGen| (htPage)
 (let (fun dep ind1 from1 to1 ind2 from2 to2 title titlePart)
  (setq fun   (|htpLabelInputString| htPage '|function|))
  (setq dep   (|htpLabelInputString| htPage '|dependent|))
  (setq ind1  (|htpLabelInputString| htPage '|independent1|))
  (setq from1 (|htpLabelInputString| 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
    ((not (string-equal title ""))
      (setq titlePart (concat "{}" "title ==\"" title "\""))
      (|bcFinish| "draw" fun
         (|bcDrawIt2| ind1 from1 to1)
         (|bcDrawIt2| ind2 from2 to2) titlePart))
    (t
      (|bcFinish| "draw" fun
         (|bcDrawIt2| ind1 from1 to1)
         (|bcDrawIt2| ind2 from2 to2))))))
```

———-

46.3.10 defun Draw - Parametrically defined tube

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Draw Basic Command by 3D parameterized tube



Pressing the **Continue** button will call the function **bcDraw3DparGen** due to this line:

```
(|doneButton| "Continue" |bcDraw3DparGen|)
```

— defun bcDraw3Dpar —

```
(defun |bcDraw3Dpar| ()
 (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 curve:"
            "{\\em ( f1(t), f2(t), f3(t) )}}\\newline "
"\\centerline{in terms of three functions {\\em f1}, {\\em f2}, and {\\em f3}}\\newline "
"\\centerline{and an independent variable {\\em t}}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
            "Enter the three {\\em functions} of the independent variable:")
    (|text| . "\\newline\\tab{2}{\\em Function f1:}")
    (|bcStrings| (42 "1.3*cos(2*t)*cos(4*t) + sin(4*t)*cos(t)" |function1| EM))
    (|text| . "\\newline\\tab{2}{\\em Function f2:}")
    (|bcStrings| (42 "1.3*sin(2*t)*cos(4*t) - sin(4*t)*sin(t)" |function2| EM))
    (|text| . "\\newline\\tab{2}{\\em Function f3:}")
    (|bcStrings| (42 "2.5*cos(4*t)" |function3| EM))
    (|text| .
"\\vspace{1}\\newline\\menuitemstyle{}\\tab{2}Enter {\\em independent} variable and range:\\newline\\tab{2} ")
    (|text| . "{\\em Variable:}") (|bcStrings| (6 |t| |ind| SY))
    (|text| . "ranges {\\em from:}")
    (|bcStrings| (9 0 |from1| F)) (|text| "{\\em to:}")
    (|bcStrings| (9 "4*\\%pi" |to1| F))
    (|text| "\\indent{0}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
            "Optionally enter a {\\em title} for your surface:")
    (|bcStrings| (15 "knot" |title| S)) (|text| . "\\indent{0}")
    (|doneButton| "Continue" |bcDraw3DparGen|)))
 (|htShowPage|))
```

———-

46.3.11 defun bcDraw3DparGen

[htpLabelInputString p1342]
[concat p1107]
[bcFinish p1333]
[bcDrawIt2 p1316]

— **defun bcDraw3DparGen** —

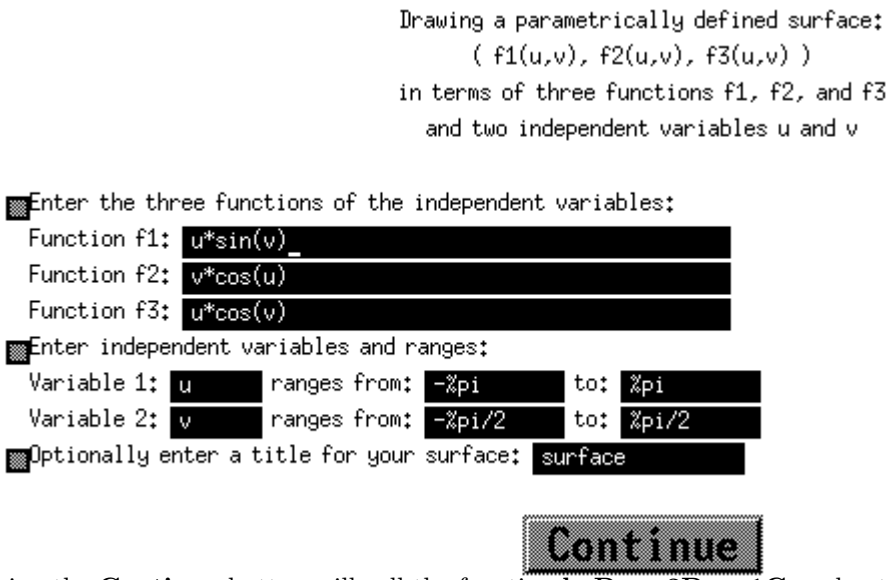
```
(defun |bcDraw3DparGen| (htPage)
 (let (fun1 fun2 fun3 ind from1 to1 title curvePart tubePart titlePart)
  (setq fun1  (|htpLabelInputString| htPage '|function1|))
  (setq fun2  (|htpLabelInputString| htPage '|function2|))
  (setq fun3  (|htpLabelInputString| htPage '|function3|))
  (setq ind   (|htpLabelInputString| htPage '|ind|))
  (setq from1 (|htpLabelInputString| htPage '|from1|))
  (setq to1   (|htpLabelInputString| htPage '|to1|))
  (setq title (|htpLabelInputString| htPage '|title|))
  (setq curvePart (concat "curve(" "{}" fun1 ",{}" fun2 ",{}" fun3 ")"))
  (setq tubePart "{}tubeRadius==.25,{}tubePoints==16")
  (cond
    ((not (string-equal title ""))
      (setq titlePart (concat "{}" "title ==\"" title "\""))
      (|bcFinish| "draw" curvePart
         (|bcDrawIt2| ind from1 to1) tubePart titlePart))
    (t
      (|bcFinish| "draw" curvePart
         (|bcDrawIt2| ind from1 to1) tubePart)))))
```

———-

46.3.12 defun Draw - Parametrically defined surface

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]
[$EmptyMode p629]

Draw Basic Command by 3D parameterized function



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|))
```

———-

46.3.13 defun bcDraw3Dpar1Gen

[htpLabelInputString p1342]
[bcDrawIt2 p1316]
[concat p1107]
[bcFinish p1333]

— **defun bcDraw3Dpar1Gen** —

```
(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 (concat "surface(" "{}" fun1 ",{}" fun2 ",{}" fun3 ")"))
   (cond
     ((not (string= title ""))
       (setq titlePart (concat "{}" "title ==\"" title "\""))
       (|bcFinish| "draw" surfacePart r1 r2 titlePart))
     (t (|bcFinish| "draw" surfacePart r1 r2)))))
```

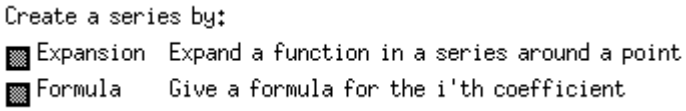
———-

46.4 Series

46.4.1 defun Basic Commands - Series

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Matrix Basic Command



— defun bcSeries —

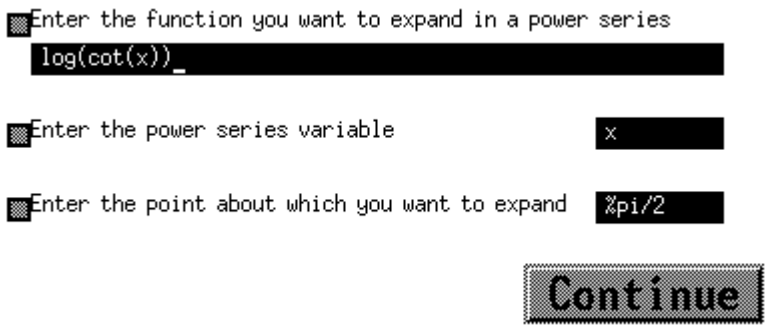
```
(defun |bcSeries| ()
 (declare (special |$EmptyMode|))
 (|htInitPage| "Series Basic Command" nil)
 (|htMakePage|
  '(((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
    (|text| . "Create a series by: ") (|text| . "\\beginmenu")
    (|text| . "\\item ")
    (|bcLinks| ("\\menuitemstyle{Expansion}" "" |bcSeriesExpansion| nil))
    (|text| . "\\tab{11}Expand a function in a series around a point")
    (|text| . "\\item ")
    (|bcLinks| ("\\menuitemstyle{Formula}" "" |bcSeriesByFormula| nil))
    (|text| . "\\tab{11}Give a formula for the {\\em i}'th coefficient")
    (|text| . "\\endmenu")))
 (|htShowPage|))
```

———-

46.4.2 defun Series - Expansion

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]
[$EmptyMode p629]

Series Basic Command expand around a point



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 ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em function} you want to expand in a power series")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "log(cot(x))" |function| EM))
    (|text| . "\\blankline ")
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em power series variable}")
    (|text| . "\\tab{49}")
    (|bcStrings| (8 |x| |variable| SY))
    (|text| . "\\blankline ")
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em point} about which you want to expand")
    (|text| . "\\tab{49}")
    (|bcStrings| (8 "\\%pi/2" |point| EM))))
 (|htMakeDoneButton| "Continue" '|bcSeriesExpansionGen|)
 (|htShowPage|))
```

———-

46.4.3 defun bcSeriesExpansionGen

[htpLabelInputString p1342]
[concat p1107]

[bcFinish p1333]

— defun bcSeriesExpansionGen —

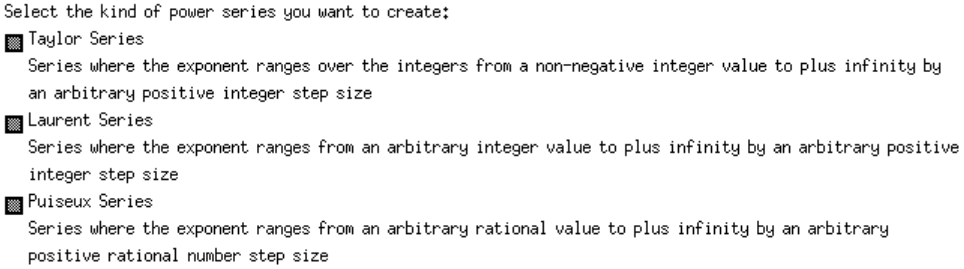
```
(defun |bcSeriesExpansionGen| (htPage)
 (let (fun var point terms)
  (setq fun (|htpLabelInputString| htPage '|function|))
  (setq var (|htpLabelInputString| htPage '|variable|))
  (setq point (|htpLabelInputString| htPage '|point|))
  (setq terms (|htpLabelInputString| htPage '|numberOfTerms|))
  (|bcFinish| "series" fun (concat var " = " point))))
```

———-

46.4.4 defun Series - Formula

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]

Series Basic Command series by formula



— defun bcSeriesByFormula —

```
(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 {\\em non-negative integer} value to plus i
    (|text| . "\\item ")
    (|bcLinks|
      ("\\menuitemstyle{Laurent Series}" "" |bcLaurentSeries| |laurent|))
    (|text| .
"\\newline Series where the exponent ranges from an arbitrary {\\em integer} value to plus infinity by an arbitr
    (|text| . "\\item ")
    (|bcLinks|
```

```
      ("\\menuitemstyle{Puiseux Series}" "" |bcPuiseuxSeries| |puiseux|))
    (|text| .
"\\newline Series where the exponent ranges from an arbitrary {\\em rational value} to plus infinity by an arbit
    (|text| . "\\endmenu")))
 (|htShowPage|))
```

———-

46.4.5 defun Series - Formula - Taylor Series

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Taylor Series Basic Command

Enter the formula for the general coefficient of the series
1/factorial(i)_

Enter the index variable for your formula i
Enter the power series variable x
Enter the point about which you want to expand 0

For Taylor Series, the exponent of the power series variable ranges from an initial value, an arbitrary non-negative integer, to plus infinity; the step size is any positive integer.

Enter the initial value of the index (an integer) 0
Enter the step size (a positive integer) 1

Continue

Pressing the **Continue** button will call the function **bcTaylorSeriesGen** due to this line:

```
   (|doneButton| "Continue" |bcTaylorSeriesGen|)
```

— **defun bcTaylorSeries** —

```
(defun |bcTaylorSeries| (a b)
 (declare (ignore a b))
 (declare (special |$EmptyMode|))
 (|htInitPage| "Taylor Series Basic Command" NIL)
 (|htMakePage|
  '((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| EEM (|Expression| |$EmptyMode|))
        (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
    (|text| . "\\menuitemstyle{}\\tab{2}")
```

```
    (|text| . "Enter the formula for the general coefficient of the series")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "1/factorial(i)" |formula| EM))
    (|text| . "\\blankline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em index variable} for your formula")
    (|text| . "\\tab{49}") (|bcStrings| (8 |i| |index| SY))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em power series variable}")
    (|text| . "\\tab{49}") (|bcStrings| (8 |x| |variable| SY))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em point} about which you want to expand")
    (|text| . "\\tab{49}") (|bcStrings| (8 0 |point| EM))
    (|text| . "\\blankline ")
    (|text| .
"For Taylor Series, the exponent of the power series variable ranges  from an {\\em initial value}, an arbitrary
    (|text| . "\\blankline ") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em initial value} of the index (an integer)")
    (|text| . "\\tab{49}") (|bcStrings| (8 "0" |min| I))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em step size} (a positive integer)")
    (|text| . "\\tab{49}") (|bcStrings| (8 "1" |step| PI))
    (|doneButton| "Continue" |bcTaylorSeriesGen|)))
 (|htShowPage|))
```

———-

46.4.6 defun bcTaylorSeriesGen

[bcSeriesGen p1281]

— **defun bcTaylorSeriesGen** —

```
(defun |bcTaylorSeriesGen| (htPage)
 (|bcSeriesGen| htPage))
```

———-

46.4.7 defun bcSeriesGen

[htpLabelInputString p1342]
[concat p1107]
[bcFinish p1333]

— **defun bcSeriesGen** —

```
(defun |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 (concat var " = " point))
(setq minPart (concat min ".."))
(|bcFinish| "series" (concat index " +-> " formula) varPart minPart step)))
```

———-

46.4.8 defun Series - Formula - Laurent Series

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Laurent Series Basic Command

Enter the formula for the general coefficient of the series
(-1)**(n - 1)/(n + 2)_

Enter the index variable for your formula n
Enter the power series variable x
Enter the point about which you want to expand 0

For Laurent Series, the exponent of the power series variable ranges from an initial value, an arbitrary integer value, to plus infinity; the step size is any positive integer.

Enter the initial value of the index (an integer -1
Enter the step size (a positive integer) 1

Continue

Pressing the **Continue** button will call the function **bcLaurentSeriesGen** due to this line:

```
(|doneButton| "Continue" |bcLaurentSeriesGen|)
```

— defun bcLaurentSeries —

```
(defun |bcLaurentSeries| (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| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the formula for the general coefficient of the series")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "(-1)**(n - 1)/(n + 2)" |formula| EM))
    (|text| . "\\vspace{1}\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em index variable} for your formula")
    (|text| . "\\tab{49}") (|bcStrings| (8 |n| |index| SY))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em power series variable}")
    (|text| . "\\tab{49}") (|bcStrings| (8 |x| |variable| SY))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em point} about which you want to expand")
    (|text| . "\\tab{49}")
    (|bcStrings| (8 0 |point| F))
    (|text| . "\\blankline")
    (|text| .
"\\newline For Laurent Series, the exponent of the power series variable ranges from an {\\em initial value}, an
    (|text| . "\\blankline")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em initial value} of the index (an integer)")
    (|text| . "\\tab{49}") (|bcStrings| (8 "-1" |min| I))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em step size} (a positive integer)")
    (|text| . "\\tab{49}") (|bcStrings| (8 "1" |step| PI))
    (|doneButton| "Continue" |bcLaurentSeriesGen|)))
 (|htShowPage|))
```

———-

46.4.9 defun bcLaurentSeriesGen

[bcSeriesGen p1281]

— **defun bcLaurentSeriesGen** —

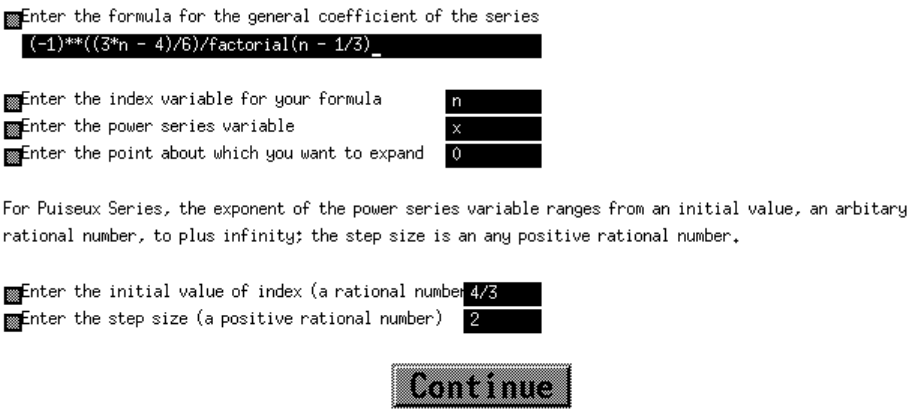
```
(defun |bcLaurentSeriesGen| (htPage)
 (|bcSeriesGen| htPage))
```

———-

46.4.10 defun Series - Formula - Puiseux Series

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Puiseux Series Basic Command



Pressing the **Continue** button will call the function **bcPuiseuxSeriesGen** due to this line:

```
(|doneButton| "Continue" |bcPuiseuxSeriesGen|)
```

— **defun bcPuiseuxSeries** —

```
(defun |bcPuiseuxSeries| (a b)
 (declare (special |$EmptyMode|) (ignore a b))
 (|htInitPage| "Puiseux Series Basic Command" nil)
 (|htMakePage|
  '((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| EEM (|Expression| |$EmptyMode|))
        (|isDomain| S (|String|)) (|isDomain| I (|Integer|))
        (|isDomain| PI (|PositiveInteger|))
        (|isDOmain| RN (|Fraction| (|Integer|)))
        (|isDomain| SY (|Symbol|)))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| .
      "Enter the {\\em formula} for the general coefficient of the series")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (55 "(-1)**((3*n - 4)/6)/factorial(n - 1/3)" |formula| EM))
    (|text| . "\\vspace{1}\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em index variable} for your formula")
    (|text| . "\\tab{49}") (|bcStrings| (8 |n| |index| SY))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em power series variable}")
    (|text| . "\\tab{49}") (|bcStrings| (8 |x| |variable| SY))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em point} about which you want to expand")
    (|text| . "\\tab{49}") (|bcStrings| (8 0 |point| F))
    (|text| . "\\blankline ")
    (|text| .
```

```
"For Puiseux Series, the exponent of the power series variable ranges from an {\\em initial value}, an arbitary
    (|text| . "\\blankline ") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em initial value} of index (a rational number)")
    (|text| . "\\tab{51}") (|bcStrings| (6 "4/3" |min| RN))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em step size} (a positive rational number)")
    (|text| . "\\tab{51}") (|bcStrings| (6 "2" |step| RN))
    (|doneButton| "Continue" |bcPuiseuxSeriesGen|)))
 (|htShowPage|))
```

———-

46.4.11 defun bcPuiseuxSeriesGen

[bcSeriesGen p1281]

— **defun bcPuiseuxSeriesGen** —

```
(defun |bcPuiseuxSeriesGen| (htPage)
 (|bcSeriesGen| htPage))
```

———-

46.4.12 defun Solve Basic Command

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Solve Basic Command

What do you want to solve?
A System Of Linear Equations
A System of Polynomial Equations
A Single Polynomial Equation

— **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}" ""
         |bcLinearSolve| |linear|))
```

```
    (|text| . "\\item ")
    (|bcLinks|
      ("\\menuitemstyle{A System of Polynomial Equations}" ""
         |bcSystemSolve| |polynomial|))
    (|text| . "\\item ")
    (|bcLinks|
      ("\\menuitemstyle{A Single Polynomial Equation}" ""
         |bcSolveSingle| |onePolynomial|))
     (|text| . "\\endmenu")))
 (|htShowPage|))
```

———-

46.4.13 defun Solve - System of Linear Equations

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Linear Solve Basic Command

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 bcLinearSolve —

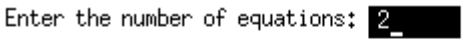
```
(defun |bcLinearSolve| (p nn)
 (declare (ignore p nn))
 (|htInitPage| "Basic Solve Command" NIL)
 (|htMakePage|
  '((|text| . "How do you want to enter the equations?")
    (|text| . "\\beginmenu")
    (|text| . "\\item ")
    (|text| . "\\newline ")
    (|bcLinks|
       ("\\menuitemstyle{Directly as equations}" ""
          |bcLinearSolveEqns| |equations|))
    (|text| . "\\item ")
    (|text| . "\\newline ")
    (|bcLinks|
       ("\\menuitemstyle{In matrix form}" ""
          |bcLinearSolveMatrix| |matrix|))
    (|text| . "\\indentrel{16}\\tab{0}")
    (|text| .
" \\spad{AX = B}, where \\spad{A} is a matrix of coefficients and \\spad{B} is a vector")
    (|text| . "\\indentrel{-16}\\item ")
    (|text| . "\\endmenu")))
 (|htShowPage|))
```

———-

46.4.14 defun System of Linear Equations - Directly as equations

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]
[$EmptyMode p629]

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|))
```

———-

46.4.15 defun bcLinearSolveEqns1

[htpSetProperty p1342]
[bcInputEquations p1323]

— **defun bcLinearSolveEqns1** —

```
(defun |bcLinearSolveEqns1| (htPage)
 (|htpSetProperty| htPage '|systemType| '|linear|)
 (|htpSetProperty| htPage '|exitFunction| '|bcLinearSolveEqnsGen|)
 (|bcInputEquations| htPage '|exact|))
```

———-

46.4.16 defun System of Linear Equations - In matrix form

[bcReadMatrix p1318]
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** —

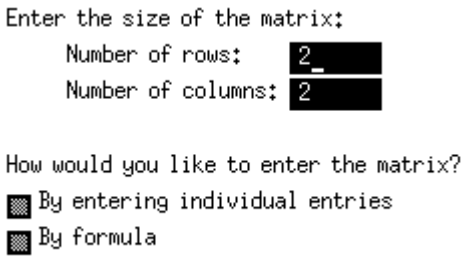
```
(defun |bcLinearSolveMatrix| (htPage junk)
 (declare (ignore htPage junk))
 (|bcReadMatrix| '|bcLinearSolveMatrix1|))
```

———-

46.4.17 defun System of Linear Equations - In matrix form direct

[htInitPage p1349]
[htpSetProperty p1342]
[htMakePage p1351]
[htShowPage p1350]

Matrix Basic Command



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 **exitFunction** 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| . "\\beginmenu")
      (|text| . "\\item ")
```

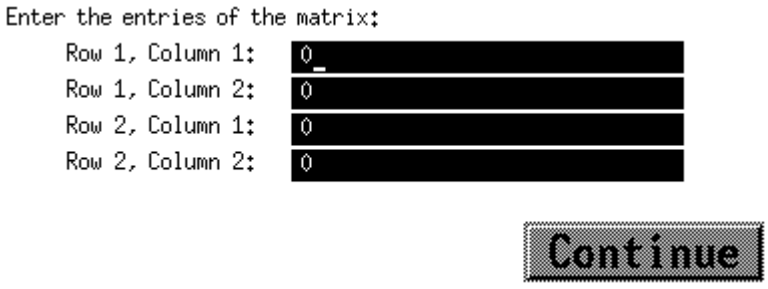
```
    (|bcLinks|
               ("\\menuitemstyle{By entering individual entries}" ""
                |bcInputExplicitMatrix| |explicit|))
    (|text| . "\\item ")
    (|bcLinks|
               ("\\menuitemstyle{By formula}" ""
                |bcInputMatrixByFormula| |formula|))
    (|text| . "\\endmenu")))
 (|htShowPage|)))
```

———-

46.4.18 defun Solve System of Linear Equations Individual

[objValUnwrap p462]
[htpLabelSpadValue p1344]
[htpLabelInputString p1342]
[parse-integer p??]
[length p??]
[nreverse0 p??]
[concat p1107]
[htInitPage p1349]
[htpPropertyList p1342]
[bcHt p1347]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htpSetProperty p1342]
[htShowPage p1350]
[$EmptyMode p629]
[$bcParseOnly p1338]

Input Explicit Matrix



Pressing the **Continue** button will call the function **bcGenExplicitMatrix** due to this line:

```
(|htMakeDoneButton| "Continue" '|bcGenExplicitMatrix|)
```

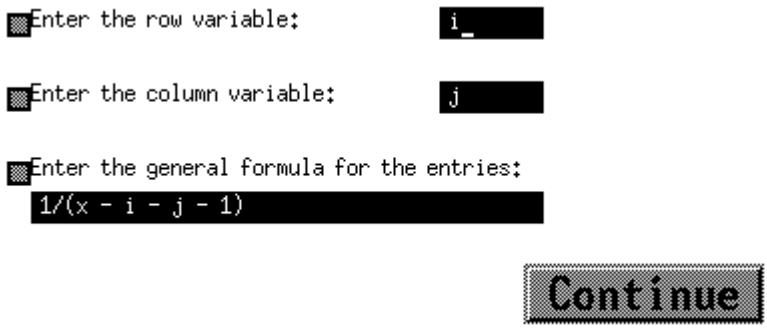
— **defun bcInputExplicitMatrix** —

```
(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 (|#| (princ-to-string nrows)))
  (setq wcols (|#| (princ-to-string ncols)))
  (setq labelList
    (do ((i 1 (1+ i))) ((> i nrows) t1)
      (setq t2 nil)
      (setq t1
       (append t1
         (do ((j 1 (1+ j))) ((> j ncols) (nreverse0 t2))
          (setq t2
           (cons
            (progn
             (setq rowpart (concat "{\\em Row" (|htStringPad| i wrows)))
             (setq colpart (concat ", Column" (|htStringPad| j wcols)
                                   ":}\\space{2}"))
             (setq prefix (concat rowpart colpart))
             (setq name (intern (princ-to-string (setq k (1+ k)))))
             (list prefix "" 30 0 name 'P))
            t2)))))))
  (setq labelList
   (list
     (list '|domainConditions|
           '(|isDomain| P (|Polynomial| |$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|)))
```

46.4.19 defun System of Linear Equations In matrix form by formula

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[objValUnwrap p462]
[htpLabelSpadValue p1344]
[parse-integer p??]
[htpLabelInputString p1342]
[htpSetProperty p1342]
[htShowPage p1350]
[$bcParseOnly p1338]

Input Matrix By Formula



Pressing the **Continue** button will call the function **bcInputMatrixByFormulaGen** due to this line:

```
(|htMakeDoneButton| "Continue" '|bcInputMatrixByFormulaGen|)
```

— 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|))
                  (|isDomain| FE (|Expression| (|Integer|))))
      (|text| . "\\menuitemstyle{}\\tab{2}")
      (|text| . "Enter the {\\em row variable}: ")
      (|text| . "\\tab{36}") (|bcStrings| (6 |i| |rowVar| S))
      (|text| . "\\blankline ") (|text| . "\\newline ")
      (|text| . "\\menuitemstyle{}\\tab{2}")
      (|text| . "Enter the {\\em column variable}: ")
      (|text| . "\\tab{36}") (|bcStrings| (6 |j| |colVar| S))
      (|text| . "\\blankline ") (|text| . "\\newline ")
```

```
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the general {\\em formula} for the entries:")
    (|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|))))
 (|htpSetProperty| page '|nrows| nrows)
 (|htpSetProperty| page '|ncols| ncols)
 (|htShowPage|)))
```

———-

46.4.20 defun Solve - System of Polynomial Equations

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]

Solve Directly As Equations

Enter the number of equations: 2

Continue

Pressing the **Continue** button will call the function **bcSystemSolveEqns1** due to this line:

```
(|htMakeDoneButton| "Continue" '|bcSystemSolveEqns1|)
```

— **defun bcSystemSolve** —

```
(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|))
```

———-

46.4.21 defun bcSystemSolveEqns1

[htpSetProperty p1342]
[bcInputEquations p1323]

— defun bcSystemSolveEqns1 —

```
(defun |bcSystemSolveEqns1| (htPage)
 (|htpSetProperty| htPage '|systemType| '|polynomial|)
 (|htpSetProperty| htPage '|exitFunction| '|bcInputSolveInfo|)
 (|bcInputEquations| htPage '|exact|))
```

———-

46.4.22 defun bcInputSolveInfo

[htInitPage p1349]
[htpPropertyList p1342]
[htpSetProperty p1342]
[htpInputAreaList p??]
[htMakePage p1351]
[htShowPage p1350]

— 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?")
      (|text| . "\\beginmenu") (|text| . "\\item ")
      (|bcLinks|
        ("\\menuitemstyle{Exact Solutions}" "" |bcSolveEquations| |exact|))
      (|text| . "\\indentrel{18}\\tab{0} ")
      (|text| .
     "Solutions expressed in terms of {\\em roots} of irreducible polynomials")
      (|text| . "\\indentrel{-18}")
      (|text| . "\\item ")
      (|bcLinks|
        ("\\menuitemstyle{Numeric Solutions}" ""
           |bcSolveEquationsNumerically| |numeric|))
      (|text| . "\\indentrel{18}\\tab{0} ")
      (|text| .
   "Solutions expressed in terms of approximate real or complex {\\em numbers}")
      (|text| . "\\indentrel{-18}")
      (|text| . "\\item ")
```

```
      (|bcLinks|
        ("\\menuitemstyle{Radical Solutions}" "" |bcSolveEquations| |radical|))
      (|text| . "\\indentrel{18}\\tab{0} ")
      (|text| .
          "Solutions expressed in terms of {\\em radicals} if it is possible")
      (|text| . "\\indentrel{-18}")
      (|text| . "\\endmenu")))
  (|htShowPage|)))
```

———-

46.4.23 defun Solve - Single Polynomial Equation

[htpSetProperty p1342]
[bcInputEquations p1323]

— **defun bcSolveSingle** —

```
(defun |bcSolveSingle| (htPage p)
 (declare (ignore p))
 (|htpSetProperty| htPage '|systemType| '|onePolynomial|)
 (|htpSetProperty| htPage '|exitFunction| '|bcInputSolveInfo|)
 (|bcInputEquations| htPage '|exact|))
```

———-

Chapter 47

HyperDoc Reference

47.1 Book

47.2 Topics

47.2.1 Numbers

Numbers - Integers Numbers - Fractions Numbers - Machine Floats Numbers - Real Numbers Numbers - Complex Numbers Numbers - Finite Fields Finite Fields - Modular Arithmetic and Prime Fields Finite Fields - Extensions of Finite Fields Extensions - Finite Field Extensions - FinitFieldCyclicGroup Extensions - FinitFieldNormalBasis Extensions - FinitFieldExtension Extensions - FinitFieldExtensionByPolynomial Extensions - FinitFieldCyclicGroupExtension Extensions - FinitFieldCyclicGroupExtensionByPolynomial Extensions - FinitFieldNormalBasisExtension Extensions - FinitFieldNormalBasisExtensionByPolynomial Finite Fields - Irreducible Modulus Polynomial Representations Finite Fields - Cyclic Group Representations Finite Fields - Normal Basis Representations Finite Fields - Conversion Operations for Finite Fields Finite Fields - Utility Operations for Finite Fields Numbers - Numeric Functions Numbers - Cardinal Numbers Numbers - Machine-size Integers Numbers - Roman Numerals Numbers - Continued Fractions Numbers - Partial Fractions Numbers - Quaternions Numbers - Octonions Numbers - Repeating Decimals Numbers - Repeating Binary Expansions Numbers - Repeating Hexadecimal Expansions Numbers - Expansions in other Bases

47.2.2 Polynomials

Polynomials - Basic Functions Polynomials - Substitutions Polynomials - Factorization Polynomials - GCDs and Friends Polynomials - Roots Polynomials - Specific Types

47.2.3 Functions

Functions - Rational Functions Functions - Algebraic Equations Functions - Elementary Functions Functions - Special Functions Functions - Unknown Functions Functions - Simplification Functions - Pattern Matching Functions - Operator Algebra

47.2.4 Solving Equations

Solving Equations - Systems of Linear Equations Solving Equations - Single Polynomial Equation Solving Equations - Systems of Polynomial Equations Solving Equations - Differential Equations

47.2.5 Calculus

Calculus - Limits Calculus - Derivatives Calculus - Integrals Calculus - More Integrals Calculus - Laplace Calculus - Series Calculus - Differential Equations

47.2.6 Linear Algebra

Linear Algebra - Introduction Linear Algebra - Creating Matrices Linear Algebra - Operations on Matrices Linear Algebra - Eigenvalues and Eigenvectors Linear Algebra - Example: Determinant of a Hilbert Matrix Linear Algebra - Computing the Permanent Linear Algebra - Working with Vectors Linear Algebra - Working with Square Matrices Linear Algebra - Working with One-Dimensional Arrays Linear Algebra - Working with Two-Dimensional Arrays Linear Algebra - Conversion (Polynomials of Matrices)

47.2.7 Graphics

Graphics - Examples Examples - Assorted Examples Examples - Three Dimensional Graphics Examples - Functions of One Variable Examples - Parametric Curves Examples - Polar Coordinates Examples - Implicit Curves Examples - Lists of Points Graphics - 2D Graphics Graphics - 3D Graphics Graphics - Viewports

47.2.8 Algebra

Algebra - Number Theory Number Theory - Galois Groups Number Theory - Number Theory Functions Algebra - Group Theory Group Theory - Info on Group Theory Group Theory - Info on Representation Theory Group Theory - Representations of A6

47.3 Language

47.4 Examples

47.5 Commands

47.6 Glossary

47.7 Hyperdoc

47.8 Search

Chapter 48

HyperDoc Topics

Chapter 49

HyperDoc Browse

Chapter 50

HyperDoc Examples

Chapter 51

HyperDoc Settings

Chapter 52

HyperDoc About

Chapter 53

HyperDoc What's New

Chapter 54

HyperDoc Support Functions

54.1 Handling page creation and deletion

54.1.1 defvar $activePageList

— **initvars** —

```
(defvar |$activePageList| nil)
```

————-

54.1.2 defun htpMakeEmptyPage

[$activePageList p1311]

— **defun htpMakeEmptyPage 0** —

```
(defun |htpMakeEmptyPage| (&rest args)
  (let (name val (propList (car args)) (options (cdr args)))
  (declare (special |$activePageList|))
   (setq name (or (car options) (gentemp)))
   (setq |$activePageList| (cons name |$activePageList|))
   (set name (setq val (vector name nil nil nil nil nil propList nil)))
   val))
```

————-

54.1.3 defun htpDestroyPage

[member p1108]
[$activePageList p1311]

— **defun htpDestroyPage** —

```
(defun |htpDestroyPage| (pageName)
 (declare (special |$activePageList|))
 (when (|member| pageName |$activePageList|)
   (set pageName nil)
   (setq |$activePageList| (delete pageName |$activePageList|))))
```

———-

54.2 Handling Axiom command execution

The bcGen function is called with a string which will be passed to the Axiom command line. For example, the path

```
Calculus -> Differentiate -> -> Continue -> Do it
```

will generate the Axiom command line

```
differentiate(sin(x*y),[x,y],[1,2])
```

54.2.1 defun Basic Command result page

[concat p1107]
[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]

bcGen : Command → nil

— defun bcGen —

```
(defun |bcGen| (command)
 (let (string)
  (|htInitPage| "Basic Command" nil)
  (setq string
    (if (< (length command) 50)
       (concat "{\\centerline{\\tt " command " }}")
       (concat "{\\tt " command " }")))
  (|htMakePage|
    (list
     '(|text|
   "{Here is the Axiom command you could have issued to compute this result:}"
        "\\vspace{2}\\newline ")
       (cons '|text| string) ))
  (|htMakeDoitButton| "Do It" command)
  (|htShowPage|)))
```

———-

The htMakeDoitButton displays the Axiom command to be executed on a page. Pushing the **"Do it"** invokes doDoitButton

54.2.2 defun htMakeDoitButton

doDoitButton : String,Command → nil
— defun htMakeDoitButton —

```
(defun |htMakeDoitButton| (label command)
 (declare (special |$curPage|))
  (cond
   ((equal label "Do It")
    (|bcHt|
"\\newline\\vspace{1}\\centerline{\\lispcommand{\\DoItBitmap}{(|doDoitButton| "))
    (t
     (|bcHt|
      (list "\\newline\\vspace{1}\\centerline{\\lispcommand{\\box{" label
         "}}{(|doDoitButton| "))))
  (|bcHt| (|htpName| |$curPage|))
  (|bcHt| (list " \"" (|htEscapeString| command) "\""))
  (|bcHt| ")}}")
  (|bcHt| "\\vspace{2}{Select \\  \\UpButton{} \\  to go back one page.}")
  (|bcHt|
    "\\newline{Select \\  \\ExitButton{QuitPage} \\  to remove this window.}"))
```

———-

The doDoitButton invokes executeInterpreterCommand. Why this intermediate function exists is unclear.

54.2.3 defun Execute a command from Hyperdoc

doDoitButton : Command → nil
— defun doDoitButton —

```
(defun |doDoitButton| (htPage command)
 (declare (ignore htPage))
 (|executeInterpreterCommand| command))
```

———-

The doDoitButton function passes a valid Axiom Command (see Volume 13[Book13] for the definition of the "Command" type). This calls the Axiom command line to execute the command.

54.2.4 defun Execute a string in the interpreter

executeInterpreterCommand : Command → nil
— defun executeInterpreterCommand —

```
(defun |executeInterpreterCommand| (command)
  (princ command)
```

```
(terpri)
(|setCurrentLine| command)
(catch 'spad_reader (|parseAndInterpret| command))
(princ (mkprompt))
(finish-output))
```

54.3 Functions creating pages

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
  - **lispdownlink** 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.

54.3.1 defun Basic Command Matrix by Formula generate

[htpProperty p1342]
[htpLabelInputString p1342]
[bcGen p1334]
[concat p1107]

— **defun bcInputMatrixByFormulaGen** —

```
(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| (concat "matrix([[" formula
                          " for " colVar
                          " in 1.." (princ-to-string ncols)
                          "] for " rowVar
                          " in 1.." (princ-to-string nrows)
                          "])"))))))
```

———-

54.3.2 defun Basic Command generate explicit matrix

[htpSetProperty p1342]
[htpInputAreaAlist p1341]
[htpProperty p1342]
[bcGen p1334]
[bcMatrixGen p1316]

— **defun bcGenExplicitMatrix** —

```
(defun |bcGenExplicitMatrix| (htPage)
 (let (fun)
  (|htpSetProperty| htPage '|matrix| (|htpInputAreaAlist| htPage))
  (if (setq fun (|htpProperty| htPage '|exitFunction|))
    (funcall fun htPage)
    (|bcGen| (|bcMatrixGen| htPage)))))
```

———-

54.3.3 defun Basic Command generate matrix

[htpProperty p1342]
[lassoc p??]
[concat p1107]
[bcwords2liststring p1335]
[systemError p??]

— **defun bcMatrixGen** —

```
(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))
      (concat "matrix([[" formula
                          " for " colVar
                          " in 1.." (princ-to-string ncols)
                          "] for " rowVar
                          " in 1.." (princ-to-string 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))))
      (concat "matrix(" matstring ")"))
    (t (|systemError| nil)))))
```

———-

;–Hypertex commands other than solve and matrix

54.3.4 defun Basic Command iteration

[bcMatrixGen p1316]

— **defun bcDrawIt2** —

```
(defun |bcDrawIt2| (ind a b)
```

```
 (concat "{}" ind "=" a "{}.." b "{}"))
```

———-

54.3.5 defun Sum Basic Command

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

— **defun bcProduct** —

```
(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")
    (|inputStrings|
        ("Enter the {\\em index of the product}:" "" 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|))
```

———-

54.3.6 defun bcProductGen

[htpLabelInputString p1342]
[concat p1107]
[bcGen p1334]

— **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| (concat "product(" mand "," index "," car "," last ")"))))
```

———-

54.3.7 defun Read Matrix

[htInitPage p1349]
[htpSetProperty p1342]
[htMakePage p1351]
[htShowPage p1350]

;–Hypertex commands other than solve and matrix

54.3.8 defun bcSeriesByFormulaGen

[bcNotReady p1336]

— **defun bcSeriesByFormulaGen** —

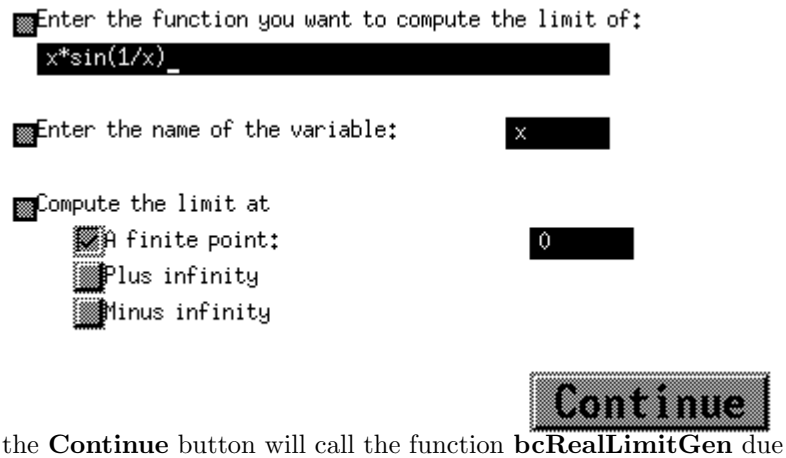
```
(defun |bcSeriesByFormulaGen| (htPage)
 (declare (ignore htPage))
 (|bcNotReady|))
```

———-

54.3.9 defun Real Limit Basic Command

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Real Limit Basic Command



Pressing the **Continue** button will call the function **bcRealLimitGen** due to this line:

```
(|doneButton| "Continue" |bcRealLimitGen|)
```

— defun bcRealLimit —

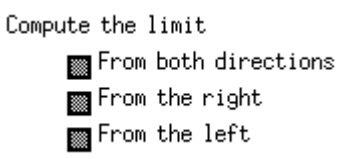
```
(defun |bcRealLimit| (a b)
 (declare (special |$EmptyMode|) (ignore a b))
 (|htInitPage| "Real Limit Basic Command" NIL)
 (|htMakePage|
   '((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| S (|String|)) (|isDomain| F (|Float|))
        (|isDomain| SY (|Symbol|)))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em function} you want to compute the limit of:")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (45 "x*sin(1/x)" |expression| EM))
    (|text| . "\\blankline") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the name of the {\\em variable}: ")
    (|text| . "\\tab{41}")
    (|bcStrings| (6 |x| |variable| SY))
    (|text| . "\\blankline") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Compute the limit at")
    (|radioButtons| |location|
      ("A finite point:"
         ((|text| . "\\tab{33}") (|bcStrings| (6 0 |point| F))) |finitePoint|)
      ("Plus infinity" "" |plusInfinity|)
      ("Minus infinity" "" |minusInfinity|))
    (|doneButton| "Continue" |bcRealLimitGen|)))
 (|htShowPage|))
```

———-

54.3.10 defun Real Limit Basic Command options

[htpButtonValue p1340]
[htpLabelInputString p1342]
[bcFinish p1333]
[htInitPage p1349]
[htMakePage p1351]
[htpSetProperty p1342]
[htShowPage p1350]

Real Limit Basic Command options



— 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| (concat |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|)))))
```

———-

54.3.11 defun bcRealLimitGen1

[htpProperty p1342]
[concat p1107]
[bcFinish p1333]

— defun bcRealLimitGen1 —

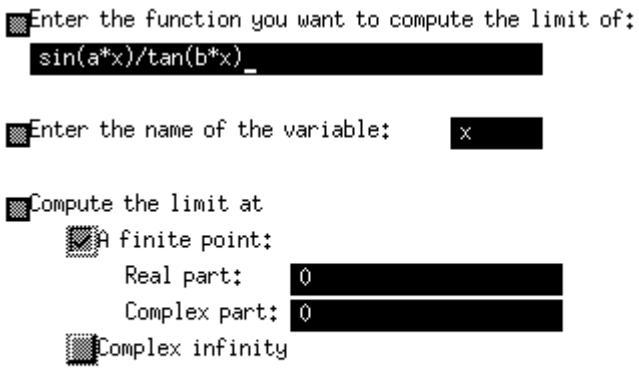
```
(defun |bcRealLimitGen1| (htPage key)
 (let (direction fun var loc varPart)
  (setq direction
   (cond
    ((eq key '|right|) "\"right\"")
    ((eq key '|left|) "\"left\"")
    (t nil)))
  (setq fun (|htpProperty| htPage '|fun|))
  (setq var (|htpProperty| htPage '|var|))
  (setq loc (|htpProperty| htPage '|loc|))
  (setq varPart (concat var " = " loc))
  (|bcFinish| "limit" fun varPart direction)))
```

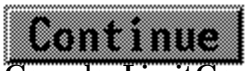
———-

54.3.12 defun Complex Limit Basic Command

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Complex Limit Basic Command





Pressing the **Continue** button will call the function **bcComplexLimitGen** due to this line:

```
(|doneButton| "Continue" |bcComplexLimitGen|)
```

— **defun bcComplexLimit** —

```
(defun |bcComplexLimit| (a b)
 (declare (special |$EmptyMode|) (ignore a b))
 (|htInitPage| "Complex Limit Basic Command" nil)
 (|htMakePage|
  '(((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| S (|String|)) (|isDomain| F (|Float|))
        (|isDomain| SY (|Symbol|)))
    (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the {\\em function} you want to compute the limit of:")
    (|text| . "\\newline\\tab{2} ")
    (|bcStrings| (40 "sin(a*x)/tan(b*x)" |expression| EM))
    (|text| . "\\blankline ") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Enter the name of the {\\em variable}: ")
    (|text| . "\\tab{37}") (|bcStrings| (5 |x| |variable| SY))
    (|text| . "\\blankline ") (|text| . "\\newline ")
    (|text| . "\\menuitemstyle{}\\tab{2}")
    (|text| . "Compute the limit at")
    (|radioButtons| |location|
      ("A finite point:"
```

```
        ((|text| . "\\newline\\space{0}Real part:\\space{3}")
         (|bcStrings| (20 0 |real| F))
         (|text| . "\\newline Complex part:")
         (|bcStrings| (20 0 |complex| F)))
        |finitePoint|)
      ("Complex infinity" "" |complexInfinity|))
    (|doneButton| "Continue" |bcComplexLimitGen|)))
 (|htShowPage|))
```

———-

54.3.13 defun bcComplexLimitGen

[htpLabelInputString p1342]
[htpButtonValue p1340]
[concat p1107]
[bcFinish p1333]

— **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 (concat comp "*%i"))))
      (cond
       ((string= real "0") (if (string= complexPart "") '|0| complexPart))
       ((string= complexPart "") real)
       (t (concat real " + " complexPart))))
    (t "%infinity")))
  (setq varPart (concat var " = " loc))
  (|bcFinish| "complexLimit" fun varPart)))
```

———-

54.3.14 defvar $systemType

— **initvars** —

```
(setq |$systemType| nil)
```

———-

54.3.15 defvar $numberOfEquations

— **initvars** —

```
(defvar |$numberOfEquations| 0)
```

———-

54.3.16 defvar $solutionMethod

— **initvars** —

```
(defvar |$solutionMethod| nil)
```

———-

54.3.17 defun bcInputEquations

[concat p1107]
[bcMakeLinearEquations p1326]
[bcMakeEquations p1325]
[htProperty p??]
[parse-integer p??]
[objValUnwrap p462]
[htInitPage p1349]
[htpPropertyList p1342]
[htpSetProperty p1342]
[htSay p1347]
[htMakePage p1351]
[bcHt p1347]
[bcMakeUnknowns p1325]
[htMakeDoneButton p1369]
[htShowPage p1350]
[$EmptyMode p629]
[$bcParseOnly p1338]

— **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
       (concat "\\newline\\tab{2}{\\em Equation " (princ-to-string i) ":}"))
   (setq prefix (concat prefix "\\space{" (princ-to-string spacer) "}"))
```

```
  (setq lnam (intern (concat "l" (princ-to-string i))))
  (setq rnam (intern (concat "r" (princ-to-string 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 '|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)
   (|$bcParseOnly|
     (parse-integer (|htpLabelInputString| htPage '|numberOfEquations|)))
    (t
     (|objValUnwrap| (|htpLabelSpadValue| htPage '|numberOfEquations|)))))
 (setq linearPred (eq (|htpProperty| htPage '|systemType|) '|linear|))
 (setq labelList
  (cond
    ((eql numEqs 1)
     '((|bcStrings| (42 "x^2+1" l1 p)) (|text| . " = ")
        (|bcStrings| (6 0 r1 P))))
    (t
     (loop for i from 1 to numEqs
      append (f i numEqs linearPred)))))
 (setq equationPart
  (cons '(|domainConditions|
            (|isDomain| P (|Polynomial| |$EmptyMode|))
            (|isDomain| S (|String|))
            (|isDomain| PI (|PositiveInteger|)))
         labelList))
 (setq page (|htInitPage| "Solve Basic Command" (|htpPropertyList| htPage)))
 (|htpSetProperty| page '|numberOfEquations| numEqs)
 (|htpSetProperty| page '|solutionMethod| solutionMethod)
 (|htSay| "\\newline\\menuitemstyle{}\\tab{2}")
 (|htSay| (if (eql numEqs 1)
              "Enter the {\\em Equation}:"
              "Enter the {\\em Equations}:"))
 (|htSay| "\\newline\\tab{2}")
 (|htMakePage| equationPart)
 (|bcHt| "\\blankline ")
 (|htSay| "\\newline\\menuitemstyle{}\\tab{2}")
 (|htMakePage|
   (if (eql numEqs 1)
    '((|text| . "Enter the {\\em unknown} (leave blank if implied): ")
      (|text| . "\\tab{48}")
      (|bcStrings| (6 "x" unknowns S . |quoteString|)))
    (list
     '(|text| . "Enter the unknowns (leave blank if implied):")
     '(|text| . "\\tab{44}")
      (list '|bcStrings|
```

```
                (list 10 (|bcMakeUnknowns| numEqs) '|unknowns| 'p)))))
  (|htMakeDoneButton| "Continue" '|bcInputEquationsEnd|)
  (|htShowPage|))))
```

———-

54.3.18 defun Create a variable string

— defun bcCreateVariableString —

```
(defun |bcCreateVariableString| (i)
  (format nil "x~a" i))
```

———-

54.3.19 defun bcMakeUnknowns

— defun bcMakeUnknowns —

```
(defun |bcMakeUnknowns| (number)
 (format nil "~{~A~^~}"
   (loop for i from 1 to number collect (format nil "x~a " i))))
```

———-

54.3.20 defun bcMakeEquations

[concat p1107]
[bcCreateVariableString p1325]
[nreverse0 p**??**]

— defun bcMakeEquations —

```
(defun |bcMakeEquations| (i number)
 (if (eql number 1)
   (concat (|bcCreateVariableString| 1) '|^2+1|))
   (progn
     (|bcCreateVariableString| i)
     (concat
      (concat (apply 'concat
       (let (t1)
        (do ((j 1 (1+ j))) ((> j number) (nreverse0 t1))
         (setq t1 (cons (concat (|bcCreateVariableString| j) '+) t1)))))
       '|1|)
      (concat '-2* (concat (|bcCreateVariableString| i) '|^2|))))))
```

———-

54.3.21 defun bcMakeLinearEquations

[bcCreateVariableString p1325]
[concat p1107]
[nreverse0 p??]

— **defun bcMakeLinearEquations** —

```
(defun |bcMakeLinearEquations| (i number)
 (cond
  ((eql number 1) (|bcCreateVariableString| 1))
  ((eql number 2)
    (cond
     ((eql i 1)
      (concat (|bcCreateVariableString| 1)
              (concat '+ (|bcCreateVariableString| 2))))
     (t
      (concat (|bcCreateVariableString| 1)
              (concat '- (|bcCreateVariableString| 2))))))
  (t
   (concat
    (concat
     (apply 'concat
      (let (t1)
       (do ((j 1 (1+ j))) ((> j number) (nreverse0 t1))
        (setq t1 (cons (concat (|bcCreateVariableString| j) '+) t1)))))
     '|1|)
    (concat '-2* (|bcCreateVariableString| i))))))
```

———-

54.3.22 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))))
```

———-

54.3.23 defun bcSolveEquationsNumerically

[htInitPage p1349]
[htMakePage p1351]
[htMakeDoneButton p1369]
[htShowPage p1350]

[htpPropertyList p1342]

— **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| PI))))
  (|htMakeDoneButton| "Continue" '|bcSolveNumerically1|)
  (|htShowPage|))
```

———-

54.3.24 defun bcSolveNumerically1

[bcSolveEquations p1327]

— **defun bcSolveNumerically1** —

```
(defun |bcSolveNumerically1| (htPage)
  (|bcSolveEquations| htPage '|numeric|))
```

———-

54.3.25 defun bcSolveEquations

[htpLabelInputString p1342]
[htpButtonValue p1340]
[member p1108]
[concat p1107]
[htpProperty p1342]
[bcString2WordList p1335]
[bcwords2liststring p1335]
[bcGenEquations p1333]
[bcFinish p1333]

— **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|))
       (concat "1.e-" digits)
       (concat "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)
     (|bcwords2liststring| varlist)
     (car varlist)))
  (setq eqnString (|bcGenEquations| r))
  (cond
   ((eq solutionMethod '|numeric|)
     (setq name (if (|member| kind '(|rf| |rr|)) "solve" "complexSolve"))
     (|bcFinish| name eqnString accString))
   (t
     (setq name (if (eq solutionMethod '|radical|) "radicalSolve" "solve"))
     (|bcFinish| name eqnString varString accString)))))
```

———-

54.3.26 defun Linear Solve Basic Command options

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Linear Solve Basic Command options

```
The right side vector B is:
     Zero:the system is homogeneous
     Not zero:the system is not homogeneous
```

— **defun bcLinearSolveMatrix1** —

```
(defun |bcLinearSolveMatrix1| (htPage)
 (let (page)
  (setq page
    (|htInitPage| "Linear Solve Basic Command" (|htpPropertyList| htPage)))
  (|htpSetProperty| page '|matrix| (|bcLinearExtractMatrix| htPage))
  (|htMakePage|
   '((|text| . "The right side vector B is:")
     (|lispLinks|
       ("Zero:" "the system is homogeneous" |bcLinearSolveMatrixHomo| |homo|)
       ("Not zero:" "the system is not homogeneous"
                  |bcLinearSolveMatrixInhomo| |nothomo|))))
  (|htShowPage|)))
```

———-

54.3.27 defun bcLinearExtractMatrix

[htpInputAreaAlist p1341]

— defun bcLinearExtractMatrix —

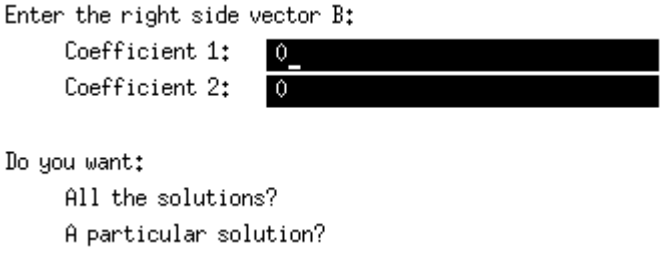
```
(defun |bcLinearExtractMatrix| (htPage)
  (reverse (|htpInputAreaAlist| htPage)))
```

———-

54.3.28 defun Linear Solve Basic Command Inhomogeneous

[concat p1107]
[htpProperty p1342]
[htInitPage p1349]
[htpPropertyList p1342]
[htpSetProperty p1342]
[htMakePage p1351]
[htShowPage p1350]
[$EmptyMode p629]

Linear Solve Basic Command Inhomogeneous



— defun bcLinearSolveMatrixInhomo —

```
(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 (concat "{\\em Coefficient " (princ-to-string i) ":}"))
    (unless (eql spacer 0)
       (setq prefix (concat prefix "\\space{" (princ-to-string spacer) "}")))
    (setq name (intern (concat "c" (princ-to-string 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|))
           ))))
  (|htShowPage|))))
```

———-

54.3.29 defun bcLinearSolveMatrixInhomoGen

[bcLinearMatrixGen p1331]

— **defun bcLinearSolveMatrixInhomoGen** —

```
(defun |bcLinearSolveMatrixInhomoGen| (htPage key)
  (|bcLinearMatrixGen| htPage key))
```

———-

54.3.30 defun bcLinearSolveMatrixHomo

[bcLinearMatrixGen p1331]

— **defun bcLinearSolveMatrixHomo** —

```
(defun |bcLinearSolveMatrixHomo| (htPage key)
 (declare (ignore key))
 (|bcLinearMatrixGen| htPage '|homo|))
```

———-

54.3.31 defun bcLinearMatrixGen

[bcMatrixGen p1316]
[bcFinish p1333]
[htpInputAreaAlist p1341]
[bcVectorGen p1336]
[bcMkFunction p1333]
[bcGen p1334]
[concat p1107]

— **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|)
                (concat form ".particular")
                form))))))
```

———-

54.3.32 defun linearFinalRequest

[sayBrightly p??]
[bcQueryInteger p??]
[explainLinear p1332]

— **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)))))
```

———-

54.3.33 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))))
```

———-

54.3.34 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)))))
```

———-

54.3.35 defun bcLinearSolveEqnsGen

[htpInputAreaAlist p1341]
[htpLabelInputString p1342]
[bcString2WordList p1335]
[bcwords2liststring p1335]
[bcGenEquations p1333]
[bcFinish p1333]

— **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)))
```

———-

54.3.36 defun bcGenEquations

[concat p1107]
[bcwords2liststring p1335]

— **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 (concat left " = " right) eqnlist)))
  (if (cdr eqnlist)
    (|bcwords2liststring| eqnlist)
    (car eqnlist))))
```

———-

54.3.37 defun Output the final formula

— **defun bcFinish** —

```
(defun |bcFinish| (&rest t1)
  (|bcGen| (|bcMkFunction| (car t1) (cadr t1) (cddr t1))))
```

———-

54.3.38 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)
 (let (str)
  (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 ")")))
```

———-

54.3.39 defun bcString2HyString2

— defun bcString2HyString2 —

```
(defun |bcString2HyString2| (s)
 (if (and (stringp s) (char= (elt s 0) #\"))
   (concatenate 'string "\\\"" s "\\\"")
   s))
```

———-

54.3.40 defun bcString2HyString

— defun bcString2HyString —

```
(defun |bcString2HyString| (s) s)
```

———-

54.3.41 defun find a character position in a string

— defun bcFindString —

```
(defun |bcFindString| (s i n char)
 (position char s :start i :end n))
```

———-

54.3.42 defun Basic Command result page – NAG version

[concat p1107]
[htInitPage p1349]
[htMakePage p1351]

[htShowPage p1350]
Except for the banner the `bcGen` and `linkGen` functions are identical. We no longer care so we just call `bcGen`.

— **defun linkGen** —

```
(defun |linkGen| (command)
 (|bcGen| command))
```

————-

54.3.43 defun bcOptional

— **defun bcOptional** —

```
(defun |bcOptional| (s)
 (if (string-equal s "") "2" s))
```

————-

54.3.44 defun create a vertical space on a page

[bcHt p1347]

— **defun bcvspace** —

```
(defun |bcvspace| ()
 (|bcHt| "\\vspace{1}\\newline "))
```

————-

54.3.45 defun break a string into words

— **defun bcString2WordList** —

```
(defun |bcString2WordList| (string)
  (loop for i = 0 then (1+ j)
        as j = (position #\space string :start i)
        collect (subseq string i j)
        while j))
```

————-

54.3.46 defun format words into a string

[concat p1107]

— **defun bcwords2liststring** —

```
(defun |bcwords2liststring| (words)
 (format nil "[~{~A~^, ~}]" words))
```

———-

54.3.47 defun format a vector

[bcwords2liststring p1335]

— **defun bcVectorGen** —

```
(defun |bcVectorGen| (vec)
 (|bcwords2liststring| vec))
```

———-

54.3.48 defun format an error message

[sayBrightlyNT p??]
[sayBrightly p??]

— **defun bcError** —

```
(defun |bcError| (string)
  (|sayBrightlyNT| "NOTE: ")
  (|sayBrightly| string))
```

———-

54.3.49 defun format intervals

[concat p1107]

— **defun bcDrawIt** —

```
(defun |bcDrawIt| (ind a b)
  (concat ind "=" a ".." b))
```

———-

54.3.50 defun Basic Command page not ready

[htInitPage p1349]
[htMakePage p1351]
[htShowPage p1350]

— **defun bcNotReady** —

```
(defun |bcNotReady| (htPage)
 (declare (ignore htPage))
```

```
 (|htInitPage| "Basic Command" NIL)
 (|htMakePage|
  '((|text| . "{\\centerline{\\em This facility will soon be available}}")))
 (|htShowPage|))
```

———-

54.3.51 defun pad a string with blanks

[concat p1107]

— **defun htStringPad** —

```
(defun |htStringPad| (n w)
 (let (s ws)
  (setq s (princ-to-string n))
  (setq ws (|#| s))
  (concat "\\space{" (princ-to-string (1+ (- w ws))) "}" s)))
```

———-

54.3.52 defun construct a name string

Given `("one" "two" "three")` generate `"(one,two,three)"`

— **defun stringList2String** —

```
(defun |stringList2String| (x)
 (let (str)
 (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 ")")))))
```

———-

54.3.53 defun construct a name string

[concat p1107]

— **defun htMkName** —

```
(defun |htMkName| (s n)
 (concat s (princ-to-string n)))
```

———-

;;; ht-util merge

54.3.54 defvar $bcParseOnly

— **initvars** —

```
(defvar |$bcParseOnly| t)
```

———-

54.3.55 defvar $htLineList

— **initvars** —

```
(defvar |$htLineList| nil)
```

———-

54.3.56 defvar $curpage

— **initvars** —

```
(defvar |$curPage| nil)
```

———-

54.3.57 HTPage Layout

This is a list with the fields

1. name
2. Domain Conditions
3. Domain Variable Alist
4. Domain Pvar Subst List
5. Radio Button Alist
6. Input Area Alist
7. Property List
8. Description

54.3.58 defun htpName

— **defun htpName** —

```
(defun |htpName| (htPage) (elt htPage 0))
```

———-

54.3.59 defun htpSetName

— **defun htpSetName** —

```
(defun |htpSetName| (htPage val) (setf (elt htPage 0) val))
```

———-

54.3.60 defun htpDomainConditions

— **defun htpDomainConditions** —

```
(defun |htpDomainConditions| (htPage) (elt htPage 1))
```

———-

54.3.61 defun htpSetDomainConditions

— **defun htpSetDomainConditions** —

```
(defun |htpSetDomainConditions| (htPage val)
  (setf (elt htPage 1) val))
```

———-

54.3.62 defun htpDomainVariableAlist

— **defun htpDomainVariableAlist** —

```
(defun |htpDomainVariableAlist| (htPage) (elt htPage 2))
```

———-

54.3.63 defun htpSetDomainVariableAlist

— defun htpSetDomainVariableAlist —

```
(defun |htpSetDomainVariableAlist| (htPage val)
  (setf (elt htPage 2) val))
```

———-

54.3.64 defun htpDomainPvarSubstList

— defun htpDomainPvarSubstList —

```
(defun |htpDomainPvarSubstList| (htPage) (elt htPage 3))
```

———-

54.3.65 defun htpSetDomainPvarSubstList

— defun htpSetDomainPvarSubstList —

```
(defun |htpSetDomainPvarSubstList| (htPage val)
  (setf (elt htPage 3) val))
```

———-

54.3.66 defun htpRadioButtonAlist

— defun htpRadioButtonAlist —

```
(defun |htpRadioButtonAlist| (htPage) (elt htPage 4))
```

———-

54.3.67 defun htpButtonValue

— defun htpButtonValue —

```
(defun |htpButtonValue| (htPage groupName)
  (prog ()
    (return
      (SEQ (DO ((G166092
                   (LASSOC groupName
                           (|htpRadioButtonAlist| htPage))
                   (CDR G166092))
                (|buttonName| nil))
```

```
        ((OR (ATOM G166092)
             (progn (setq |buttonName| (car G166092)) nil))
         NIL)
        (SEQ (EXIT (COND
                    ((BOOT-EQUAL
                         (|stripSpaces|
                             (|htpLabelInputString| htPage
                              |buttonName|))
                         "t")
                     (EXIT (RETURN |buttonName|)))))))))
```

———-

54.3.68 defun htpSetRadioButtonAlist

— defun htpSetRadioButtonAlist —

```
(defun |htpSetRadioButtonAlist| (htPage val)
  (setf (elt htPage 4) val))
```

———-

54.3.69 defun htpInputAreaAlist

— defun htpInputAreaAlist —

```
(defun |htpInputAreaAlist| (htPage)
 (elt htPage 5))
```

———-

54.3.70 defun htpSetInputAreaAlist

— defun htpSetInputAreaAlist —

```
(defun |htpSetInputAreaAlist| (htPage val)
  (setf (elt htPage 5) val))
```

———-

54.3.71 defun htpAddInputAreaProp

— defun htpAddInputAreaProp —

```
(defun |htpAddInputAreaProp| (htPage label prop)
  (setf (elt htPage 5)
```

```
    (cons
      (cons label (cons nil (cons nil (cons nil prop))))
      (elt htPage 5))))
```

———-

54.3.72 defun htpPropertyList

— defun htpPropertyList —

```
(defun |htpPropertyList| (htPage)
 (elt htPage 6))
```

———-

54.3.73 defun htpProperty

— defun htpProperty —

```
(defun |htpProperty| (htPage propName)
  (lassoc propName (elt htPage 6)))
```

———-

54.3.74 defun htpSetProperty

— defun htpSetProperty —

```
(defun |htpSetProperty| (htPage propName val)
 (let (pair)
  (setq pair (|assoc| propName (elt htPage 6)))
  (cond
   (pair (rplacd pair val))
   (t (setf (elt htPage 6) (cons (cons propName val) (elt htPage 6)))))))
```

———-

54.3.75 defun htpLabelInputString

— defun htpLabelInputString —

```
(defun |htpLabelInputString| (htPage label)
 (let (props s)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when (and props (stringp (setq s (elt props 0))))
    (if (equal s "") s (|trimString| s)))))
```

———-

54.3.76 defun htpLabelFilteredInputString

— defun htpLabelFilteredInputString —

```
(defun |htpLabelFilteredInputString| (htPage label)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props
    (cond
     ((and (> (|#| props) 5) (elt props 6))
       (funcall (symbol-function (elt props 6)) (elt props 0)))
     (t (|replacePercentByDollar| (elt props 0)))))))
```

———-

54.3.77 defun replacePercentByDollar,fn

— defun replacePercentByDollar,fn —

```
(defun |replacePercentByDollar,fn| (s i n)
 (let (m)
  (cond
   ((> i n) "")
   ((> (setq m (|charPosition| #\% s i)) n) (substring s i nil))
   (t (concat (substring s i (- m i)) "$"
       (|replacePercentByDollar,fn| s (1+ m) n))))))
```

———-

54.3.78 defun replacePercentByDollar

— defun replacePercentByDollar —

```
(defun |replacePercentByDollar| (s)
 (|replacePercentByDollar,fn| s 0 (maxindex s)))
```

———-

54.3.79 defun htpSetLabelInputString

— defun htpSetLabelInputString —

```
(defun |htpSetLabelInputString| (htPage label val)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
```

```
  (when props (setf (elt props 0) (princ-to-string val)))))
```

———-

54.3.80 defun htpLabelSpadValue

— defun htpLabelSpadValue —

```
(defun |htpLabelSpadValue| (htPage label)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (elt props 1))))
```

———-

54.3.81 defun htpSetLabelSpadValue

— defun htpSetLabelSpadValue —

```
(defun |htpSetLabelSpadValue| (htPage label val)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (setf (elt props 1) val))))
```

———-

54.3.82 defun htpLabelErrorMsg

— defun htpLabelErrorMsg —

```
(defun |htpLabelErrorMsg| (htPage label)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (elt props 2))))
```

———-

54.3.83 defun htpSetLabelErrorMsg

— defun htpSetLabelErrorMsg —

```
(defun |htpSetLabelErrorMsg| (htPage label val)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (setf (elt props 2) val))))
```

———-

54.3.84 defun htpLabelType

— defun htpLabelType —

```
(defun |htpLabelType| (htPage label)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (elt props 3))))
```

———-

54.3.85 defun htpLabelDefault

— defun htpLabelDefault —

```
(defun |htpLabelDefault| (htPage label)
 (let (msg props)
  (cond
   ((setq msg (|htpLabelInputString| htPage label))
      (cond
       ((equal msg "t") 1)
       ((equal msg "nil") 0)
       (t msg)))
   (t
    (setq props (lassoc label (|htpInputAreaAlist| htPage)))
    (when props (elt props 4))))))
```

———-

54.3.86 defun htpLabelSpadType

— defun htpLabelSpadType —

```
(defun |htpLabelSpadType| (htPage label)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (elt props 5))))
```

———-

54.3.87 defun htpLabelFilter

— defun htpLabelFilter —

```
(defun |htpLabelFilter| (htPage label)
 (let (props)
  (setq props (lassoc label (|htpInputAreaAlist| htPage)))
  (when props (elt props 6))))
```

———-

54.3.88 defun htpPageDescription

— defun htpPageDescription —

```
(defun |htpPageDescription| (htPage)
 (elt htPage 7))
```

———-

54.3.89 defun htpSetPageDescription

— defun htpSetPageDescription —

```
(defun |htpSetPageDescription| (htPage pageDescription)
  (setf (elt htPage 7) pageDescription))
```

———-

54.3.90 defun htpAddToPageDescription

— defun htpAddToPageDescription 0 —

```
(defun |htpAddToPageDescription| (htPage pageDescrip)
 (let (newDescript)
  (setq newDescript
   (if (stringp pageDescrip)
    (cons pageDescrip (elt htPage 7))
    (nconc (nreverse (copy-list pageDescrip)) (elt htPage 7))))
  (setf (elt htPage 7) newDescript)))
```

———-

54.3.91 defun issue a single hypertex line or group of lines

[mapStringize p1348]
[basicStringize p1348]
[$htLineList p1338]
[$newPage p1253]

— **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|)))))
```

———-

54.3.92 defun bcHt

[htpAddToPageDescription p1346]
[mapStringize p1348]
[basicStringize p1348]
[$newPage p1253]
[$htLineList p1338]
[$curPage p??]

— **defun bcHt** —

```
(defun |bcHt| (line)
 (let (text)
  (declare (special |$newPage| |$htLineList| |$curPage|))
   (cond
     (|$newPage|
      (setq text
       (cond
        ((consp line) (list (cons '|text| line)))
        ((stringp line) line)
        (t (list (cons '|text| (list line))))))
      (|htpAddToPageDescription| |$curPage| text))
     ((consp line)
       (setq |$htLineList|
        (nconc (nreverse (|mapStringize| (copy-list line))) |$htLineList|)))
     (t
      (setq |$htLineList| (cons (|basicStringize| line) |$htLineList|))))))
```

———-

54.3.93 defun htSay

[bcHt p1347]

— **defun htSay** —

```
(defun |htSay| (&rest args)
```

```
  (|bcHt| (car args))
  (loop for y in (cdr args) do (|bcHt| y)))
```

———-

54.3.94 defun bcIssueHt

— **defun bcIssueHt** —

```
(defun |bcIssueHt| (line)
  (cond ((consp line) (|htMakePage1| line)) (t (|iht| line))))
```

———-

54.3.95 defun mapStringize

[basicStringize p1348]
[mapStringize p1348]

— **defun mapStringize** —

```
(defun |mapStringize| (z)
  (cond
    ((atom z) z)
    (t (rplaca z (|basicStringize| (car z)))
       (rplacd z (|mapStringize| (cdr z))) z)))
```

———-

54.3.96 defun basicStringize

— **defun basicStringize 0** —

```
(defun |basicStringize| (s)
  (cond
    ((stringp s)
     (cond
       ((equal s "\\$") "\\%")
       ((equal s "{\\em $}") "{\\em \\%}")
       (t s)))
    ((eq s '$) "\\%")
    (t (princ-to-string s))))
```

———-

54.3.97 defun stringize

— defun stringize 0 —

```
(defun |stringize| (s)
  (if (stringp s) s (princ-to-string s)))
```

————-

54.3.98 defun htInitPage

This function creates a page with the given title. For example, a call might be:

```
(|htInitePage| |Differentiate Basic Command| nil)
```

which creates a blank page with the given title.

[htInitPageNoScroll p1349]
[htSayStandard p1350]
[$curPage p??]

— defun htInitPage —

```
(defun |htInitPage| (title propList)
 (declare (special |$curPage|))
  (|htInitPageNoScroll| propList title)
  (|htSayStandard| "\\beginscroll ")
  |$curPage|)
```

————-

54.3.99 defun htInitPageNoScroll

[htpMakeEmptyPage p1311]
[htpName p1339]
[htSayStandard p1350]
[htSay p1347]
[$atLeastOneUnexposed p??]
[$curPage p??]
[$newPage p1253]
[$htLineList p1338]

— defun htInitPageNoScroll —

```
(defun |htInitPageNoScroll| (&rest args)
 (let (title (propList (car args)) (options (cdr args)))
 (declare (special |$atLeastOneUnexposed| |$curPage| |$newPage| |$htLineList|))
  ; reset every time a new page is initialized
  (setq |$atLeastOneUnexposed| nil)
  (setq title (car options))
  (setq |$curPage| (|htpMakeEmptyPage| propList))
  (setq |$newPage| t)
```

```
  (setq |$htLineList| nil)
  (when title
    (|htSayStandard| (list "\\begin{page}{" (|htpName| |$curPage|) "}{" ))
    (|htSay| title)
    (|htSayStandard| "} "))
  |$curPage|))
```

———-

54.3.100 defun htSayStandard

[htSayBind p1350]

— **defun htSayStandard 0** —

```
(defun |htSayStandard| (&rest args)
 (|htSayBind| (car args) (cdr args)))
```

———-

54.3.101 defun htSayBind

[bcHt p1347]

— **defun htSayBind** —

```
(defun |htSayBind| (x options)
 (|bcHt| x)
 (loop for y in options do (|bcHt| y)))
```

———-

54.3.102 defun htAddHeading

— **defun htAddHeading** —

```
(defun |htAddHeading| (title)
 (declare (special |$curPage|))
 (|htNewPage| title)
 |$curPage|)
```

———-

54.3.103 defun htShowPage

— **defun htShowPage** —

```
(defun |htShowPage| ()
```

```
  (|htSayStandard| "\\endscroll")
  (|htShowPageNoScroll|))
```

———-

54.3.104 defun show the page which has been computed

Until this point in page processing the page has been represented by a list of pages elements in reverse order. This function reverses the list so they appear in page display order.

— **defun htShowPageNoScroll** —

```
(defun |htShowPageNoScroll| ()
 (let (line)
 (declare (special |$htLineList| |$curPage| |$newPage|))
   (|htSayStandard| "\\autobuttons")
   (|htpSetPageDescription| |$curPage|
     (nreverse (|htpPageDescription| |$curPage|)))
   (setq |$newPage| nil)
   (setq |$htLineList| nil)
   (|htMakePage| (|htpPageDescription| |$curPage|))
   (setq line (apply #'concat (nreverse |$htLineList|)))
   (|issueHT| line)
   (|endHTPage|)))
```

———-

54.3.105 defun make a page given the description in itemList

— **defun htMakePage** —

```
(defun |htMakePage| (itemList)
  (declare (special |$curPage| |$newPage|))
  (progn
    (cond
      (|$newPage| (|htpAddToPageDescription| |$curPage| itemList)))
    (|htMakePage1| itemList)))
```

———-

54.3.106 defun htMakePage1

Make a page given the description in itemList

— **defun htMakePage1** —

```
(defun |htMakePage1| (itemList)
 (let (itemType items)
  (loop for u in itemList do
   (setq itemType '|text|)
   (setq items
    (cond
```

```
    ((stringp u) u)
    ((atom u) (princ-to-string u))
    ((stringp (car u)) u)
    ((and (consp u) (eq (car u) '|text|)) (cdr u))
    (t
     (setq itemType (car u)) ; look up the tag for the next description
     (cdr u))))
  (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 (|systemError| (list "unknown itemType" itemType)))))))
```

———-

54.3.107 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| (|htpPageDescription| htPage))
        (setq line (apply #'CONCAT (NREVERSE |$htLineList|)))
        (|issueHT| line)
        (|endHTPage|)))))
```

———-

54.3.108 defun htQuote

— **defun htQuote** —

```
(defun |htQuote| (s)
  (|iht| "\"")
  (|iht| s)
  (|iht| "\""))
```

———-

54.3.109 defun htProcessToggleButtons

— defun htProcessToggleButtons —

```
(defun |htProcessToggleButtons| (buttons)
  (prog (message info defaultValue buttonName)
  (declare (special |$curPage|))
    (return
      (SEQ (progn
             (|iht| "\\newline\\indent{5}\\beginitems ")
             (DO ((G166302 buttons (CDR G166302))
                  (G166286 nil))
                 ((OR (ATOM G166302)
                      (progn (setq G166286 (car G166302)) nil)
                      (progn
                        (progn
                          (setq message (car G166286))
                          (setq info (CADR G166286))
                          (setq defaultValue (CADDR G166286))
                          (setq buttonName (CADDDR G166286))
                          G166286)
                        nil))
                  nil)
               (SEQ (EXIT (progn
                            (cond
                              ((NULL (LASSOC buttonName
                                      (|htpInputAreaAlist| |$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} "))))))
```

———-

54.3.110 defun htProcessBcButtons

— defun htProcessBcButtons —

```
(defun |htProcessBcButtons| (buttons)
  (prog (defaultValue buttonName k)
  (declare (special |$curPage|))
    (return
      (SEQ (DO ((G166328 buttons (CDR G166328)) (G166317 nil))
               ((OR (ATOM G166328)
                    (progn (setq G166317 (car G166328)) nil)
                    (progn
                      (progn
                        (setq defaultValue (car G166317))
                        (setq buttonName (CADR G166317))
                        G166317)
                       nil))
                nil)
           (SEQ (EXIT (progn
                        (cond
                          ((NULL (LASSOC buttonName
                                  (|htpInputAreaAlist| |$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
                                      "}{\\htbmfile{pick}}{\\htbmfile{unpick}}"
                                          nil)))))))))))))))
```

———-

54.3.111 defun htProcessBcStrings

— defun htProcessBcStrings —

```
(defun |htProcessBcStrings| (strings)
  (PROG (numChars default stringName spadType filter mess2)
  (declare (special |$curPage|))
    (return
      (SEQ (DO ((g2 strings (CDR g2)) (G166343 nil))
               ((or (atom g2)
                    (progn (setq G166343 (CAR g2)) nil)
                    (progn
                      (progn
                        (setq numChars (car G166343))
                        (setq default (cadr G166343))
                        (setq stringName (caddr G166343))
                        (setq spadType (cadddr G166343))
                        (setq filter (cddddr G166343))
                        G166343)
                      nil))
                nil)
           (SEQ (EXIT (progn
                        (setq mess2 "")
                        (cond
                          ((null (LASSOC stringName
                                  (|htpInputAreaAlist| |$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| |$curPage|
                               stringName nil)))
                        (|iht| (cons "\\inputstring{"
                                     (cons stringName
                                      (cons "}{"
                                       (cons numChars
                                        (cons "}{"
                                         (cons
                                          (|htpLabelDefault|
                                           |$curPage| stringName)
                                          (cons "} "
```

```
                                        (cons mess2 nil)))))))))))))))))
```

———-

54.3.112 defun bcSadFaces

— defun bcSadFaces —

```
(defun |bcSadFaces| ()
       "\\space{1}{\\em\\htbitmap{error}\\htbitmap{error}\\htbitmap{error}}")
```

———-

54.3.113 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
                        (progn
                          (setq message (car G166387))
                          (setq info (cadr G166387))
                          (setq func (caddr G166387))
                          (setq value (cdddr G166387))
                          G166387)
                        nil))
                  nil)
               (SEQ (EXIT (progn
                            (|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} ")))))
```

———-

54.3.114 defun htLispMemoLinks

— defun htLispMemoLinks —

```
(defun |htLispMemoLinks| (links) (|htLispLinks| links t))
```

———-

54.3.115 defun htBcLinks

— defun htBcLinks —

```
(defun |htBcLinks| (&rest a1)
 (let (skipStateInfo? t1 message info func value options links)
   (setq links (car a1))
   (setq options (cdr a1))
   (setq skipStateInfo? (ifcar options))
   (setq t1 (|beforeAfter| '|options| links))
   (setq links (car t1))
   (setq options (cadr t1))
   (do ((g1 links (CDR g1)) (g2 nil))
       ((or (atom g1)
            (progn (setq g2 (car g1)) nil)
            (progn
              (progn
                (setq message (car g2))
                (setq info (cadr g2))
                (setq func (caddr g2))
                (setq value (cdddr g2))
                g2)
               nil))
           nil)
     (|htMakeButton| "\\lispdownlink" message
        (|mkCurryFun| func value)  skipStateInfo?)
     (|bcIssueHt| info))))
```

———-

54.3.116 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
                        (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)))))))))
```

———-

54.3.117 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))))))
```

———-

54.3.118 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
                                     (cons (mkq fun)
                                      (cons
                                       (cons 'cons
                                        (cons 'arg
                                         (cons (mkq val) nil)))
                                       nil)))
                                    nil)))))
        (eval code)
        name))))
```

———-

54.3.119 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
```

```
                "}{\\htbmfile{pick}}{\\htbmfile{unpick}}\\beginitems "
                          nil))))
            (setq defaultValue "1")
            (DO ((G166568 buttons (cdr G166568))
                 (G166540 nil))
                ((or (atom G166568)
                     (progn (setq G166540 (car G166568)) nil)
                     (progn
                       (progn
                         (setq message (car G166540))
                         (setq info (cadr G166540))
                         (setq buttonName (caddr G166540))
                         G166540)
                        nil))
                 nil)
              (SEQ (EXIT (progn
                           (cond
                             ((null (LASSOC buttonName
                                    (|htpInputAreaAlist| |$curPage|)))
                              (|setUpDefault| buttonName
                                  (cons '|button|
                                   (cons defaultValue nil)))
                              (setq defaultValue
                                      "0")))
                           (|iht| (cons "\\item{\\em\\radiobox["
                                    (cons
                                     (|htpLabelDefault| |$curPage|
                                      buttonName)
                                     (cons "]{"
                                      (cons buttonName
                                       (cons "}{"
                                        (cons boxesName
                                         (cons
                                          "}\\space{}"
                                          nil))))))))
                           (|bcIssueHt| message)
                           (|iht| "\\space{}}")
                           (|bcIssueHt| info)))))
            (|iht| "\\enditems\\indent{0} ")))))
```

———-

54.3.120 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)))))))
```

———-

54.3.121 defun setUpDefault

— **defun setUpDefault** —

```
(defun |setUpDefault| (name props)
```

```
  (declare (special |$curPage|))
  (|htpAddInputAreaProp| |$curPage| name props))
```

———-

54.3.122 defun buttonNames

— defun buttonNames —

```
(defun |buttonNames| (buttons)
  (prog (buttonName)
    (return
      (SEQ (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))
                    (NREVERSE0 G166645))
                 (SEQ (EXIT (setq G166645
                                  (cons buttonName G166645))))))))))
```

———-

54.3.123 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
                            (car (cddddr G166665)))
                    (setq spadType
                            (cadr (cddddr G166665)))
                    (setq filter (cddr (cddddr G166665)))
                    G166665)
                  nil))
          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 "))))))
```

———-

54.3.124 defun htProcessDomainConditions

— defun htProcessDomainConditions —

```
(defun |htProcessDomainConditions| (condList)
 (declare (special |$curPage|))
  (|htpSetDomainConditions| |$curPage| (|renamePatternVariables| condList))
```

```
  (|htpSetDomainVariableAlist| |$curPage| (|computeDomainVariableAlist|)))
```

———-

54.3.125 defun renamePatternVariables

— defun renamePatternVariables —

```
(defun |renamePatternVariables| (condList)
  (declare (special |$curPage| |$PatternVariableList|))
  (progn
    (|htpSetDomainPvarSubstList| |$curPage|
        (|renamePatternVariables1| condList nil
            |$PatternVariableList|))
    (|substFromAlist| condList (|htpDomainPvarSubstList| |$curPage|))))
```

———-

54.3.126 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)))))))
```

———-

54.3.127 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
                        (progn
                          (setq pvar (car G166783))
                          (setq replace (cdr G166783))
                          G166783)
                        nil))
                  nil)
               (SEQ (EXIT (setq z (subst replace pvar z :test #'equal)))))
             z)))))
```

———-

54.3.128 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
                          (progn
                            (setq pvar (cdr G166805))
                            G166805)
                           NIL))
                    (NREVERSE0 G166813))
                 (SEQ (EXIT (setq G166813
                                  (cons (cons pvar
                                         (|pvarCondList| pvar))
                                        G166813)))))))))))
```

———-

54.3.129 defun pvarCondList

— defun pvarCondList —

```
(defun |pvarCondList| (pvar)
  (declare (special |$curPage|))
  (NREVERSE
      (|pvarCondList1| (cons pvar nil) nil
          (|htpDomainConditions| |$curPage|))))
```

———-

54.3.130 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)))))
                 (|member| pv pvarList))
            (|pvarCondList1|
                (NCONC pvarList (|pvarsOfPattern| pattern))
                (cons cond activeConds) restConds))
           (t (|pvarCondList1| pvarList activeConds restConds))))))))
```

———-

54.3.131 defun pvarsOfPattern

— 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))))))))))))))))
```

———-

54.3.132 defun htMakeTemplates,substLabel

— defun htMakeTemplates,substLabel —

```
(defun |htMakeTemplates,substLabel| (i template)
  (SEQ (if (consp template)
           (EXIT (intern (CONCAT (car template) (princ-to-string i)
                                 (cdr template)))))
       (EXIT template)))
```

———-

54.3.133 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
                                          (car G166927))
                                         nil))
                                      (NREVERSE0 G166922))
                                      (SEQ
                                       (EXIT
                                        (setq G166922
                                         (cons
                                          (|htMakeTemplates,substLabel|
                                           i template)
                                          G166922)))))))
                              G166910))))))))))))
```

———-

54.3.134 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)))))))))))
```

———-

54.3.135 defun htMakeDoneButton

— defun htMakeDoneButton —

```
(defun |htMakeDoneButton| (message func)
  (progn
    (|bcHt| "\\newline\\vspace{1}\\centerline{")
    (cond
      ((equal message "Continue")
       (|bchtMakeButton| "\\lispdownlink"
           '|\\ContinueBitmap| func))
```

```
      (t
       (|bchtMakeButton| "\\lispdownlink"
           (CONCAT "\\box{" message "}")
           func)))
    (|bcHt| "} ")))
```

———-

54.3.136 defun htProcessDoneButton

— defun htProcessDoneButton —

```
(defun |htProcessDoneButton| (arg)
 (let (label func)
  (setq label (car arg))
  (setq func (cadr arg))
  (|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| "} ")))
```

———-

54.3.137 defun htMakeButton

— defun htMakeButton —

```
(defun |htMakeButton| (&rest arg)
 (let (options func message htCommand)
  (setq htCommand (car arg))
  (setq message (cadr arg))
  (setq func (caddr arg))
  (setq options (cdddr arg))
  (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
                          (progn
                            (setq id (car G166965))
                            (setq type (car (cddddr G166965)))
                            G166965)
                          nil))
                    nil)
                 (SEQ (EXIT (progn
                              (|iht| (cons "(|htpSetLabelInputString| "
                                           (cons (|htpName| |$curPage|)
                                            (cons "'|"
                                             (cons id
                                              (cons "| " nil))))))
                              (cond
                                ((eq type '|string|)
                                 (|iht| (cons "\"\\stringvalue{"
                                              (cons id
                                               (cons "}\""
                                                nil)))))
                                (t
                                 (|iht| (cons
                                         "\"\\boxvalue{"
                                         (cons id
                                          (cons "}\""
                                           nil))))))
                              (|iht| ") ")))))
               (|iht| (cons (|htpName| |$curPage|)
                            (cons "))}" nil)))))))))))
```

———-

54.3.138 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| |$curPage|)
                            (cdr G167004))
                   (G166992 nil))
                  ((or (atom G167004)
                       (progn (setq G166992 (car G167004)) nil)
                       (progn
                         (progn
                           (setq id (car G166992))
                           (setq type (car (cddddr G166992)))
                           G166992)
                          nil))
                   nil)
                (SEQ (EXIT (progn
                             (|bcHt| (cons "(|htpSetLabelInputString| "
                                      (cons (|htpName| |$curPage|)
                                       (cons "'|"
                                        (cons id
                                         (cons "| " nil))))))
                             (cond
                               ((eq type '|string|)
                                (|bcHt| (cons
                                         "\"\\stringvalue{"
                                         (cons id
                                          (cons "}\"" nil)))))
                               (t
                                (|bcHt| (cons
                                         "\"\\boxvalue{"
                                         (cons id
                                          (cons "}\"" nil))))))
                             (|bcHt| ") "))))))
              (|bcHt| (cons (|htpName| |$curPage|)
                            (cons "))} " nil))))))))
```

———-

54.3.139 defun htProcessDoitButton

— defun htProcessDoitButton —

```
(defun |htProcessDoitButton| (arg)
 (let (label command func fun)
  (setq label (car arg))
  (setq command (cadr arg))
  (setq func (caddr arg))
  (setq fun (|mkCurryFun| func (cons command nil)))
  (|iht| "\\newline\\vspace{1}\\centerline{")
```

```
  (|htMakeButton| "\\lispcommand" (concat "\\box{" label "}") fun)
  (|iht| "} ")
  (|iht| "\\vspace{2}{Select \\  \\UpButton{} \\  to go back one page.}")
  (|iht|
   "\\newline{Select \\  \\ExitButton{QuitPage} \\  to remove this window.}")))
```

———-

54.3.140 defun htDoneButton

— defun htDoneButton —

```
(defun |htDoneButton| (func htPage)
  (cond
    ((|typeCheckInputAreas| htPage) (|htMakeErrorPage| htPage))
    ((null (fboundp func))
     (|systemError| (cons "unknown function" (cons func nil))))
    (t (funcall (symbol-function func) htPage))))
```

———-

54.3.141 defun typeCheckInputAreas

— 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))
                    (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|
               htPage))
             (|htpDomainVariableAlist|
              htPage)))
  (setq string
            (|htpLabelFilteredInputString|
             htPage stringName))
  (cond
    (|$bcParseOnly|
        (cond
          ((null
```

```
                                        (|ncParseFromString| string))
                                       (|htpSetLabelErrorMsg| htPage
                                        "Syntax Error"
                                        "Syntax Error"))
                                      (t nil)))
                                (t
                                 (setq val
                                          (|checkCondition|
                                           (|htpLabelInputString|
                                            htPage stringName)
                                           string condList))
                                 (cond
                                   ((stringp val)
                                    (setq errorCondition t)
                                    (|htpSetLabelErrorMsg| htPage
                                     stringName val))
                                   (t
                                    (|htpSetLabelSpadValue| htPage
                                     stringName val)))))))))
          errorCondition)))))
```

———-

54.3.142 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))
                    (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)))))))))

          ————-
```

54.3.143 defun condErrorMsg

— defun condErrorMsg —

```
(defun |condErrorMsg| (type)
  (prog (typeString)
    (return
      (progn
        (setq typeString (|form2String| type))
        (cond
          ((consp typeString)
           (setq typeString
                    (apply #'CONCAT typeString))))
```

```
        (CONCAT "Error: Could not make your input into a "
                typeString)))))
```

———-

54.3.144 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)))))
```

———-

54.3.145 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 ")
   (pform
    (setq val
     (|applyWithOutputToString| '|processInteractive|
      (cons pform (list nil))))
    (cond
     ((car val) (car val))
     (t "Type Analysis Error")))
   (t nil))))
```

———-

54.3.146 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)))))))
```

———-

54.3.147 defun makeSpadCommand

— defun makeSpadCommand —

```
(defun |makeSpadCommand| (&rest a1)
 (let (opForm lastArg argList z)
  (setq z a1)
  (setq opForm (concat (car z) "("))
  (setq lastArg (|last| z))
  (setq z (cdr z))
  (setq argList nil)
  (do ((g1 z (cdr g1)) (arg nil))
       ((or (atom g1)
            (progn (setq arg (car g1)) nil)
            (null (nequal arg lastArg)))
            nil)
    (setq argList (cons (concat arg ", ") argList)))
    (setq argList (nreverse (cons lastArg argList)))
    (concat opForm (apply #'concat argList) ")")))
```

———-

54.3.148 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)))
(|bracketString| (apply #'CONCAT argList)))))))
```

———-

54.3.149 defun bracketString

— defun bracketString —

```
(defun |bracketString| (string)
  (CONCAT "[" string "]"))
```

———-

54.3.150 defun quoteString

— defun quoteString —

```
(defun |quoteString| (string)
  (concat "\"" string "\""))
```

———-

54.3.151 defvar $funnyQuote

— initvars —

```
(defvar |$funnyQuote| #\Rubout)
```

———-

54.3.152 defvar $funnyBacks

— initvars —

```
(defvar |$funnyBacks| #\\200)
```

———-

54.3.153 defun htEscapeString

— defun htEscapeString —

```
(defun |htEscapeString| (str)
 (declare (special |$funnyBacks| |$funnyQuote|))
  (setq str (substitute |$funnyQuote| #\" str))
  (substitute |$funnyBacks| #\\ str))
```

———-

54.3.154 defun htsv

— defun htsv —

```
(defun |htsv| ()
 (|startHTPage| 50)
 (|htSetVars|))
```

———-

54.3.155 defun htSetVars

— defun htSetVars —

```
(defun |htSetVars| ()
 (declare (special |$setOptions| |$lastTree| |$path|))
  (setq |$path| nil)
  (setq |$lastTree| nil)
  (when (nequal 0 (lastatom |$setOptions|)) (|htMarkTree| |$setOptions| 0))
  (|htShowSetTree| |$setOptions|))
```

———-

54.3.156 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 (- (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|
                                (PRINC-TO-STRING
                                 (elt setData 1)))
                               maxWidth2)))))))))
     (setq maxWidth1 (max 9 maxWidth1))
     (setq maxWidth2 (max 41 maxWidth2))
     (setq tabset1 (PRINC-TO-STRING maxWidth1))
     (setq tabset2
              (PRINC-TO-STRING
                  (-
                      (+ maxWidth2 maxWidth1) 1)))
     (|htSay| "\\tab{2}\\newline Variable\\tab{"
              (PRINC-TO-STRING
                  (+ maxWidth1
                        (quotient maxWidth2 3)))
              "}Description\\tab{"
              (PRINC-TO-STRING
                  (+ (+ maxWidth2 maxWidth1) 2))
              "}Value\\newline\\beginitems ")
     (DO ((G167392 (reverse okList) (CDR G167392))
          (setData nil))
         ((or (atom G167392)
              (progn
                (setq setData (car G167392))
                nil))
          nil)
       (SEQ (EXIT (progn
                    (|htSay| "\\item")
                    (setq label
```

```
                    (CONCAT "\\menuitemstyle{"
                     (elt setData 0)
                     "}"))
                (setq links
                    (cons label
                     (cons
                      (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|)))))))
```

———-

54.3.157 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)))))))))
```

———-

54.3.158 defun htShowSetTreeValue

— **defun htShowSetTreeValue** —

```
(defun |htShowSetTreeValue| (setData)
 (let (st)
  (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|)))))
```

———-

54.3.159 defun mkSetTitle

— **defun mkSetTitle** —

```
(defun |mkSetTitle| ()
 (declare (special |$path|))
 (concat "Command {\\em )set " (|listOfStrings2String| |$path|) "}"))
```

———-

54.3.160 defun listOfStrings2String

— **defun listOfStrings2String** —

```
(defun |listOfStrings2String| (u)
 (cond
```

```
  ((null u) "")
  (t (concat (|listOfStrings2String| (cdr u)) " " (|stringize| (car u))))))
```

———-

54.3.161 defun htShowSetPage

— defun htShowSetPage —

```
(defun |htShowSetPage| (htPage branch)
 (let (setTree setData st)
 (declare (special |$path|))
  (setq setTree (|htpProperty| htPage '|setTree|))
  (setq |$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")))))))
```

———-

54.3.162 defun htShowLiteralsPage

— defun htShowLiteralsPage —

```
(defun |htShowLiteralsPage| (htPage setData)
  (|htSetLiterals| htPage (elt setData 0) (elt setData 1)
      (elt setData 4) (elt setData 5) '|htSetLiteral|))
```

———-

54.3.163 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)))
(|htpSetProperty| page '|variable| variable)
(|bcHt| (cons "\\centerline{Set {\\em "
              (cons name
                    (cons "}}\\newline" nil))))
(|bcHt| (cons "{\\em Description: } "
              (cons message
                    (cons "\\newline\\vspace{1} "
                          nil))))
(|bcHt| "Select one of the following: \\newline\\tab{3} ")
(setq links
        (prog (g2)
          (setq g2 nil)
          (return
            (DO ((G167465 values (cdr G167465))
                 (opt nil))
                ((or (atom G167465)
                     (progn
                       (setq opt (car G167465))
                       nil))
                 (NREVERSE0 g2))
              (SEQ (EXIT (setq g2
                               (cons
                                (cons
                                 (CONCAT ""
                                  (PRINC-TO-STRING opt))
                                 (cons "\\newline\\tab{3}"
                                  (cons functionToCall
                                   (cons opt nil))))
                                g2)))))))))
(|htMakePage| (cons (cons '|bcLispLinks| links) nil))
(|bcHt|
 (cons
 '|\\indent{0}\\newline\\vspace{1} The current setting is: {\\em |
              (cons (|translateTrueFalse2YesNo|
                        (eval variable))
                    (cons "} " nil))))
(|htShowPage|))))))
```

———-

54.3.164 defun htSetLiteral

— **defun htSetLiteral** —

```
(defun |htSetLiteral| (htPage val)
  (|htInitPage| "Set Command" nil)
  (set (|htpProperty| htPage '|variable|) (|translateYesNo2TrueFalse| val))
  (|htKill| htPage val))
```

———-

54.3.165 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| (list "\\centerline{Set {\\em " (elt setData 0) "}}\\newline"))
        (setq message (elt setData 1))
        (|bcHt| (list "{\\em Description: } " message "\\newline\\vspace{1} "))
        (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|
            (cons '(|domainConditions| (|Satisfies| S chkRange))
                  (cons (cons '|bcStrings|
                     (list (list 5 (|eval| (elt setData 4)) '|value|  'S)))
                        nil)))
        (|htSetvarDoneButton| "Select to Set Value" '|htSetInteger|)
        (|htShowPage|)))))
```

———-

54.3.166 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"
                     (cons nil
                           (cons "\\vspace{3}\\centerline{{\\em "
                                 (cons val
                                       (cons
 "}}\\vspace{2}\\newline\\centerline{Click on \\UpBitmap{} to re-enter value}"
                                        nil)))))))
          (t (set (|htpProperty| htPage '|variable|) val)
           (|htKill| htPage val)))))))
```

———-

54.3.167 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))))))
```

———-

54.3.168 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 (cadr (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 '|Satisfies|
                                       (cons 'S (cons checker nil)))
                                 nil))
                     (cons (cons '|text| phrase)
                           (cons (cons '|inputStrings|
                                       (cons
                                        (cons ""
                                         (cons ""
                                          (cons 60
                                           (cons currentValue
                                            (cons '|value|
                                             (cons 'S nil))))))
                                        nil))
                                 nil))))
           (|htSetvarDoneButton| "Select To Set Value" '|htSetFunCommand|)
           (|htShowPage|)))))))
```

———-

54.3.169 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| "} ")))
```

———-

54.3.170 defun htFunctionSetLiteral

— defun htFunctionSetLiteral —

```
(defun |htFunctionSetLiteral| (htPage val)
  (progn
    (|htInitPage| "Set Command" nil)
    (set (|htpProperty| htPage '|variable|)
         (|translateYesNo2TrueFalse| val))
    (|htSetFunCommandContinue| htPage val)))
```

———-

54.3.171 defun htSetFunCommand

— defun htSetFunCommand —

```
(defun |htSetFunCommand| (htPage)
 (let (variable checker value)
  (setq variable (|htpProperty| htPage '|variable|))
  (setq checker (|htpProperty| htPage '|checker|))
  (setq value (|htCheck| checker (|htpLabelInputString| htPage '|value|)))
  (set variable value)
  (|htSetFunCommandContinue| htPage value)))
```

———-

54.3.172 defun htSetFunCommandContinue

— defun htSetFunCommandContinue —

```
(defun |htSetFunCommandContinue| (htPage value)
 (let (parts continue)
  (setq parts (|htpProperty| htPage '|parts|))
  (setq continue
   (cond
    ((null parts) nil)
    ((and (consp parts)
```

```
          (consp (qcar parts)) (eq (qcaar parts) '|break|)
          (consp (qcdar parts)) (eq (qcddar parts) nil))
       (|eval| (qcadar parts)))
    (t t)))
  (cond
   (continue
    (|htpSetProperty| htPage '|parts| (qcdr parts))
      (|htShowFunctionPageContinued| htPage))
   (t (|htKill| htPage value)))))
```

———-

54.3.173 defun htKill

— defun htKill —

```
(defun |htKill| (htPage value)
  (declare (ignore htPage))
  (prog (string)
  (declare (special |$path|))
    (return
      (progn
        (|htInitPage| "System Command" nil)
        (setq string
                 (CONCAT "{\\em )set "
                         (|listOfStrings2String|
                             (cons value |$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|)))))
```

———-

54.3.174 defun htSetNotAvailable

— defun htSetNotAvailable —

```
(defun |htSetNotAvailable| (htPage whatToType)
 (let (page string)
  (setq page
    (|htInitPage| "Unavailable Set Command" (|htpPropertyList| htPage)))
```

```
    (|htInitPage| "Unavailable System Command" nil)
    (setq string (concat "{\\em " whatToType "}"))
    (|htMakePage|
            (cons '(|text| "\\vspace{1}\\newline"
 "{Sorry, but this system command is not available through HyperDoc. Please directly issue this command in an AX
                           "\\vspace{2}\\newline\\centerline{\\tt")
                  (cons (cons '|text| string) nil)))
    (|htMakePage| '((|text| . "}\\vspace{1}\\newline")))
    (|htProcessDoitButton| (list "Press to Remove Page" "" '|htDoNothing| ))
    (|htShowPage|)))
```

———-

54.3.175 defun htDoNothing

— defun htDoNothing —

```
(defun |htDoNothing| (htPage command)
 (declare (ignore htPage command))
 nil)
```

———-

54.3.176 defun htCheck

— defun htCheck —

```
(defun |htCheck| (checker value)
  (cond
    ((consp checker) (|htCheckList| checker (|parseWord| value)))
    (t (funcall checker value))))
```

———-

54.3.177 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))))))
```

———-

54.3.178 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|)))
        (cond
          ((|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)))))))
```

———-

54.3.179 defun translateYesNoToTrueFalse

— defun translateYesNoToTrueFalse —

```
(defun |translateYesNoToTrueFalse| (x)
  (cond
    ((eq x '|yes|) t)
    ((eq x '|no|) nil)
    (t x)))
```

———-

54.3.180 defun chkNameList

— 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")))))))
```

———-

54.3.181 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")))))
```

———-

54.3.182 defun chkOutputFileName

— defun chkOutputFileName —

```
(defun |chkOutputFileName| (s)
  (cond
    ((|member| (|bcString2WordList| s) '(CONSOLE |console|))
     '|console|)
    (t (|chkDirectory| s))))
```

———-

54.3.183 defun chkDirectory

— defun chkDirectory —

```
(defun |chkDirectory| (s) s)
```

———-

54.3.184 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")))))
```

———-

54.3.185 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|)
         (CONCAT "Please enter an integer greater than "
                 (|stringize| (- |$htInitial| 1))))
        (t
         (CONCAT "Please enter an integer between "
                 (|stringize| |$htInitial|) " and "
                 (|stringize| |$htFinal|)))))))
```

———-

54.3.186 defun chkAllNonNegativeInteger

— 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"))))
```

———-

54.3.187 defun htMakePathKey,fn

— defun htMakePathKey,fn —

```
(defun |htMakePathKey,fn| (a b)
  (SEQ (if (null b) (EXIT a))
       (EXIT (|htMakePathKey,fn|
```

```
            (CONCAT a "." (PNAME (car b)))
            (cdr b)))))
```

———-

54.3.188 defun htMakePathKey

— defun htMakePathKey —

```
(defun |htMakePathKey| (path)
  (cond
    ((null path) (|systemError| "path is not set"))
    (t
     (intern (|htMakePathKey,fn| (PNAME (car path)) (cdr path))))))
```

———-

54.3.189 defun htMarkTree

— defun htMarkTree —

```
(defun |htMarkTree| (tree n)
  (SEQ (progn
         (rplacd (last 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))))))))))))
```

———-

54.3.190 defun htSetHistory

— defun htSetHistory —

```
(defun |htSetHistory| (htPage)
 (let (msg data)
  (setq msg
 '|when the history facility is on (yes), results of computations are saved in memory|)
  (setq data
    (list '|history| msg '|history| 'literals '|$HiFiAccess|
          '(|on| |off| |yes| |no|)))
  (|htShowLiteralsPage| htPage data)))
```

———-

54.3.191 defun htSetOutputLibrary

— defun htSetOutputLibrary —

```
(defun |htSetOutputLibrary| (htPage)
  (|htSetNotAvailable| htPage ")set compiler output"))
```

———-

54.3.192 defun htSetInputLibrary

— defun htSetInputLibrary —

```
(defun |htSetInputLibrary| (htPage)
  (|htSetNotAvailable| htPage ")set compiler input"))
```

———-

54.3.193 defun htSetExpose

— defun htSetExpose —

```
(defun |htSetExpose| (htPage)
  (|htSetNotAvailable| htPage ")set expose"))
```

———-

54.3.194 defun htSetOutputCharacters

— defun htSetOutputCharacters —

```
(defun |htSetOutputCharacters| (htPage)
  (|htSetNotAvailable| htPage ")set output characters"))
```

———-

54.3.195 defun htSetLinkerArgs

— defun htSetLinkerArgs —

```
(defun |htSetLinkerArgs| (htPage)
  (|htSetNotAvailable| htPage ")set fortran calling linker"))
```

———-

54.3.196 defun htSetCache

— defun htSetCache —

```
(defun |htSetCache| (&REST arg &AUX options htPage)
 (declare (special |$valueList| |$path|))
 (setq htPage (car arg))
 (setq options (cdr arg))
 (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{1}\\newline ")
    (|domainConditions| (|Satisfies| S chkNameList))
    (|text|
"Enter below a list of interpreter functions you would like specially cached. "
    "Use the name {\\em all} to give a default setting for all "
    "interpreter functions. " "\\vspace{1}\\newline "
    "Enter {\\em all} or a list of names (separate names by blanks):")
     (|inputStrings| ("" "" 60 "all" names S))
     (|doneButton| "Push to enter names" |htCacheAddChoice|)))
 (|htShowPage|))
```

———-

54.3.197 defun htCacheAddChoice

— defun htCacheAddChoice —

```
(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))
                   nil)
                (SEQ (EXIT (|htMakePage|
                              (cons (cons '|inputStrings|
                                     (cons
                                      (cons
                                       (CONCAT "Function {\\em "
                                        name
                                        "} will cache")
                                       (cons "values"
                                        (cons 5
                                         (cons 10
                                          (cons
                                           (|htMakeLabel|
                                            "c" i)
                                           (cons 'ALLPI nil))))))
                                      nil))
                                    nil)))))
              (|htSetvarDoneButton| "Select to Set Values" '|htCacheSet|)
              (|htShowPage|))))))))
```

———-

54.3.198 defun htMakeLabel

— **defun htMakeLabel** —

```
(defun |htMakeLabel| (prefix i)
  (intern (concat prefix (|stringize| i))))
```

———-

54.3.199 defun htCacheSet

— 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
                              (progn
                                (setq name (car G167774))
                                (setq val (CDR G167774))
                                G167774)
                              nil))
                        nil)
```

```
                (SEQ (EXIT (cond
                            ((NEQUAL val |$cacheCount|)
                             (progn
                               (|bcHt| "\\newline function {\\em ")
                               (|bcHt| (|stringize| name))
                               (|bcHt| "} will cache ")
                               (|htAllOrNum| val)
                               (|bcHt| "} values")))))))
           (|htProcessDoitButton|
               (cons "Press to Remove Page"
                     (cons "" (cons '|htDoNothing| nil))))
           (|htShowPage|))))))
```

———-

54.3.200 defun htAllOrNum

— **defun htAllOrNum** —

```
(defun |htAllOrNum| (val)
  (|bcHt| (cond
           ((eq val '|all|) "{\\em all")
           ((eql val 0) "{\\em no")
           (t
            (CONCAT "the last {\\em "
                    (|stringize| val))))))
```

———-

54.3.201 defun htCacheOne

— **defun htCacheOne** —

```
(defun |htCacheOne| (names)
  (prog (page)
    (return
      (progn
        (setq page (|htInitPage| (|mkSetTitle|) nil))
        (|htpSetProperty| 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|)))))
```

———-

54.3.202 defvar $historyDisplayWidth

— **initvars** —

```
(defvar |$historyDisplayWidth| 120)
```

———-

54.3.203 defvar $newline

— **initvars** —

```
(defvar |$newline| #\Newline)
```

———-

54.3.204 defun downlink

— **defun downlink** —

```
(defun |downlink| (page)
 (|htInitPage| "Bridge" nil)
 (|htSay| "\\replacepage{" page "}")
 (|htShowPage|))
```

———-

54.3.205 defun dbNonEmptyPattern

— **defun dbNonEmptyPattern** —

```
(defun |dbNonEmptyPattern| (pattern)
  (cond
    ((null pattern) "*")
    (t (setq pattern (PRINC-TO-STRING pattern))
     (cond ((> (|#| pattern) 0) pattern) (t "*")))))
```

———-

54.3.206 defun htSystemVariables,gn

— defun htSystemVariables,gn —

```
(defun |htSystemVariables,gn| (t1 al)
 (let (class key options)
 (declare (special |$heading| |$levels|))
   (setq class (caddr t1))
   (setq key (cadddr t1))
   (setq options (cadr (cddddr t1)))
   (cond
    ((null (member class |$levels|)) al)
    ((or (or (eq key 'literals) (eq key 'integer))
         (eq key 'string))
      (cons (cons |$heading| t1) al))
    ((eq key 'tree)
      (|htSystemVariables,fn| options al nil))
    ((eq key 'function)
      (cons (cons |$heading| t1) al))
    (t (|systemError| key)))))
```

———-

54.3.207 defun htSystemVariables,fn

— defun htSystemVariables,fn —

```
(defun |htSystemVariables,fn| (t1 al firstTime)
  (declare (special |$heading|))
  (SEQ (if (atom t1) (EXIT al))
       (if firstTime (setq |$heading| (|opOf| (car t1))) nil)
       (EXIT (|htSystemVariables,fn| (cdr t1)
                 (|htSystemVariables,gn| (car t1) al) firstTime))))
```

———-

54.3.208 defun htSystemVariables,displayOptions

— defun htSystemVariables,displayOptions —

```
(defun |htSystemVariables,displayOptions| (name class variable val options)
  (SEQ (if (eq class 'integer)
           (EXIT (SEQ (|htMakePage|
                         (cons (cons '|bcLispLinks|
                                     (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))
                            nil))
                  (|htMakePage|
                      '((|domainConditions|
                            (|isDomain| INT (|Integer|)))))
                  (EXIT (|htMakePage|
(list
  (cons '|bcStrings| (list (list 5 (princ-to-string val) name 'int)))))))))
(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
                                         (cons x nil))
                                        nil))))
                                    nil))
                             nil))))))))
```

———-

54.3.209 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
                                     (cons '|bcStrings|
                                      (cons
                                       (cons 30
                                        (cons (PRINC-TO-STRING val)
                                         (cons name
                                          (cons valuesOrFunction nil))))
                                       nil))
                                     nil))))))
           (EXIT (|htSystemVariables,displayOptions| name class
                     var val valuesOrFunction))))))
```

———-

54.3.210 defun htSystemVariables

— defun htSystemVariables —

```
(defun |htSystemVariables| ()
  (prog (|$levels| |$heading| classlevel table heading 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))
             nil)
          (SEQ (EXIT (progn
                       (|htSay| "\\newline\\item ")
                       (cond
                         ((equal heading lastHeading)
                          (|htSay| "\\tab{8}"))
                         (t
                          (|htSay| heading
                                   "\\tab{8}")
                          (setq lastHeading heading)))
                       (|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|
                                         name class var
                                         valuesOrFunction)))))))))
                           (t (setq val (|eval| variable))
                            (|htSystemVariables,displayOptions|
                                name key variable val options)))))))
        (|htSay| "\\endmenu") (|htShowPage|)))))))
```

———-

54.3.211 defun htSetSystemVariableKind

— **defun htSetSystemVariableKind** —

```
(defun |htSetSystemVariableKind| (htPage arg)
 (let (variable name fun value)
  (setq variable (car arg))
  (setq name (cadr arg))
  (setq fun (caddr arg))
```

```
  (setq value (|htpLabelInputString| htPage name))
  (when (and (stringp value) fun) (setq value (funcall fun value)))
  (set variable value)
  (|htSystemVariables|)))
```

———-

54.3.212 defun htSetSystemVariable

— defun htSetSystemVariable —

```
(defun |htSetSystemVariable| (htPage arg)
 (declare (ignore htPage))
 (let (name value)
   (setq name (car arg))
   (setq value (cadr arg))
   (setq value
    (cond
     ((eq value '|on|) t)
     ((eq value '|off|) nil)
     (t value)))
   (set name value)
   (|htSystemVariables|)))
```

———-

54.3.213 defun htGloss

— defun htGloss —

```
(defun |htGloss| (pattern)
  (|htGlossPage| nil
      (or (|dbNonEmptyPattern| pattern) "*") t))
```

———-

54.3.214 defun htGlossPage

— 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
             (CONCAT (getenviron "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
                        (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)
             (SEQ (EXIT (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|)))))))))
```

———-

54.3.215 defun gatherGlossLines

— defun gatherGlossLines —

```
(defun |gatherGlossLines| (results defstream)
  (prog (n keyAndTick byteAddress line k pointer def x
         j nextPointer xtralines acc)
```

```
(declare (special |$tick|))
  (return
    (SEQ (progn
           (setq acc nil)
           (DO ((G168098 results (cdr G168098))
                (keyline nil))
               ((or (atom G168098)
                    (progn (setq keyline (car G168098)) nil))
                nil)
             (SEQ (EXIT (progn
                          (setq n
                                (|charPosition| |$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
                                 (CONCAT keyAndTick def
                                  (prog (G168110)
                                    (setq G168110 "")
                                    (return
                                      (DO
                                       ((G168115
```

```
                                          (NREVERSE xtralines)
                                          (CDR G168115))
                                         (G168081 nil))
                                        ((OR (ATOM G168115)
                                          (progn
                                            (setq G168081
                                             (car G168115))
                                            nil))
                                         G168110)
                                         (SEQ
                                          (EXIT
                                           (setq G168110
                                            (CONCAT G168110
                                             G168081)))))))
                                  acc))))))
            (reverse acc))))))
```

———-

54.3.216 defun htGlossSearch

— **defun htGlossSearch** —

```
(defun |htGlossSearch| (htPage junk)
  (declare (ignore junk))
  (|htGloss| (|htpLabelInputString| htPage '|filter|)))
```

———-

54.3.217 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
                    (cons (cons "Missing search string"
                                nil)
                          (cons nil
                                (cons
      "\\vspace{2}\\centerline{To select one of the greek letters:}\\newline "
                                      (cons
```

```
"\\centerline{{\\em first} enter a search key into the input area}\\newline "
                                  (cons
"\\centerline{{\\em 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)
                 (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{{\\em |
                         ss
                        '|}}\\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
                    (nonmatches
                       (|htSay|
                "The greek letters that {\\em match} your search string {\\em "
                              ss "}:"))
                    (t
                     (|htSay| "Your search string {\\em "
                              ss
                              '|} matches all of the greek letters:|)))
                  (|htSay| "{\\em \\table{")
                  (DO ((G168158 matches (CDR G168158))
                       (x nil))
                      ((or (atom G168158)
                           (progn (setq x (car G168158)) nil))
                       nil)
                    (SEQ (EXIT (|htSay| "{" x
```

```
                                        "}"))))
                    (|htSay| "}}\\vspace{1}")
                    (cond
                      (nonmatches
                          (|htSay|
"The greek letters that {\\em do not match} your search string:{\\em \\table{")
                          (DO ((G168167 nonmatches (CDR G168167))
                               (x nil))
                              ((or (atom G168167)
                                   (progn
                                     (setq x (car G168167))
                                     nil))
                               nil)
                            (SEQ (EXIT (|htSay| "{" x
                                        "}"))))
                          (|htSay| "}}")))
                    (|htShowPage|)))))))))))
```

———-

54.3.218 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"
                                nil)
                          (cons nil
                                (cons
      "\\vspace{2}\\centerline{To select one of the lines of text:}\\newline "
                                      (cons
  "\\centerline{{\\em first} enter a search key into the input area}\\newline "
                                       (cons
  "\\centerline{{\\em then } move the mouse cursor to the work {\\em search} and click}"
                                        nil)))))))
               (t (setq filter s)
                (setq lines
                          (cons
             "{{\\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
                      (|htSay| "Your search string {\\em "
                               s '|} matches both lines:|)))
                   (|htSay| "{\\em \\table{")
                   (DO ((G168200 matches (CDR G168200))
                        (x nil))
                       ((or (atom G168200)
                            (progn (setq x (car G168200)) nil))
                        nil)
                     (SEQ (EXIT (|htSay| "{" x
                                         "}"))))
                   (|htSay| "}}\\vspace{1}")
                   (cond
                     (nonmatches
                         (|htSay|
          "The line that {\\em does not match} your search string:{\\em \\table{")
                         (DO ((G168209 nonmatches (cdr G168209))
                              (x nil))
                             ((or (atom G168209)
                                  (progn
```

```
                                    (setq x (car G168209))
                                    nil))
                               nil)
                           (SEQ (EXIT (|htSay| "{" x
                                       "}"))))
                      (|htSay| "}}")))
                 (|htShowPage|))))))))))
```

———-

54.3.219 defun htTutorialSearch

— 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))
             (setq source "$AXIOM/doc/hypertex/pages/ht.db")
             (setq target "/tmp/temp.text.$SPADNUM")
             (OBEY (CONCAT "$AXIOM/lib/hthits"
                           " \"" s "\" "
                           source " > " target))
             (setq lines (|dbReadLines| '|temp|))
             (|htInitPageNoScroll| nil
                 (cons "Tutorial Pages mentioning {\\em "
                       (cons pattern (cons "}" nil))))
             (|htSay| "\\beginscroll\\table{")
             (DO ((G168241 lines (cdr G168241)) (line nil))
                 ((or (atom G168241)
                      (progn (setq line (car G168241)) nil))
                  nil)
               (SEQ (EXIT (progn
                            (setq t1 (|dbParts| line 3 0))
                            (setq name (car t1))
                            (setq title (cadr t1))
                            (|htSay| (cons "{\\downlink{"
                                           (cons title
                                            (cons "}{"
                                             (cons name
                                              (cons "}}" nil))))))))))
             (|htSay| "}")
```

```
(|htShowPage|))))))
```

54.3.220 defun mkUnixPattern

— defun mkUnixPattern —

```
(defun |mkUnixPattern| (s)
  (prog (starPositions k u)
  (declare (special |$wild|))
    (return
      (SEQ (progn
             (setq u (|mkUpDownPattern| s))
             (setq starPositions
                      (reverse (prog (G168264)
                                     (setq G168264 nil)
                                     (return
                                       (DO
                                        ((G168270
                                          (+ (- 1)
                                           (MAXINDEX u)))
                                         (i 1 (QSADD1 i)))
                                        ((QSGREATERP i G168270)
                                         (NREVERSE0 G168264))
                                         (SEQ
                                          (EXIT
                                           (cond
                                             ((equal (elt u i)
                                                |$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
                                (CONCAT (SUBSTRING u 0 i)
                                 ".*"
                                 (SUBSTRING u (1+ i) nil))))))
             (cond
               ((NEQUAL (elt u 0) |$wild|)
                (setq u (CONCAT "[^a-zA-Z]" u)))
               (t (setq u (SUBSTRING u 1 nil))))
             (cond
               ((NEQUAL (elt u (setq k (MAXINDEX u))) |$wild|)
                (setq u (CONCAT u "[^a-zA-Z]")))
               (t (setq u (SUBSTRING u 0 k))))
             u)))))
```

Chapter 55

Browser Support Code

55.1 Pages Initiated from HyperDoc Pages

55.1.1 Search routines

55.1.2 defun dKind

— **defun dbKind 0** —

```
(defun |dbKind| (line)
 (elt line 0))
```

————-

55.1.3 defun checkFilter

[trimString p**??**]

— **defun checkFilter** —

```
(defun |checkFilter| (filter)
  (setq filter (princ-to-string filter))
  (if (string= filter "")
   "*"
   (|trimString| filter)))
```

————-

;concatWithBlanks r == ; r is [head,:tail] =¿ ; tail =¿ CONCAT(head,’” ”,concatWithBlanks tail) ; head ; ”””

55.1.4 defun Concatenate words with blanks

— **defun concatWithBlanks 0** —

```
(defun |concatWithBlanks| (r)
 (if (consp r)
  (format nil "~{~a~^ ~}" r)
  ""))
```

————-

55.1.5 defun Make constructor names lowercase

[hget p1105]
[ifcar p??]
[conLowerCaseConTran p1420]
[$lowerCaseConTb p??]

— **defun conLowerCaseConTran** —

```
(defun |conLowerCaseConTran| (x)
 (declare (special |$lowerCaseConTb|))
 (cond
  ((identp x) (or (ifcar (hget |$lowerCaseConTb| x)) x))
  ((atom x) x)
  (t (loop for y in x collect (|conLowerCaseConTran| y)))))
```

————-

55.1.6 defun string2Constructor

[downcase p1140]
[hget p1105]
[ifcar p??]
[$lowerCaseConTb p??]

— **defun string2Constructor** —

```
(defun |string2Constructor| (x)
 (declare (special |$lowerCaseConTb|))
 (cond
  ((null (stringp x)) x)
  (t (or (ifcar (hget |$lowerCaseConTb| (intern (downcase x)))) x))))
```

————-

55.1.7 defvar dbDelimiters

— **initvars** —

```
(defvar |$dbDelimiters| (list #\space #\( #\) ))
```

————-

55.1.8 defun String to words respecting delimiters

This breaks a string into words respecting delimiters, so if

```
$dbDelimiters = ( #\space #\( #\) )
```

then

```
(|dbString2Words| "now is (the) time")
Value = ("now" "is" #\( "the" #\) "time")
```

[dbWordFrom p1421]

— **defun dbString2Words** —

```
(defun |dbString2Words| (z)
  (loop
   with i = 0
   with pair = nil
   do (setq pair (|dbWordFrom| z i))
   while (and (consp pair) (= (length pair) 2)) ; dbWordFrom(l,i) is [w,i]]
   do (setq i (second pair))
   collect (first pair)))
```

————-

55.1.9 defun Next word respecting delimiters

This returns the next word or the next delimiter. So given

```
$dbDelimiters = ( #\space #\( #\) )
(|dbWordFrom| "now is (the) time")

(|dbWordFrom| b 0)   Value = ("now" 3)
(|dbWordFrom| b 3)   Value = ("is" 6)
(|dbWordFrom| b 6)   Value = (#\( 8)
(|dbWordFrom| b 8)   Value = ("the" 11)
(|dbWordFrom| b 11)  Value = (#\) 12)
(|dbWordFrom| b 12)  Value = ("time" 17)
(|dbWordFrom| b 17)  Value = NIL
```

[maxindex p**??**]
[member p1108]
[concat p1107]
[$dbDelimiters p1421]

— **defun dbWordFrom** —

```
(defun |dbWordFrom| (z i)
 (let (maxIndex c ch buf k g1)
 (declare (special |$dbDelimiters|))
  (setq maxIndex (maxindex z))
  (loop while (and (>= maxIndex i) (char= (elt z i) #\space)) do (incf i))
  (if (and (>= maxIndex i) (|member| (elt z i) |$dbDelimiters|))
    (list (elt z i) (+ i 1))
    (progn
     (setq k
      (do ((g2 nil g1) (j i (+ j 1)))
           ((or g2 (> j maxIndex)) g1)
        (unless (|member| (elt z j) |$dbDelimiters|) (setq g1 (or g1 j)))))
     (when k
      (setq buf "")
      (do ()
          ((null (and (<= k maxIndex)
                      (null (|member| (setq c (elt z k)) |$dbDelimiters|))))
                   nil)
       (setq ch (if (char= c #\_) (elt z (setq k (+ 1 k))) c))
       (setq buf (concat buf ch))
       (setq k (+ k 1)))
      (list buf k))))))
```

———-

This creates a page for any cat, dom, package, default package

```
constructors Cname\#\E\sig \args   \abb \comments (C is C, D, P, X)
```

There are 8 parts of an htPage:

1. kind
2. name
3. nargs
4. xflag
5. sig
6. args
7. abbrev
8. comments

[dbXParts p**??**]
[mkConform p1454]
[opOf p**??**]
[capitalize p**??**]
[ncParseFromString p1172]
[dbSourceFile p**??**]
[dbConformGenUnder p**??**]
[concat p1107]
[isExposedConstructor p973]
[htInitPageNoScroll p1349]

[htAddHeading p1350]
[htSayStandard p1350]
[htpSetProperty p1342]
[dbShowConsDoc1 p1470]
[addParameterTemplates p??]
[htSay p1347]
[htSayStandard p1350]
[kPageContextMenu p??]
[htShowPageNoScroll p1351]
[$atLeastOneUnexposed p??]
[$conformsAreDomains p??]

— **defun kPage** —

```
(defun |kPage| (&rest a1)
 (let (parts name nargs sig args form isFile kind
       conform conname capitalKind signature sourceFileName constrings
       emString heading page options line)
  (declare (special |$conformsAreDomains| |$atLeastOneUnexposed|))
   (setq line (car a1))
   (setq options (cdr a1))
   ; constructors Cname\#\E\sig \args   \abb \comments (C is C, D, P, X)
   (setq parts (|dbXParts| line 7 1))
   (setq kind (first parts))
   (setq name (second parts))
   (setq nargs (third parts))
   (setq sig (fifth parts))
   (setq args (sixth parts))
   (setq form (ifcar options))
   (setq isFile (null kind))
   (setq kind (or kind "package"))
   (rplaca parts kind)
   (setq conform (|mkConform| kind name args))
   (setq conname (|opOf| conform))
   (setq capitalKind (|capitalize| kind))
   (setq signature (|ncParseFromString| sig))
   (setq sourceFileName (|dbSourceFile| (intern name)))
   (setq constrings
     (if (ifcdr form)
      (|dbConformGenUnder| form)
      (list (concat name args))))
   (setq emString (cons "{\\sf " (append constrings (list "}"))))
   (setq heading (cons capitalKind (cons " " emString)))
   (unless (|isExposedConstructor| conname)
     (setq heading (cons "Unexposed " heading)))
   (setq page (|htInitPageNoScroll| NIL))
   (|htAddHeading| heading)
   (|htSayStandard| '|\\beginscroll |)
   (|htpSetProperty| page '|isFile| t)
   (|htpSetProperty| page '|parts| parts)
   (|htpSetProperty| page '|heading| heading)
   (|htpSetProperty| page '|kind| kind)
   (|htpSetProperty| page '|conform| conform)
```

```
  (|htpSetProperty| page '|signature| signature)
  ; what follows is stuff from kiPage with domain = nil
  (setq |$conformsAreDomains| nil)
  (|dbShowConsDoc1| page conform nil)
  (when (and (nequal kind '|category|) (> nargs 0))
    (|addParameterTemplates| page conform))
  (when |$atLeastOneUnexposed|
    (|htSay| "\\newline{}{\\em *} = unexposed"))
  (|htSayStandard| '|\\endscroll |)
  (|kPageContextMenu| page)
  (|htShowPageNoScroll|)))
```

———-

55.1.10 defun Hyperdoc category search

[constructorSearch p1426]

— **defun cSearch** —

```
(defun |cSearch| (filter)
  (|constructorSearch| (|checkFilter| filter) '|c| "category"))
```

———-

55.1.11 defun Hyperdoc default domain search

[constructorSearch p1426]

— **defun xSearch** —

```
(defun |xSearch| (filter)
  (|constructorSearch| (|checkFilter| filter) '|x| "default package"))
```

———-

55.1.12 defun Hyperdoc domain search

[constructorSearch p1426]

— **defun dSearch** —

```
(defun |dSearch| (filter)
  (|constructorSearch| (|checkFilter| filter) '|d| "domain"))
```

———-

55.1.13 defun Hyperdoc package search

[constructorSearch p1426]

— defun pSearch —

```
(defun |pSearch| (filter)
  (|constructorSearch| (|checkFilter| filter) '|p| "package"))
```

———-

55.1.14 defun Hyperdoc constructor search

[constructorSearch p1426]

— defun kSearch —

```
(defun |kSearch| (filter)
  (|constructorSearch| (|checkFilter| filter) '|k| "constructor"))
```

———-

55.1.15 defun Hyperdoc default constructor search

[constructorSearch p1426]

— defun ySearch —

```
(defun |ySearch| (filter)
  (|constructorSearch| (|checkFilter| filter) '|y| "constructor"))
```

———-

55.1.16 defun Read libdb.text at file-position n

— defun dbRead 0 —

```
(defun |dbRead| (n)
  (with-open-file
     (instream (concat (getenviron "AXIOM") "/algebra/libdb.text"))
   (file-position instream n)
   (read-line instream)))
```

———-

55.1.17 defun String trim with newlines removed

— defun libdbTrim 0 —

```
(defun |libdbTrim| (s)
  (string-trim '(#\space #\tab #\newline) (substitute #\space #\newline s)))
```

———-

55.1.18 defun Hyperdoc common constructor search

[dbKind p??]
[conSpecialString? p1427]
[conPage p1428]
[lassoc p??]
[downcase p1140]
[downlink p1402]
[kPage p??]
[htInitPage p1349]
[htpSetProperty p1342]
[dbName p??]
[htQuery p??]
[htShowPage p1350]
[grepSearchQuery p??]
[constructorSearchGrep p??]
[$lowerCaseConTb p??]

— **defun constructorSearch** —

```
(defun |constructorSearch| (filter key kind)
 (let (parse pageName name u line newkind page message)
 (declare (special |$lowerCaseConTb|))
  (cond
   ((null filter) nil)
   ((setq parse (|conSpecialString?| filter)) (|conPage| parse))
   ((setq pageName
     (lassoc (downcase filter)
       '(("union" . |DomainUnion|)
         ("record" . |DomainRecord|)
         ("mapping" . |DomainMapping|)
         ("enumeration" . |DomainEnumeration|))))
         (|downlink| pageName))
   (t
    (setq name (if (stringp filter) (intern filter) filter))
    (when (setq u (hget |$lowerCaseConTb| name))
      (setq filter (princ-to-string (car u))))
    (cond
      ((setq line (|conPageFastPath| (downcase filter)))
        (setq newkind
         (case (|dbKind| line)
          (#\p "package")
          (#\d "domain")
          (#\c "category")))
        (cond
         ((or (equal kind "constructor") (equal kind newkind))
           (|kPage| line))
```

```
        (t
          (setq page (|htInitPage| "Query Page" nil))
          (|htpSetProperty| page '|line| line)
          (setq message
            (list "{\\em " (|dbName| line) "} is not a {\\em " kind
                  "} but a {\\em " newkind
                  "}. Would you like to view it?\\vspace{1}" ))
          (|htQuery| message '|grepConstructorSearch| 't)
          (|htShowPage|))))
      ((equal filter "*")
        (|grepSearchQuery| kind
          (list filter key kind '|constructorSearchGrep| )))
      (t (|constructorSearchGrep| filter key kind)))))))
```

———-

55.1.19 defun conSpecialString?

[ifcar p**??**]
[string2Words p**??**]
[ncParseFromString p1172]
[member p1108]
[conLowerCaseConTran p1420]
[contained p**??**]
[kisValidType p1452]
[concat p1107]
[dbString2Words p1421]
[string2Constructor p1420]
[conSpecialString? p1427]

— **defun conSpecialString?** —

```
(defun |conSpecialString?| (&REST a1 &AUX options filter)
 (let (secondTime t1 words parse form u)
  (setq filter (car a1))
  (setq options (cdr a1))
  (setq secondtime (ifcar options))
  (setq t1 (|string2Words| filter))
  (setq parse
   (cond
    ((and (consp t1) (not (qcdr t1))) ; t1 is [s]
      (setq words (|ncParseFromString| (qcar t1))))
    ((every #'(lambda (x) (null (|member| x '("and" "or" "not")))) words)
      (|ncParseFromString| filter))))
  (cond
   ((null parse) nil)
   (t
    (setq form (|conLowerCaseConTran| parse))
    (cond
     ((or (member (ifcar form) '(|and| |or| |not|)) (contained '* form)) nil)
     ((equal filter "Mapping") nil)
     ((setq u (|kisValidType| form)) u)
```

```
    (secondTime nil)
    (t
     (setq u
      (reduce #'concat
       (loop for x in (|dbString2Words| filter)
             collect (|string2Constructor| x))))
     (|conSpecialString?| u t)))))))
```

———-

55.1.20 Page construction

55.1.21 defun conPage

[form2HtString p??]
[downcase p1140]
[lassq p??]
[downlink p1402]
[conPageFastPath p1429]
[kPage p??]
[ySearch p1425]
[$conArgstrings p??]

— **defun conPage** —

```
(defun |conPage| (a &rest b)
 (let (|$conArgstrings| form da pageName line)
 (declare (special |$conArgstrings|))
  ; the next 4 lines allow e.g. MATRIX INT ==> Matrix Integer (see kPage)
  (setq form (if (atom a) (cons a b) a))
  (setq |$conArgstrings| (loop for x in (ifcdr 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|)
          (|enumeration| . |DomainEnumeration|))))
       (|downlink| pageName))
    ((setq line (|conPageFastPath| da)) ; lower case name of cons?
      (|kPage| line form))
    ((setq line (|conPageFastPath| (upcase a))) ; upper case an abbrev?
      (|kPage| line form))
    (t (|ySearch| a)))))) ; slow search (include default packages)
```

———-

55.1.22 defun gets line quickly for constructor name or abbreviation

[length p??]
[charPosition p??]
[lassq p??]
[dbRead p1425]
[conPageConEntry p1429]
[$lowerCaseConTb p??]

— **defun conPageFastPath** —

```
(defun |conPageFastPath| (x)
 (let (s name entry lineNumber)
 (declare (special |$lowerCaseConTb|))
  (setq s (princ-to-string x))
  (unless (> (|#| s) (|charPosition| #\* s 0)) ; quit if name has * in it
    (setq name (cond ((stringp x) (intern x)) (t x)))
    (setq entry (hget |$lowerCaseConTb| name))
    (when entry
      ;'dbLineNumbers property is set by function dbAugmentConstructorDataTable
      (if (setq lineNumber (lassq '|dbLineNumber| (cddr entry)))
        (|dbRead| lineNumber)
        (|conPageConEntry| (car entry)))))))
```

———-

55.1.23 defun conPageConEntry

[conPageConEntry buildLIbdbConEntry (vol9)]
[$conname p??]
[$conform p??]
[$exposed? p??]
[$doc p??]
[$kind p??]

— **defun conPageConEntry** —

```
(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)))
```

———-

55.1.24 defun kdPageInfo

[htSay p1347]
[nequal p??]
[bcHt p1347]
[htSayStandard p1350]
[kPageArgs p??]
[length p??]
[extractFileNameFromPath p??]
[subseq p??]
[getdatabase p1070]
[htSay p1347]
[htMakePage p1351]

— **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 | (princ-to-string nargs) '| arguments:|))))))
  (|htSayStandard| "\\indentrel{2}")
  (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| "."))))
```

———-

55.1.25 defun kArgPage

[htpProperty p1342]
[getConstructorModemap p??]
[position p??]

[sublisFormal p??]
[mkDomTypeForm p1431]
[domainDescendantsOf p1431]
[htpSetProperty p1342]
[dbShowCons p1466]

— **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|)))
```

———-

55.1.26 defun mkDomTypeForm

[sublislis p??]
[mkDomTypeForm p1431]
[hasIndent p??]

— **defun mkDomTypeForm** —

```
(defun |mkDomTypeForm| (typeForm conform domname)
 (cond
   (domname (sublislis (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)))
```

———-

55.1.27 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))))))
    alist))
  ; keep only those domains that appear in ALL parts of Join
  (jfn (arg domlist)
   (let (y r item pred u keepList alist)
    (setq y (car arg))
    (setq r (cdr arg))
    (setq alist (|domainsOf| y (ifcar domlist)))
    (dolist (x r)
      (setq domlist (ifcdr domlist))
      (when (and (consp x) (eq (qcar x) 'category) (consp (qcdr x)))
        (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| #'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))))
```

———-

55.2 Branches of Constructor Page

55.2.1 defun kiPage

[htpProperty p1342]
[mkConform p1454]
[kDomainName p1450]
[errorPage p??]
[capitalize p??]
[htInitPage p1349]
[htCopyProplist p??]
[dbShowConsDoc1 p1470]
[htShowPage p1350]
[$conformsAreDomains p??]

— **defun kiPage** —

```
(defun |kiPage| (htPage junk)
 (declare (ignore junk))
 (let (lt1 kind name nargs args conform domname heading page)
  (declare (special |$conformsAreDomains|))
   (setq lt1 (|htpProperty| htPage '|parts|))
   (setq kind (first lt1))
   (setq name (second lt1))
   (setq nargs (third lt1))
   (setq args (sixth lt1))
   (setq conform (|mkConform| kind name args))
   (setq domname (|kDomainName| htPage kind name nargs))
   (cond
     ((and (consp domname) (eq (qcar domname) '|error|))
           (|errorPage| htPage domname))
     (t
      (setq heading
        (list "Description of " (|capitalize| kind) " {\\sf " name args "}"))
      (setq page (|htInitPage| heading (|htCopyProplist| htPage)))
      (setq |$conformsAreDomains| domname)
      (|dbShowConsDoc1| htPage conform nil)
      (|htShowPage|)))))
```

———-

55.2.2 defun kePage

[htpProperty p1342]
[concat p1107]
[kDomainName p1450]
[errorPage p??]
[htpSetProperty p1342]
[mkConform p1454]
[opOf p??]
[capitalize p??]
[form2HtString p??]
[sublisFormal p??]
[ifcdr p??]
[getConstructorExports p??]
[simpHasPred p??]
[pluralSay p??]
[length p??]
[htInitPage p1349]
[htCopyProplist p??]
[htSayStandard p1350]
[htpSetProperty p1342]
[htMakePage p1351]
[menuButton p??]
[htSay p1347]
[bcConPredTable p??]
[htBigSkip p??]
[kePageDisplay p1436]
[kePageOpAlist p1435]
[htSowPage p??]
[$confomrsAreDomains p??]

— **defun kePage** —

```
(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))
    (|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| (+ (+ (|#| conlist) (|#| attrlist)) (|#| oplist))
              "Export" "Exports"))
(setq page
 (|htInitPage| (append prefix (cons " of " heading))
               (|htCopyProplist| htPage)))
(|htSayStandard| "\\beginmenu ")
(|htpSetProperty| page '|data| data)
(when conlist
  (|htMakePage|
    (list
     (list '|bcLinks|
      (list (|menuButton|) "" '|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|)))))
```

———-

55.2.3 defun kePageOpAlist

[lassoc p??]
[insertAlist p874]
[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 (lassoc op opAlist))
    (setq opAlist
     (|insertAlist| (|zeroOneConvert| op)
                    (cons (list sig pred) u)
                    opAlist)))
   opAlist))
```

———-

55.2.4 defun kePageDisplay

[length p??]
[htpSetProperty p1342]
[htMakePage p1351]
[menuButton p??]
[htSayStandard p1350]
[htSay p1347]
[pluralize p??]
[dbGatherData p??]
[dbSowOpItems p??]

— **defun kePageDisplay** —

```
(defun |kePageDisplay| (htPage which opAlist)
 (let (count total expandProperty data)
  (setq count (|#| opAlist))
  (cond
   ((eql count 0) nil)
   (t
    (setq total
     (apply #'+ (loop for entry in opAlist collect (|#| (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| (princ-to-string count) " names for ")))
    (if (> total 1)
      (|htSay| (princ-to-string total) " " (|pluralize| which)
```

```
            " are explicitly exported:")
      (|htSay| "1 " which " is explicitly exported:"))
    (setq data (|dbGatherData| htPage opAlist which '|names|))
    (|dbShowOpItems| which data nil)))))
```

———-

55.2.5 defun ksPage

[htpProperty p1342]
[kDomainName p1450]
[errorPage p??]
[form2HtString p??]
[htpSetProperty p1342]
[htInitPageNoScroll p1349]
[htCopyProplist p??]
[htSay p1347]
[htSayStandard p1350]
[dbSearchOrder p1438]
[dbShowCons p1466]

— **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
       (|htpSetProperty| htPage '|domname| domname)
       (|htpSetProperty| htPage '|heading| heading))
     (setq domain (unless (string= kind "category") (eval domname)))
     (setq conform (|htpProperty| htPage '|conform|))
     (setq page
      (|htInitPageNoScroll| (|htCopyProplist| htPage)
        (cons "Search order for " heading)))
     (|htSay| (concat
       "When an operation is not defined by the domain, the following "
       "domains are searched in order for a \"default definition"))
     (|htSayStandard| "\\beginscroll ")
     (setq u (|dbSearchOrder| conform domname domain))
```

```
    (|htpSetProperty| htPage '|cAlist| u)
    (|htpSetProperty| htPage '|thing| "constructor")
    (|dbShowCons| htPage '|names|)))))
```

———-

55.2.6 defun dbSearchOrder

[opOf p??]
[dbInfovec p??]
[getdatabase p1070]
[simpCatPredicate p??]
[sublislis p??]
[kTestPred p1464]
[devaluate p??]
[kFormatSlotDomain p??]
[dbSubConform p1465]
[dbAddChain p1466]
[$domain p??]
[$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))))
```

———-

55.2.7 defun kcPage

[htpProperty p1342]
[kDomainName p1450]
[qcar p??]
[errorPage p??]
[opOf p??]
[form2HtString p??]
[htInitPage p1349]
[htCopyProplist p??]
[htpSetProperty p1342]
[dbpHasDefaultCategory? p??]
[htSay p1347]
[brCon p??]
[htSayStandard p1350]
[htBeginMenu p??]
[htMakePage p1351]
[satBreak p??]
[nequal p??]
[hget p1105]
[hasNewInfoAlist p??]
[htEndMenu p??]
[htShowPage p1350]
[$defaultPackageNamesHT 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
    (if (null domname)
      (|htpProperty| htPage '|heading|)
      (list "{\\sf " (|form2HtString| domname nil t) "}")))
   (setq page
    (|htInitPage| (cons "Cross Reference for " heading)
                  (|htCopyProplist| htPage)))
   (when domname
     (|htpSetProperty| htPage '|domname| domname)
     (|htpSetProperty| htPage '|heading| heading))
   (when (and (string= kind "category")
              (|dbpHasDefaultCategory?| xpart))
     (|htSay| "This category has default package ")
     (|bcCon| (concat name #\&) ""))
   (|htSayStandard| "\\newline")
   (|htBeginMenu| 3)
   (|htSayStandard| "\\item ")
   (setq message
    (if (string= kind "category")
      (list "Categories it directly extends")
      (list "Categories the "
       (if (string= kind "default package") "package" kind)
        " belongs to by assertion")))
   (|htMakePage|
    (list
     (list '|bcLinks|
           (list "\\menuitemstyle{Parents}"
           (list (list '|text| "\\tab{12}" message)) '|kcpPage| nil))))
   (|satBreak|)
   (setq message
    (if (string= kind "category")
     (list "All categories it is an extension of")
     (list "All categories the " kind " belongs to")))
   (|htMakePage|
    (list
     (list '|bcLinks|
      (list "\\menuitemstyle{Ancestors}"
       (list (list '|text| "\\tab{12}" message)) '|kcaPage| nil))))
   (when (string= kind "category")
     (|satBreak|)
     (|htMakePage|
      (list
       (list '|bcLinks|
```

```
      (list "\\menuitemstyle{Children}"
       (list (list '|text| "\\tab{12}"
                   "Categories which directly extend this category"))))))
  (|satBreak|)
  (|htMakePage|
   (list
    (list '|bcLinks|
     (list "\\menuitemstyle{Descendants}"
      (list (list '|text| "\\tab{12}"
                  "All categories which extend this category")))))))
(|satBreak|)
(setq message "Constructors mentioning this as an argument type")
(|htMakePage|
  (list
   (list '|bcLinks|
    (list "\\menuitemstyle{Dependents}"
     (list (list '|text| "\\tab{12}" message)) '|kcdePage| nil))))
(when (nequal kind "category")
 (|satBreak|)
 (|htMakePage|
  (list
   (list '|bcLinks|
    (list "\\menuitemstyle{Lineage}"
          "\\tab{12}Constructor hierarchy used for operation lookup"
          '|ksPage| nil)))))
 (when (string= kind "category")
  (|satBreak|)
  (|htMakePage|
   (list
    (list '|bcLinks|
     (list "\\menuitemstyle{Domains}"
      (list (list '|text| "\\tab{12}"
                  "All domains which are of this category"))
       '|kcdoPage| nil)))))
 (unless (string= kind "category")
  (|satBreak|)
  (|htMakePage|
   (list
    (list '|bcLinks|
     (list "\\menuitemstyle{Clients}" "\\tab{12}Constructors"
           '|kcuPage| nil))))
   (if (hget |$defaultPackageNamesHT| conname)
    (|htSay| " which {\\em may use} this default package")
    (|htSay| " which {\\em use} this " kind)))
(when (or (nequal kind "category") (|dbpHasDefaultCategory?| xpart))
 (|satBreak|)
 (setq message
  (if (string= kind "category")
    (list "Constructors {\\em used by} its default package")
    (list "Constructors {\\em used by} the " kind)))
 (|htMakePage|
  (list
   (list '|bcLinks|
    (list "\\menuitemstyle{Benefactors}"
```

```
          (list (list '|text| "\\tab{12}" message) '|kcnPage| nil))))))
     (when (|hasNewInfoAlist| conname)
      (|satBreak|)
      (setq message (list "Cross reference for capsule implementation"))
      (|htMakePage|
       (list
        (list '|bcLinks|
         (list "\\menuitemstyle{CapsuleInfo}"
          (list (list '|text| "\\tab{12}" message)) '|kciPage| nil)))))
     (|htEndMenu| 3)
     (|htShowPage|)))))
```

———-

55.2.8 defun kcpPage

[htpProperty p1342]
[kDomainName p1450]
[errorPage p??]
[qcar p??]
[form2HtString p??]
[htpSetProperty p1342]
[opOf p??]
[htInitPage p1349]
[htCopyProplist p??]
[parentsOf p??]
[sublislis p??]
[dbShowCons p1466]

— **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|))
    (|dbShowCons| htPage choice)))))
```

———-

55.2.9 defun reduceAlistForDomain

[sublislis p??]
[simpHasPred p??]
[nreverse0 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 (nreverse0 result))
   (setq pred (cdr pair))
   (when pred (setq result (cons pair result))))))
```

———-

55.2.10 defun kcaPage

[kcaPage1 p1444]
[ancestorsOf p??]

— **defun kcaPage** —

```
(defun |kcaPage| (htPage junk)
 (declare (ignore junk))
 (|kcaPage1| htPage "category" " an "
      "ancestor" #'|ancestorsOf| nil))
```

———-

55.2.11 defun kcdPage

[kcaPage1 p1444]
[descendantsOf p??]

— **defun kcdPage** —

```
(defun |kcdPage| (htPage junk)
  (declare (ignore junk))
  (|kcaPage1| htPage "category" " a "
      "descendant" #'|descendantsOf| t))
```

————-

55.2.12 defun kcdoPage

[kcdoPage p1444]
[domainsOf p??]

— **defun kcdoPage** —

```
(defun |kcdoPage| (htPage junk)
  (declare (ignore junk))
  (|kcaPage1| htPage "domain" " a "
      "descendant" #'|domainsOf| nil))
```

————-

55.2.13 defun kcaPage1

[htpProperty p1342]
[kDomainName p1450]
[errorPage p??]
[form2HtString p??]
[htpSetProperty p1342]
[opOf p??]
[augmentHasArgs p1446]
[listSort p??]
[function p??]
[dbShowCons p1466]

— **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))
  (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 (and domname (null isCatDescendants?))
       (|htpSetProperty| htPage '|domname| domname)
       (|htpSetProperty| htPage '|heading| heading))
     (setq conform (|htpProperty| htPage '|conform|))
     (setq conname (|opOf| conform))
     (setq ancestors (FUNCALL fn conform domname))
     (unless (string= whichever "ancestor")
       (setq ancestors (|augmentHasArgs| ancestors conform)))
     (setq ancestors (|listSort| #'glesseqp ancestors))
     (|htpSetProperty| htPage '|cAlist| ancestors)
     (|htpSetProperty| htPage '|thing| whichever)
     (setq choice '|names|)
     (|dbShowCons| htPage choice)))))
```

———-

55.2.14 defun kccPage

[htpProperty p1342]
[kDomainName p1450]
[qcar p??]
[errorPage p??]
[form2HtString p??]
[htpSetProperty p1342]
[opOf p??]
[htInitPage p1349]
[htCopyProplist p??]
[augmentHasArgs p1446]
[childrenOf p??]
[reduceAlistForDomain p1443]
[dbShowCons p1466]

— **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|)))))
```

———-

55.2.15 defun augmentHasArgs

[opOf 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 (ifcdr conform))
  (cond
   (args
    (setq n (|#| args))
    (dolist (item alist (nreverse0 result))
     (setq name (car item))
     (setq p (cdr item))
     (setq pred
      (if (consp (|extractHasArgs| p))
       p
       (|quickAnd| p
        (cons '|hasArgs|
         (take n (ifcdr (|getConstructorForm| (|opOf| name))))))))
     (setq result (cons (cons name pred) result))))
  (t alist))))
```

———-

55.2.16 defun kcdePage

[htpProperty p1342]
[concat p1107]
[nequal p??]
[ncParseFromString p1172]
[opOf p??]
[getDependentsOfConstructor p1447]
[getConstructorForm p??]
[htpSetProperty p1342]
[dbShowCons p1466]

— **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|)))
```

————-

55.2.17 defun getDependentsOfConstructor

[readLibPathFast p??]
[pathname p1103]
[rread p814]
[rshut p??]

— **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))
```

———-

55.2.18 defun kcuPage

[htpProperty p1342]
[concat p1107]
[nequal p??]
[ncParseFromString p1172]
[opOf p??]
[getUsersOfConstructor p1448]
[getConstructorForm p??]
[htpSetProperty p1342]
[dbShowCons p1466]

— **defun kcuPage** —

```
(defun |kcuPage| (htPage junk)
 (declare (ignore junk))
 (let (lt1 kind name args constring conform pakname domlist cAlist)
   (setq lt1 (|htpProperty| htPage '|parts|))
   (setq kind (first lt1))
   (setq name (second lt1))
   (setq args (sixth lt1))
   (setq constring (concat name args))
   (setq conform
    (if (nequal 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|)))
```

———-

55.2.19 defun getUsersOfConstructor

[readLibPathFast p??]
[pathname p1103]
[rread p814]
[rshut p??]

— **defun getUsersOfConstructor** —

```
(defun |getUsersOfConstructor| (con)
 (let (stream val)
  (setq stream (|readLibPathFast| (|pathname| (list '|users| 'database '|a|))))
  (setq val (|rread| con stream nil))
  (rshut stream)
  val))
```

———-

55.2.20 defun kcnPage

[kDomainName p1450]
[qcar p**??**]
[errorPage p**??**]
[htpProperty p1342]
[form2HtString p**??**]
[htpSetProperty p1342]
[concat p1107]
[pname p1106]
[opOf p**??**]
[getImports p**??**]
[sublislis p**??**]
[dbShowCons p1466]

— **defun kcnPage** —

```
(defun |kcnPage| (htPage junk)
 (declare (ignore junk))
 (let (lt1 kind name nargs domname heading conform pakname domlist cAlist
       conname)
  (setq lt1 (|htpProperty| htPage '|parts|))
  (setq kind (first lt1))
  (setq name (second lt1))
  (setq conname (intern name))
  (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
       (sublislis (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|)))))
```

———-

55.2.21 defun koPageInputAreaUnchanged?

[htpLabelInputString p1342]
[concat p1107]
[htpProperty p1342]

— **defun koPageInputAreaUnchanged?** —

```
(defun |koPageInputAreaUnchanged?| (htPage nargs)
 (equal
  (loop for i from 1 to nargs
   collect
   (|htpLabelInputString| htPage (intern (concat "*" (princ-to-string i)))))
  (|htpProperty| htPage '|inputAreaList|)))
```

———-

55.2.22 defun kDomainName

[htpSetProperty p1342]
[htpLabelInputString p1342]
[getdatabase p1070]
[kArgumentCheck p1451]
[concat p1107]
[unabbrev p??]
[mkConform p1454]
[kisValidType p1452]
[dbMkEvalable p1452]
[spad-reader p??]
[$PatternVariableList p15]

— **defun kDomainName** —

```
(defun |kDomainName| (htPage kind name nargs)
 (let (inputAreaList conname args n argTailPart argString typeForm
       evaluatedTypeForm)
```

```
  (|htpSetProperty| htPage '|domname| nil)
  (setq inputAreaList
   (loop for i from 1 to nargs for var in |$PatternVariableList|
    collect (|htpLabelInputString| htPage var)))
  (|htpSetProperty| htPage '|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 (ifcdr 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)))))))
```

———-

55.2.23 defun kArgumentCheck

[conSpecialString? p1427]
[opOf p??]
[form2String p??]

— defun kArgumentCheck —

```
(defun |kArgumentCheck| (domain? s)
 (let (form)
  (cond
   ((string= s "") nil)
   ((and domain? (setq form (|conSpecialString?| s)))
    (if (null (ifcdr form))
      (list (princ-to-string (|opOf| form)))
      (|form2String| form)))
```

```
   (t (list s)))))
```

———-

55.2.24 defun dbMkEvalable

[getdatabase p1070]
[mkEvalable p994]

— **defun dbMkEvalable** —

```
(defun |dbMkEvalable| (form)
 (let (op kind)
  (setq op (car form))
  (setq kind (getdatabase op 'constructorkind))
  (if (eq kind '|category|)
   form
   (|mkEvalable| form))))
```

———-

55.2.25 defun topLevelInterpEval

[processInteractive p308]
[$ProcessInteractiveValue p310]
[$noEvalTypeMsg p1000]

— **defun topLevelInterpEval** —

```
(defun |topLevelInterpEval| (x)
 (let (|$ProcessInteractiveValue| |$noEvalTypeMsg|)
 (declare (special |$ProcessInteractiveValue| |$noEvalTypeMsg|))
  (setq |$ProcessInteractiveValue| t)
  (setq |$noEvalTypeMsg| t)
  (|processInteractive| x nil)))
```

———-

55.2.26 defun kisValidType

[processInteractive p308]
[member p1108]
[kCheckArgumentNumbers p1453]
[$ProcessInteractiveValue p310]
[$noEvalTypeMsg p1000]
[spad-reader p??]

— **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))))
```

———-

55.2.27 defun kCheckArgumentNumbers

[getdatabase p1070]
[kCheckArgumentNumber p??]

— **defun kCheckArgumentNumbers** —

```
(defun |kCheckArgumentNumbers| (tt)
 (let (conname args cosig)
  (setq conname (car tt))
  (setq args (cdr tt))
  (setq cosig (ifcdr (getdatabase conname 'cosig)))
  (every #'identity
   (loop for domain? in cosig for x in args
    collect (if domain? (|kCheckArgumentNumbers| x) t)))))
```

———-

55.2.28 defun parseNoMacroFromString

[next p298]
[function p??]
[ncloopParse p297]
[lineoftoks p363]
[incString p298]
[StreamNull p555]
[pf2Sex p531]

— **defun parseNoMacroFromString** —

```
(defun |parseNoMacroFromString| (s)
 (setq s
  (|next| #'|ncloopParse|
   (|next| #'|lineoftoks|
    (|incString| s))))
 (if (|StreamNull| s)
  nil
  (|pf2Sex| (cadar s))))
```

———-

55.2.29 defun mkConform

[nequal p??]
[parseNoMacroFromString p1453]
[sayBrightlyNT p??]
[pp p??]
[systemError p??]
[ncParseFromString p1172]
[concat p1107]

— **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))))))))
```

———-

55.3 Operation Page for a Domain Form from Scratch

55.3.1 defun conOpPage

[dbCompositeWithMap p1456]
[htpProperty p1342]
[conOpPage1 p1455]
[dbExtractUnderlyingDomain p1457]

— **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)))))
```

———-

55.3.2 defun conOpPage1

[ifcar p??]
[opOf p??]
[dbSpecialOperations p1476]
[conPageFastPath p1429]
[dbXParts p??]
[concat p1107]
[mkConform p1454]
[captialize p??]
[ncParseFromString p1172]
[dbSourceFile p??]
[isExposedConstructor p973]
[htInitPage p1349]
[htpSetProperty p1342]
[lassoc p??]
[ifcdr p??]
[koPage p1457]
[$Primitives p??]

— **defun conOpPage1** —

```
(defun |conOpPage1| (&rest arg)
 (let (bindingsAlist conname domname line parts name sig args isFile kind
       constring capitalKind signature sourceFileName emString heading page
       selectedOperation options conform)
  (declare (special |$Primitives|))
  (setq conform (car arg))
  (setq options (cdr arg))
  (setq bindingsAlist (ifcar options))
  (setq conname (|opOf| conform))
  (cond
   ((member conname |$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))
    (|htpSetProperty| page '|isFile| t)
    (|htpSetProperty| page '|fromConOpPage1| t)
    (|htpSetProperty| page '|parts| parts)
    (|htpSetProperty| page '|heading| heading)
    (|htpSetProperty| page '|kind| kind)
    (|htpSetProperty| page '|domname| domname)
    (|htpSetProperty| page '|conform| conform)
    (|htpSetProperty| page '|signature| signature)
    (when
     (setq selectedOperation (lassoc '|selectedOperation| (ifcdr options)))
      (|htpSetProperty| page '|selectedOperation| selectedOperation))
    (loop for item in bindingsAlist
     collect (|htpSetProperty| page (car item) (cdr item)))
    (|koPage| page "operation")))))
```

———-

55.3.3 defun dbCompositeWithMap

[htpProperty p1342]
[dbExtractUnderlyingDomain p1457]

— **defun dbCompositeWithMap** —

```
(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")))))))
```

———-

55.3.4 defun dbExtractUnderlyingDomain

[isValidType p??]

— **defun dbExtractUnderlyingDomain** —

```
(defun |dbExtractUnderlyingDomain| (domain)
 (some #'identity
  (loop for x in (ifcdr domain) when (|isValidType| x) collect x)))
```

———-

55.4 Operation Page from Main Page

55.4.1 defun koPage

[htpProperty p1342]
[concat p1107]
[koPageInputAreaUnchanged? p1450]
[kDomainName p1450]
[errorPage p??]
[form2HtString p??]
[capitalize p??]
[htpSetProperty p1342]
[koPageAux p1458]

— **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))))
  (cond
   ((and (consp domname) (eq (qcar domname) '|error|))
    (|errorPage| htPage domname))
   (t
    (|htpSetProperty| htPage '|domname| domname)
    (setq headingString (if domname (|form2HtString| domname nil t) constring))
```

```
    (setq heading (list (|capitalize| kind) " {\\sf " headingString "}" ))
    (|htpSetProperty| htPage '|which| which)
    (|htpSetProperty| htPage '|heading| heading)
    (|koPageAux| htPage which domname heading)))))
```

———-

55.4.2 defun koPageFromKKPage

[koPageAux p1458]
[htpProperty p1342]

— **defun koPageFromKKPage** —

```
(defun |koPageFromKKPage| (htPage ao)
  (|koPageAux| htPage ao (|htpProperty| htPage '|domname|)
      (|htpProperty| htPage '|heading|)))
```

———-

55.4.3 defun koPageAux

[htpSetProperty p1342]
[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)))
```

———-

55.4.4 defun koPageAux1

[htpProperty p1342]
[dbShowOperationsFromConform p??]

— defun koPageAux1 —

```
(defun |koPageAux1| (htPage opAlist)
 (let (which)
  (setq which (|htpProperty| htPage '|which|))
  (|dbShowOperationsFromConform| htPage which opAlist)))
```

———-

55.4.5 defun koaPageFilterByName

[htpLabelInputString p1342]
[koaPageFilterByCategory p??]
[pmTransFilter p??]
[htpProperty p1342]
[dbGetInputString p??]
[superMatch? p??]
[downcase p1140]
[htpSetProperty p1342]

— 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 (princ-to-string (car x))))
           collect x))
    (|htpSetProperty| htPage '|opAlist| opAlist)
    (funcall functionToCall htPage nil)))))
```

———-

55.5 Get Constructor Documentation

55.5.1 defun dbConstructorDoc,hn

[length p??]
[sublislis p??]

[$FormalMapVariableList p15]
[$sig p??]
[$args p??]

— defun dbConstructorDoc,hn —

```
(defun |dbConstructorDoc,hn| (sig)
 (declare (special |$sig| |$args|))
 (and (equal (|#| |$sig|) (|#| sig))
      (equal |$sig| (sublislis |$args| |$FormalMapVariableList| sig))))
```

————-

55.5.2 defun dbConstructorDoc,gn

[dbConstructorDoc,hn p1459]
[$op p??]

— defun dbConstructorDoc,gn —

```
(defun |dbConstructorDoc,gn| (arg)
 (let (op alist)
 (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) '(""))))))))
```

————-

55.5.3 defun dbConstructorDoc,fn

[dbConstructorDoc,gn p1460]
[getdatabase p1070]
[$args p??]

— defun dbConstructorDoc,fn —

```
(defun |dbConstructorDoc,fn| (conform)
 (declare (special |$args|))
  (let (conname docs)
   (setq conname (car conform))
   (setq |$args| (cdr conform))
   (loop for y in (getdatabase conname 'documentation)
    collect (|dbConstructorDoc,gn| y))))
```

————-

55.5.4 defun dbConstructorDoc

[dbConstructorDoc,fn p1460]
[$sig p??]
[$op p??]

— **defun dbConstructorDoc** —

```
(defun |dbConstructorDoc| (conform |$op| |$sig|)
  (declare (special |$op| |$sig|))
  (|dbConstructorDoc,fn| conform))
```

———-

55.5.5 defun dbDocTable

[hget p1105]
[make-hashtable p??]
[originsInOrder p1461]
[dbAddDocTable p1462]
[$docTable p??]
[$docTableHash p??]

— **defun dbDocTable** —

```
(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|))))
```

———-

55.5.6 defun originsInOrder

[getdatabase p1070]
[assocleft p??]
[ancestorsOf p??]
[parentsOf p??]
[originsInOrder p1461]
[insert p??]

— **defun originsInOrder** —

```
(defun |originsInOrder| (conform)
 (let (con argl acc)
  (setq con (car conform))
  (setq argl (cdr conform))
  (cond
   ((eq (getdatabase con 'constructorkind) '|category|)
     (assocleft (|ancestorsOf| conform nil)))
   (t
     (setq acc (assocleft (|parentsOf| con)))
     (loop for x in acc do
       (loop for y in (|originsInOrder| x) do
        (setq acc (|insert| y acc))))
     acc))))
```

———-

55.5.7 defun dbAddDocTable

[opOf p??]
[getConstructorForm p??]
[sublislis p??]
[getdatabase p1070]
[hput p1105]
[hget p1105]
[$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)))))))
```

———-

55.5.8 defun dbGetDocTable,hn

[sublislis p??]
[qcdr p??]
[qcar p??]
[$which p??]
[$conform p??]
[$sig p??]
[$FormalMapVariableList p15]

— defun dbGetDocTable,hn —

```
(defun |dbGetDocTable,hn| (arg)
 (let (sig doc alteredSig pred)
 (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
      (eql (|#| |$sig|) (|#| sig))
      (setq alteredSig
       (sublislis (ifcdr |$conform|) |$FormalMapVariableList| sig))
      (equal alteredSig |$sig|)))
    (when (and pred doc
           (and (consp doc) (eq (qcar doc) '|constant|)) (qcdr doc) doc)
      '(""))))))
```

———-

55.5.9 defun dbGetDocTable,gn

[lastatom p??]
[dbGetDocTable,hn p1463]
[$conform p??]

— defun dbGetDocTable,gn —

```
(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)))))
```

———-

55.5.10 defun dbGetDocTable

[string2Integer p??]
[dbConstructorDoc p1461]
[qcdr p??]
[hget p1105]
[dbGetDocTable,gn p1463]
[$sig p??]
[$which p??]
[$conform p??]
[$op p??]

— **defun dbGetDocTable** —

```
(defun |dbGetDocTable| (op |$sig| docTable |$which| aux)
 (declare (special |$sig| |$which|))
 (let (doc origin s)
 (declare (special |$conform| |$op|))
  (when (and (null (integerp op)) (digitp (elt (setq s (princ-to-string 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)))))))
```

———-

55.5.11 defun kTestPred

[testBitVector p??]
[simpHasPred p??]
[$predvec p??]
[$domain p??]

— **defun kTestPred** —

```
(defun |kTestPred| (n)
 (declare (special |$predvec| |$domain|))
 (cond
   ((eql n 0) t)
   (|$domain| (|testBitVector| |$predvec| n))
   (t (|simpHasPred| (elt |$predvec| (1- n))))))
```

———-

55.5.12 defun dbAddChainDomain

[dbInfovec p??]
[dbSubConform p1465]
[kFormatSlotDomain p??]
[devaluate p??]
[$infovec p??]

— **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)))))))
```

———-

55.5.13 defun dbSubConform

[position p??]
[dbSubConform p1465]
[$FormalMapVariableList p15]

— **defun dbSubConform** —

```
(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))))))
```

———-

55.5.14 defun dbAddChain

[dbAddChainDomain p1465]
[dbAddChain p1466]

— **defun dbAddChain** —

```
(defun |dbAddChain| (conform)
 (let (u)
  (when (setq u (|dbAddChainDomain| conform))
   (unless (atom u)
    (cons (cons u t) (|dbAddChain| u))))))
```

———-

55.6 Constructor Page Menu

55.6.1 defun dbShowCons

[htpProperty p1342]
[pmTransFilter p??]
[ifcar p??]
[dbGetInputString p??]
[bcErrorPage p??]
[constructor? p??]
[superMatch? p??]
[downcase p1140]
[emptySearchPage p??]
[htInitPageNoScroll p1349]
[htCopyProplist p??]
[htpSetProperty p1342]
[dbShowCons p1466]
[member p1108]
[dbShowCons1 p1467]
[$exposedOnlyIfTrue p??]

— **defun dbShowCons** —

```
(defun |dbShowCons| (&rest args)
 (let (cAlist filter abbrev? conname subject u options key htPage)
  (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 (princ-to-string subject))))
           collect x))
        (cond
         ((null u)
           (|emptySearchPage| "constructor" filter))
         (t
           (setq htPage (|htInitPageNoScroll| (|htCopyProplist| htPage)))
           (|htpSetProperty| htPage '|cAlist| u)
           (|dbShowCons| htPage (|htpProperty| htPage '|exclusion|)))))))
   (t
    (when (member key '(|exposureOn| |exposureOff|))
      (setq |$exposedOnlyIfTrue| (eq key '|exposureOn|))
      (setq key (|htpProperty| htPage '|exclusion|)))
    (|dbShowCons1| htPage cAlist key)))))
```

———-

55.6.2 defun conPageChoose

[getConstructorForm p??]
[dbShowCons1 p1467]

— **defun conPageChoose** —

```
(defun |conPageChoose| (conname)
 (let (cAlist)
  (setq cAlist (list (cons (|getConstructorForm| conname) t)))
  (|dbShowCons1| nil cAlist '|names|)))
```

———-

55.6.3 defun dbShowCons1

[remdup p??]
[isExposedConstructor p973]
[opOf p??]
[conPage p1428]
[htpProperty p1342]
[union p??]
[dbConstructorKind p??]
[htCopyProplist p??]
[htInitPageNoScroll p1349]
[dbConsHeading p1473]

[htSayStandard p1350]
[htpSetProperty p1342]
[bcNameConTable p??]
[bcAbbTable p??]
[getCDTEntry p??]
[getdatabase p1070]
[bcUnixTable p1474]
[listSort p??]
[function p??]
[qlesseqp p??]
[dbShowConsDoc p1470]
[isExposedConstructor p973]
[dbShowConditions p1472]
[bcConTable p??]
[assocleft p??]
[dbShowConsKinds p??]
[dbConsExposureMessage p1469]
[dbPresentCons p??]
[htShowPageNoScroll p1351]
[$conformsAreDomains p??]
[$exposedOnlyIfTrue p??]

— **defun dbShowCons1** —

```
(defun |dbShowCons1| (htPage cAlist key)
 (let (|$conformsAreDomains| conlist kinds kind proplist page u flist result)
 (declare (special |$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|
     (if (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 ")
  (|htpSetProperty| page '|cAlist| cAlist)
  (setq |$conformsAreDomains| (|htpProperty| page '|domname|))
  (cond
   ((eq key '|names|) (|bcNameConTable| conlist))
   ((eq key '|abbrs|)
    (|bcAbbTable|
     (loop for con in conlist collect (|getCDTEntry| con t))))
   ((eq key '|files|)
    (setq flist
     (loop for con in conlist collect (getdatabase con 'sourcefile)))
    (|bcUnixTable|
     (|listSort| #'glesseqp (remdup flist))))
   ((eq key '|documentation|) (|dbShowConsDoc| page conlist))
   (t
    (when |$exposedOnlyIfTrue|
     (setq cAlist
      (loop for x in cAlist
       when (|isExposedConstructor| (|opOf| (car x)))
       collect x)))
    (cond
     ((eq key '|conditions|) (|dbShowConditions| page cAlist kind))
     ((eq key '|parameters|)
       (|bcConTable| (remdup (assocleft cAlist))))
     ((eq key '|kinds|) (|dbShowConsKinds| cAlist)))))
  (|dbConsExposureMessage|)
  (|htSayStandard| '|\\endscroll |)
  (|dbPresentCons| page kind key)
  (|htShowPageNoScroll|)))))
```

———-

55.6.4 defun dbConsExposureMessage

[htSay p1347]
[$atLeastOneUnexposed p??]

— **defun dbConsExposureMessage** —

```
(defun |dbConsExposureMessage| ()
 (declare (special |$atLeastOneUnexposed|))
  (when |$atLeastOneUnexposed|
   (|htSay| "\\newline{}-------------\\newline{}{\\em *} = unexposed")))
```

———-

55.6.5 defun dbShowConsKindsFilter

[htpSetProperty p1342]
[dbShowCons p1466]
[htpProperty p1342]

— **defun dbShowConsKindsFilter** —

```
(defun |dbShowConsKindsFilter| (htPage args)
 (|htpSetProperty| htPage '|cAlist| (second args))
 (|dbShowCons| htPage (|htpProperty| htPage '|exclusion|)))
```

————-

55.6.6 defun dbShowConsDoc

[systemError p??]
[dbShowConsDoc1 p1470]
[getConstructorForm p??]
[opOf p??]
[htpProperty p1342]
[remdup p??]

— **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 (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)))))))
```

————-

55.6.7 defun dbShowConsDoc1

[member p1108]
[htpProperty p1342]

[getl p1110]
[displayDomainOp p??]
[isExposedConstructor p973]
[getConstructorDocumentation p1471]
[getConstructorSignature p??]
[getdatabase p1070]
[sublislis p??]
[sublisFormal p??]
[displayDomainOp 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 |$Primitives|)
    (setq conname (|htpProperty| htPage '|conname|))
    (setq lt1 (getl conname '|documentation|))
    (cond ((eq (caar lt1) '|constructor|) (caar lt1)))
    (cond ((eq (caadar lt1) 'nil) (caadar lt1)))
    (setq doc (car (cdadar lt1)))
    (setq sig '((category domain) (|SetCategory|) (|SetCategory|)))
    (|displayDomainOp| htPage "constructor"
       conform conname sig t doc indexOrNil '|dbSelectCon| nil nil))
   (t
    (setq exposeFlag (|isExposedConstructor| conname))
    (setq doc (list (|getConstructorDocumentation| conname)))
    (setq signature (|getConstructorSignature| conname))
    (setq sig
     (if (eq (getdatabase conname 'constructorkind) '|category|)
        (sublislis conargs |$TriangleVariableList| signature)
        (|sublisFormal| conargs signature)))
    (|displayDomainOp| htPage "constructor" conform conname sig t doc
      indexOrNil '|dbSelectCon| (null exposeFlag) nil)))))
```

———-

55.6.8 defun getConstructorDocumentation

[lassoc p??]
[getdatabase p1070]
[qcar p??]
[qcaar p??]
[qcdar 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 (qcdar IT1))
        (qcadar IT1))
   "")))
```

———-

55.6.9 defun dbSelectCon

[conPage p1428]
[opOf p??]
[htpProperty p1342]

— **defun dbSelectCon** —

```
(defun |dbSelectCon| (htPage which index)
 (declare (ignore which))
 (|conPage| (|opOf| (car (elt (|htpProperty| htPage '|cAlist|) index)))))
```

———-

55.6.10 defun dbShowConditions

[htpProperty p1342]
[opOf p??]
[splitConTable p??]
[pluralize p??]
[length p??]
[dbSayItems p??]
[bcConPredTable p??]
[htSayHrule p??]

— **defun dbShowConditions** —

```
(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 (|splitConTable| cAlist))
  (setq consNoPred (car lt1))
  (setq consPred (cdr lt1))
  (setq singular (list kind " is"))
  (setq plural (list (|pluralize| (princ-to-string kind)) " are"))
  (|dbSayItems| (|#| consNoPred) singular plural " unconditional")
```

```
  (|bcConPredTable| consNoPred conname)
  (|htSayHrule|)
  (|dbSayItems| (|#| consPred) singular plural " conditional")
  (|bcConPredTable| consPred conname)))
```

———-

55.6.11 defun dbConsHeading

[htpProperty p1342]
[length p??]
[remdup p??]
[form2HtString p??]
[capitalize p??]
[pluralize p??]
[member p1108]
[nequal p??]
[$exposedOnlyIfTrue p??]

— **defun dbConsHeading** —

```
(defun |dbConsHeading| (htPage conlist view kind)
 (let (thing place count rank modifier exposureWord firstWord prefix
       placepart connective heading)
(declare (special |$exposedOnlyIfTrue|))
 (setq thing (or (and htPage (|htpProperty| htPage '|thing|)) "constructor"))
 (setq place
  (when htPage
   (or (|htpProperty| htPage '|domname|) (|htpProperty| htPage '|conform|))))
 (setq count (|#| (remdup conlist)))
 (cond
  ((string= thing "benefactor")
    (list (princ-to-string count) " Constructors Used by "
          (|form2HtString| place nil t) ))
  (t
   (setq modifier
    (cond
     ((string= thing "argument")
       (setq rank (and htPage (|htpProperty| htPage '|rank|)))
       (list " Possible " rank " "))
     ((eq kind '|constructor|)
       (list " "))
     (t
       (cons " " (|capitalize| (princ-to-string kind)) " "))))
   (setq exposureWord (when |$exposedOnlyIfTrue| '(" Exposed ")))
   (setq prefix
    (cond
     ((eql count 1)
      (cons (princ-to-string count)
       (append modifier (list (|capitalize| thing)))))
     (t
      (setq firstWord (if (eql count 0) "No "(princ-to-string 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 " (princ-to-string view) "}")))))
   heading))))
```

———-

55.6.12 defun dbShowConstructorLines

[getConstructorForm p??]
[intern p??]
[dbName p??]
[dbShowCons1 p1467]
[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| #'glesseqp cAlist) '|names|)))
```

———-

55.6.13 defun bcUnixTable

[htSay p1347]
[htBeginTable p??]
[namestring p1102]
[findfile p??]
[htMakePage p1351]
[htEndTable p??]
[firstTime p??]

— defun bcUnixTable —

```
(defun |bcUnixTable| (u)
 (declare (special firstTime))
 (let (filename)
  (|htSay| "\\newline")
  (|htBeginTable|)
  (setq firstTime t)
  (loop for x in u do
    (|htSay| "{")
    (setq filename (namestring ($findfile (princ-to-string x) "SPAD")))
    (|htMakePage|
     (list
       (list '|text| "\\unixcommand{" (pathname-name (string x))
             "}{$AXIOM/lib/SPADEDIT " filename "} )))
    (|htSay| "}"))
  (|htEndTable|)))
```

———-

55.6.14 Special Code for Union, Mapping, and Record

55.6.15 defun dbSpecialDescription

[getConstructorForm p??]
[form2HtString p??]
[htInitPage p1349]
[htpSetProperty p1342]
[dbShowConsDoc1 p1470]
[htShowPage p1350]
[$conformsAreDomains p??]

— defun dbSpecialDescription —

```
(defun |dbSpecialDescription| (conname)
 (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|)))
```

———-

55.6.16 defun dbSpecialOperations

[htInitPage p1349]
[getConstructorForm p??]
[dbSpecialExpandIfNecessary p1477]
[getl p1110]
[form2HtString p??]
[htpSetProperty p1342]
[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|)))
```

———-

55.6.17 defun dbSpecialExports

[getConstructorForm p??]
[htInitPage p1349]
[form2HtString p??]
[dbSpecialExpandIfNecessary p1477]
[getl p1110]
[kePageDisplay p1436]
[htShowPage p1350]

— **defun dbSpecialExports** —

```
(defun |dbSpecialExports| (conname)
 (let (conform page opAlist)
  (setq conform (|getConstructorForm| conname))
  (setq page
   (|htInitPage| (list "Exports of {\\sf " (|form2HtString| conform) "}") nil))
  (setq opAlist
   (|dbSpecialExpandIfNecessary| conform
    (cdr (getl conname '|documentation|))))
  (|kePageDisplay| page "operation" opAlist)
  (|htShowPage|)))
```

———-

55.6.18 defun dbSpecialExpandIfNecessary

[qcar p??]
[qcdar p??]
[qcadar p??]
[qcdr p??]

— **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|) $)
 "\\spad{coerce(r)} returns an representation of \\spad{r} as an output form")
    (($ (|List| (|Any|)))
   ,(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]}")))
```

```
    (|elt| ((A $ "a")
  ,(concatenate 'string
    "\\spad{r . a} returns the value stored in record \\spad{r} under "
    "selector \\spad{a}."))
    ((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."))
    ((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 {\\em A} or of domain {\\em B}. The {\\sf Union} "
"constructor can take any number of arguments. "
"For an alternate form of {\\sf Union} with \"tags\", see "
"\\downlink{Union(a:A,b:B)}{DomainUnion}. {\\sf 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
"\\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."))
    ((|Boolean|) $ "B")
  ,(concatenate 'string
"\\spad{u case B} tests if \\spad{u} is of the \\spad{B} branch "
"of the union.")))
```

```
    (|coerce| ((A $)
  ,(concatenate 'string
"\\spad{coerce(u)} returns \\spad{x} of type \\spad{A} if "
"\\spad{x} is of the \\spad{A} branch of the union. "
"Error: if \\spad{u} is of the \\spad{B} branch of the union."))
    ((B $)
  ,(concatenate 'string
"\\spad{coerce(u)} returns \\spad{x} of type \\spad{B} if "
"\\spad{x} is of the \\spad{B} branch of the union. "
"Error: if \\spad{u} is of the \\spad{A} branch of the union."))
    (($ 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 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."))
```

———-

— 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|) $ $)
  ,(concatenate 'string
"\\spad{e = f} tests for equality of two enumerations \\spad{e} "
"and \\spad{f}")))
   (^= ((|Boolean|) $ $)
  ,(concatenate 'string
"\\spad{e ^= f} tests that two enumerations \\spad{e} and "
"\\spad{f} are not equal")))
   (|coerce| (((|OutputForm|) $)
  ,(concatenate 'string
"\\spad{coerce(e)} returns a representation of enumeration "
"\\spad{r} as an output form"))
   (($ (|Symbol|))
  ,(concatenate 'string
"\\spad{coerce(s)} converts a symbol \\spad{s} into an "
"enumeration which has \\spad{s} as a member symbol"))))
 :test #'equal)))
```

———-

55.6.19 defun lefts

[hkeys p1105]
[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)))
```

———-

55.6.20 Build Library Database (libdb.text,...)

55.6.21 defun dbMkForm

— defun dbMkForm —

```
(defun |dbMkForm| (x)
 (or (and (atom x) (cons x nil)) x))
```

————-

55.6.22 defun libConstructorSig

[getdatabase p1070]
[take p**??**]
[length p**??**]
[sublislis p**??**]
[form2LispString p**??**]
[ncParseFromString p1172]
[sayBrightly p**??**]
[$TriangleVariableList 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))))
(setq sigpart (|form2LispString| (cons '|Mapping| sig)))
(unless (|ncParseFromString| sigpart)
  (|sayBrightly| (list "Won't parse: " sigpart)))
sigpart)))
```

———-

Chapter 56

Utility functions

56.1 Utility functions

56.1.1 defun Delete an alist pair given the key

— **defun delasc 0** —

```
(defun delasc (key alist)
 (remove key alist :key #'car))
```

———-

56.1.2 defun readline

— **defun readline** —

```
(defun readline (t1)
 (if t1
  (|read-line| t1)
  (|read-line| *STANDARD-INPUT*)))
```

———-

56.1.3 defun isWrapped

This was proven by ACL2 to accept any input and return either T or NIL. Note that ACL2 does not support FLOATP.

— **defun isWrapped 0 ACL2** —

```
(defun isWrapped (x)
  (or (and (consp x) (eq (car x) 'wrapped))
      (acl2-numberp x)
      (stringp x)))
```

```
;==>
;(OR (EQUAL (ISWRAPPED X) T) (EQUAL (ISWRAPPED X) NIL))
```

---

isWrapped : t → (or t nil)
— defun isWrapped :proven —

```
(defun |isWrapped| (x)
  (or (and (consp x) (eq (qcar x) 'wrapped))
      (numberp x)
      (floatp x)
      (stringp x)))
```

Chapter 57

The Proofs

— **acl2** —

```
\getchunk{defun isWrapped 0 ACL2}
```

————-

Chapter 58

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 cdancols 0}
\getchunk{defmacro cdanrows 0}
\getchunk{defmacro cdaref2 0}
\getchunk{defmacro cdelt 0}
\getchunk{defmacro cdlen 0}
\getchunk{defmacro cdsetaref2 0}
\getchunk{defmacro cdsetelt 0}
\getchunk{defmacro dancols 0}
\getchunk{defmacro danrows 0}
\getchunk{defmacro daref2 0}
\getchunk{defmacro delt 0}
\getchunk{defmacro DFAcos 0}
\getchunk{defmacro DFAcosh 0}
\getchunk{defmacro DFAdd 0}
\getchunk{defmacro DFAsin 0}
\getchunk{defmacro DFAsinh 0}
\getchunk{defmacro DFAtan 0}
\getchunk{defmacro DFAtan2 0}
\getchunk{defmacro DFAtanh 0}
\getchunk{defmacro DFCos 0}
\getchunk{defmacro DFCosh 0}
\getchunk{defmacro DFDivide 0}
\getchunk{defmacro DFEql 0}
```

```
\getchunk{defmacro DFExp 0}
\getchunk{defmacro DFExpt 0}
\getchunk{defmacro DFIntegerDivide 0}
\getchunk{defmacro DFIntegerExpt 0}
\getchunk{defmacro DFIntegerMultiply 0}
\getchunk{defmacro DFLessThan 0}
\getchunk{defmacro DFLog 0}
\getchunk{defmacro DFLogE 0}
\getchunk{defmacro DFMax 0}
\getchunk{defmacro DFMin 0}
\getchunk{defmacro DFMinusp 0}
\getchunk{defmacro DFMultiply 0}
\getchunk{defmacro DFSin 0}
\getchunk{defmacro DFSinh 0}
\getchunk{defmacro DFSqrt 0}
\getchunk{defmacro DFSubtract 0}
\getchunk{defmacro DFTan 0}
\getchunk{defmacro DFTanh 0}
\getchunk{defmacro DFUnaryMinus 0}
\getchunk{defmacro DFZerop 0}
\getchunk{defmacro dlen 0}
\getchunk{defmacro dsetaref2 0}
\getchunk{defmacro dsetelt 0}
\getchunk{defmacro idChar? 0}
\getchunk{defmacro identp 0}
\getchunk{defmacro FloatError 0}
\getchunk{defmacro fracpart 0}
\getchunk{defmacro frameExposureData 0}
\getchunk{defmacro frameHiFiAccess 0}
\getchunk{defmacro frameHistListAct 0}
\getchunk{defmacro frameHistList 0}
\getchunk{defmacro frameHistListLen 0}
\getchunk{defmacro frameHistoryTable 0}
\getchunk{defmacro frameHistRecord 0}
\getchunk{defmacro frameInteractive 0}
\getchunk{defmacro frameIOIndex 0}
\getchunk{defmacro frameName 0}
\getchunk{defmacro frameNames 0}
\getchunk{defmacro getMsgArgL 0}
\getchunk{defmacro getMsgKey 0}
\getchunk{defmacro getMsgPosTagOb 0}
\getchunk{defmacro getMsgPrefix 0}
\getchunk{defmacro getMsgPrefix? 0}
\getchunk{defmacro getMsgTag 0}
\getchunk{defmacro getMsgTag? 0}
\getchunk{defmacro getMsgText 0}
\getchunk{defmacro hashCode? 0}
\getchunk{defmacro line-clear 0}
\getchunk{defmacro make-cdouble-matrix 0}
\getchunk{defmacro make-cdouble-vector 0}
\getchunk{defmacro make-double-matrix 0}
\getchunk{defmacro make-double-matrix1 0}
\getchunk{defmacro make-double-vector 0}
\getchunk{defmacro make-double-vector1 0}
```

```
\getchunk{defmacro qcsize 0}
\getchunk{defmacro qsabsval 0}
\getchunk{defmacro qsadd1 0}
\getchunk{defmacro qsdifference 0}
\getchunk{defmacro qsgreaterp 0}
\getchunk{defmacro qslessp 0}
\getchunk{defmacro qsmax 0}
\getchunk{defmacro qsmin 0}
\getchunk{defmacro qsminus 0}
\getchunk{defmacro qsoddp 0}
\getchunk{defmacro qsplus 0}
\getchunk{defmacro qssub1 0}
\getchunk{defmacro qstimes 0}
\getchunk{defmacro qszerop 0}
\getchunk{defmacro setMsgPrefix 0}
\getchunk{defmacro setMsgText 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 eltU8}
\getchunk{defmacro eltU16}
\getchunk{defmacro eltU32}
\getchunk{defmacro funfind}
\getchunk{defmacro hget}
\getchunk{defmacro leader?}
\getchunk{defmacro line?}
\getchunk{defmacro makeMatrixU8}
\getchunk{defmacro makeMatrix1U8}
\getchunk{defmacro makeMatrixU16}
\getchunk{defmacro makeMatrix1U16}
\getchunk{defmacro makeMatrixU32}
\getchunk{defmacro makeMatrix1U32}
\getchunk{defmacro mkObj}
\getchunk{defmacro mkObjCode}
\getchunk{defmacro mkObjWrap}
\getchunk{defmacro objCodeVal}
\getchunk{defmacro objCodeMode}
\getchunk{defmacro objMode}
\getchunk{defmacro objSetMode}
\getchunk{defmacro objSetVal}
\getchunk{defmacro objVal}
```

```
\getchunk{defmacro objValUnwrap}
\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 toScreen?}
\getchunk{defmacro trapNumericErrors}
\getchunk{defmacro truth-to-bit}
\getchunk{defmacro while}
\getchunk{defmacro whileWithResult}

;;; layer 0 (all common lisp)

\getchunk{defun acot 0}
\getchunk{defun acoth 0}
\getchunk{defun acsc 0}
\getchunk{defun acsch 0}
\getchunk{defun asec 0}
\getchunk{defun asech 0}
\getchunk{defun axiomVersion 0}

\getchunk{defun basicStringize 0}
\getchunk{defun BesselasymptA 0}
\getchunk{defun BesselasymptB 0}
\getchunk{defun BesselIBackRecur 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 cgammaAdjust 0}
\getchunk{defun cgammaBernsum 0}
```

```
\getchunk{defun cgammaG 0}
\getchunk{defun cgammat 0}
\getchunk{defun chebf01coefmake 0}
\getchunk{defun chebstarevalarr 0}
\getchunk{defun cleanupLine 0}
\getchunk{defun clearMacroTable 0}
\getchunk{defun concat 0}
\getchunk{defun concatWithBlanks 0}
\getchunk{defun cot 0}
\getchunk{defun cotdiffeval 0}
\getchunk{defun coth 0}
\getchunk{defun createCurrentInterpreterFrame 0}
\getchunk{defun credits 0}
\getchunk{defun csc 0}
\getchunk{defun csch 0}
\getchunk{defun c-to-s 0}

\getchunk{defun dbKind 0}
\getchunk{defun dbRead 0}
\getchunk{defun delasc 0}
\getchunk{defun Delay 0}
\getchunk{defun desiredMsg 0}
\getchunk{defun DirToString 0}
\getchunk{defun displayFrameNames 0}
\getchunk{defun divide2 0}
\getchunk{defun dqAppend 0}
\getchunk{defun dqToList 0}
\getchunk{defun dqUnit 0}

\getchunk{defun embed2 0}
\getchunk{defun emptyInterpreterFrame 0}
\getchunk{defun endedp 0}
\getchunk{defun evalSharpOne 0}

\getchunk{defun fin 0}
\getchunk{defun findFrameInRing 0}
\getchunk{defun flatten 0}
\getchunk{defun flattenOperationAlist 0}
\getchunk{defun fnameExists? 0}
\getchunk{defun fnameName 0}
\getchunk{defun fnameReadable? 0}
\getchunk{defun fnameType 0}
\getchunk{defun frameNames 0}
\getchunk{defun From 0}
\getchunk{defun FromTo 0}

\getchunk{defun get-a-line 0}
\getchunk{defun get-current-directory 0}
\getchunk{defun getenviron 0}
\getchunk{defun getl 0}
\getchunk{defun getLinePos 0}
\getchunk{defun getLineText 0}
\getchunk{defun getMsgKey? 0}
\getchunk{defun getParserMacroNames 0}
```

```
\getchunk{defun getPreStL 0}
\getchunk{defun getspoolname 0}

\getchunk{defun hasCorrectTarget 0}
\getchunk{defun hasOptArgs? 0}
\getchunk{defun horner 0}
\getchunk{defun htpAddToPageDescription 0}
\getchunk{defun htpMakeEmptyPage 0}
\getchunk{defun htSayStandard 0}

\getchunk{defun ignorep 0}
\getchunk{defun incActive? 0}
\getchunk{defun incCommand? 0}
\getchunk{defun incDrop 0}
\getchunk{defun incHandleMessage 0}
\getchunk{defun inclmsgConsole 0}
\getchunk{defun inclmsgFinSkipped 0}
\getchunk{defun inclmsgPrematureEOF 0}
\getchunk{defun inclmsgCmdBug 0}
\getchunk{defun inclmsgIfBug 0}
\getchunk{defun incPrefix? 0}
\getchunk{defun initial-substring 0}
\getchunk{defun init-memory-config 0}
\getchunk{defun insertPos 0}
\getchunk{defun integer-decode-float-denominator 0}
\getchunk{defun integer-decode-float-exponent 0}
\getchunk{defun integer-decode-float-sign 0}
\getchunk{defun integer-decode-float-numerator 0}
\getchunk{defun intloopPrefix? 0}
\getchunk{defun isIntegerString 0}
\getchunk{defun isWrapped :proven}

\getchunk{defun keyword 0}
\getchunk{defun keyword? 0}

\getchunk{defun lastcount 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 libdbTrim 0}
\getchunk{defun limitedPrint1 0}
\getchunk{defun line-advance-char 0}
\getchunk{defun line-at-end-p 0}
\getchunk{defun line-current-segment 0}
\getchunk{defun line-new-line 0}
\getchunk{defun line-next-char 0}
\getchunk{defun line-past-end-p 0}
\getchunk{defun line-print 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 logH 0}
\getchunk{defun logS 0}

\getchunk{defun macODefine 0}
\getchunk{defun macOInfiniteExpansion,name 0}
\getchunk{defun make-absolute-filename 0}
\getchunk{defun makeByteWordVec2 0}
\getchunk{defun makeInitialModemapFrame 0}
\getchunk{defun manexp 0}
\getchunk{defun markUnique 0}
\getchunk{defun member 0}
\getchunk{defun mkObjFn 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 next-line 0}
\getchunk{defun nonBlank 0}
\getchunk{defun npAnyNo 0}
\getchunk{defun npboot 0}
\getchunk{defun npEqPeek 0}
\getchunk{defun nplisp 0}
\getchunk{defun npPop1 0}
\getchunk{defun npPop2 0}
\getchunk{defun npPop3 0}
\getchunk{defun npPush 0}

\getchunk{defun objEnv 0}
\getchunk{defun objModeFn 0}
\getchunk{defun objValFn 0}
\getchunk{defun opTran 0}

\getchunk{defun pfAndLeft 0}
\getchunk{defun pfAndRight 0}
\getchunk{defun pfAppend 0}
\getchunk{defun pfApplicationArg 0}
\getchunk{defun pfApplicationOp 0}
\getchunk{defun pfAssignLhsItems 0}
\getchunk{defun pf0AssignLhsItems 0}
\getchunk{defun pfAssignRhs 0}
\getchunk{defun pfBreakFrom 0}
\getchunk{defun pfCoercetoExpr 0}
\getchunk{defun pfCoercetoType 0}
\getchunk{defun pfCollectBody 0}
\getchunk{defun pfCollectIterators 0}
\getchunk{defun pfDefinitionLhsItems 0}
\getchunk{defun pfDefinitionRhs 0}
\getchunk{defun pfDoBody 0}
\getchunk{defun pfExitCond 0}
\getchunk{defun pfExitExpr 0}
\getchunk{defun pfFirst 0}
\getchunk{defun pfFreeItems 0}
\getchunk{defun pfForinLhs 0}
\getchunk{defun pfForinWhole 0}
\getchunk{defun pfFromdomDomain 0}
\getchunk{defun pfFromdomWhat 0}
\getchunk{defun pfIfCond 0}
\getchunk{defun pfIfElse 0}
\getchunk{defun pfIfThen 0}
\getchunk{defun pfLambdaArgs 0}
\getchunk{defun pfLambdaBody 0}
\getchunk{defun pfLambdaRets 0}
\getchunk{defun pfLiteral? 0}
\getchunk{defun pfLocalItems 0}
\getchunk{defun pfLoopIterators 0}
\getchunk{defun pfMacroLhs 0}
\getchunk{defun pfMacroRhs 0}
\getchunk{defun pfMLambdaArgs 0}
\getchunk{defun pfMLambdaBody 0}
```

```
\getchunk{defun pfNotArg 0}
\getchunk{defun pfNovalueExpr 0}
\getchunk{defun pfOrLeft 0}
\getchunk{defun pfOrRight 0}
\getchunk{defun pfParts 0}
\getchunk{defun pfPile 0}
\getchunk{defun pfPretendExpr 0}
\getchunk{defun pfPretendType 0}
\getchunk{defun pfRestrictExpr 0}
\getchunk{defun pfRestrictType 0}
\getchunk{defun pfReturnExpr 0}
\getchunk{defun pfRuleLhsItems 0}
\getchunk{defun pfRuleRhs 0}
\getchunk{defun pfSecond 0}
\getchunk{defun pfSequenceArgs 0}
\getchunk{defun pfSuchthatCond 0}
\getchunk{defun pfTaggedExpr 0}
\getchunk{defun pfTaggedTag 0}
\getchunk{defun pfTree 0}
\getchunk{defun pfTypedId 0}
\getchunk{defun pfTypedType 0}
\getchunk{defun pfTupleParts 0}
\getchunk{defun pfWhereContext 0}
\getchunk{defun pfWhereExpr 0}
\getchunk{defun pfWhileCond 0}
\getchunk{defun placep 0}
\getchunk{defun pmDontQuote? 0}
\getchunk{defun pname 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 PsiAsymptoticOrder 0}
\getchunk{defun PsiEps 0}
\getchunk{defun PsiIntpart 0}

\getchunk{defun qenum 0}
\getchunk{defun qeset 0}
\getchunk{defun qsquotient 0}
\getchunk{defun qsremainder 0}
\getchunk{defun quotient2 0}

\getchunk{defun random 0}
\getchunk{defun rdigit? 0}
\getchunk{defun reclaim 0}
\getchunk{defun remainder2 0}
\getchunk{defun remLine 0}
\getchunk{defun removeOption 0}
\getchunk{defun rep 0}
\getchunk{defun resetStackLimits 0}
\getchunk{defun resultp 0}
```

```
\getchunk{defun sameUnionBranch 0}
\getchunk{defun satisfiesUserLevel 0}
\getchunk{defun scanCloser? 0}
\getchunk{defun sec 0}
\getchunk{defun sech 0}
\getchunk{defun setCurrentLine 0}
\getchunk{defun set-restart-hook 0}
\getchunk{defun showMsgPos? 0}
\getchunk{defun smallEnoughCount 0}
\getchunk{defun startsComment? 0}
\getchunk{defun storeblanks 0}
\getchunk{defun s-to-c 0}
\getchunk{defun StreamNull 0}
\getchunk{defun stringize 0}
\getchunk{defun stringPrefix? 0}
\getchunk{defun stripLisp 0}
\getchunk{defun stripSpaces 0}
\getchunk{defun substring 0}

\getchunk{defun theid 0}
\getchunk{defun thefname 0}
\getchunk{defun theorigin 0}
\getchunk{defun tokPart 0}
\getchunk{defun To 0}
\getchunk{defun Top? 0}
\getchunk{defun trademark 0}

\getchunk{defun /untrace-reduce 0}

\getchunk{defun vec2list 0}
\getchunk{defun vmread 0}

\getchunk{defun zeroOneTran 0}

;;; above level 0

\getchunk{defun abbQuery}
\getchunk{defun abbreviations}
\getchunk{defun abbreviationsSpad2Cmd}
\getchunk{defun absolutelyCanCoerceByCheating}
\getchunk{defun addBinding}
\getchunk{defun addBindingInteractive}
\getchunk{defun addInputLibrary}
\getchunk{defun addNewInterpreterFrame}
\getchunk{defun addoperations}
\getchunk{defun addTraceItem}
\getchunk{defun Advance-Char}
\getchunk{defun algCoerceInteractive}
\getchunk{defun algEqual}
\getchunk{defun allConstructors}
\getchunk{defun allOperations}
\getchunk{defun alqlGetOrigin}
\getchunk{defun alqlGetParams}
\getchunk{defun alqlGetKindString}
```

```
\getchunk{defun alreadyOpened?}
\getchunk{defun apropos}
\getchunk{defun assertCond}
\getchunk{defun augmentHasArgs}
\getchunk{defun augmentTraceNames}

\getchunk{defun basicLookup}
\getchunk{defun basicLookupCheckDefaults}
\getchunk{defun bcComplexLimit}
\getchunk{defun bcComplexLimitGen}
\getchunk{defun bcCreateVariableString}
\getchunk{defun bcDefiniteIntegrate}
\getchunk{defun bcDefiniteIntegrateGen}
\getchunk{defun bcDifferentiate}
\getchunk{defun bcDifferentiateGen}
\getchunk{defun bcDraw}
\getchunk{defun bcDrawIt}
\getchunk{defun bcDrawIt2}
\getchunk{defun bcDraw2Dfun}
\getchunk{defun bcDraw2DfunGen}
\getchunk{defun bcDraw2Dpar}
\getchunk{defun bcDraw2DparGen}
\getchunk{defun bcDraw2DSolve}
\getchunk{defun bcDraw2DSolveGen}
\getchunk{defun bcDraw3Dfun}
\getchunk{defun bcDraw3DfunGen}
\getchunk{defun bcDraw3Dpar}
\getchunk{defun bcDraw3DparGen}
\getchunk{defun bcDraw3Dpar1}
\getchunk{defun bcDraw3Dpar1Gen}
\getchunk{defun bcError}
\getchunk{defun bcFindString}
\getchunk{defun bcFinish}
\getchunk{defun bcGen}
\getchunk{defun bcGenEquations}
\getchunk{defun bcGenExplicitMatrix}
\getchunk{defun bcHt}
\getchunk{defun bchtMakeButton}
\getchunk{defun bcIndefiniteIntegrate}
\getchunk{defun bcIndefiniteIntegrateGen}
\getchunk{defun bcInputEquations}
\getchunk{defun bcInputEquationsEnd}
\getchunk{defun bcInputExplicitMatrix}
\getchunk{defun bcInputMatrixByFormula}
\getchunk{defun bcInputMatrixByFormulaGen}
\getchunk{defun bcInputSolveInfo}
\getchunk{defun bcIssueHt}
\getchunk{defun bcLaurentSeries}
\getchunk{defun bcLaurentSeriesGen}
\getchunk{defun bcLimit}
\getchunk{defun bcLinearExtractMatrix}
\getchunk{defun bcLinearMatrixGen}
\getchunk{defun bcLinearSolve}
\getchunk{defun bcLinearSolveEqns}
```

```
\getchunk{defun bcLinearSolveEqns1}
\getchunk{defun bcLinearSolveEqnsGen}
\getchunk{defun bcLinearSolveMatrix}
\getchunk{defun bcLinearSolveMatrix1}
\getchunk{defun bcLinearSolveMatrixHomo}
\getchunk{defun bcLinearSolveMatrixInhomo}
\getchunk{defun bcLinearSolveMatrixInhomoGen}
\getchunk{defun bcMatrix}
\getchunk{defun bcMatrixGen}
\getchunk{defun bcMakeEquations}
\getchunk{defun bcMakeLinearEquations}
\getchunk{defun bcMakeUnknowns}
\getchunk{defun bcMkFunction}
\getchunk{defun bcNotReady}
\getchunk{defun bcOptional}
\getchunk{defun bcProduct}
\getchunk{defun bcProductGen}
\getchunk{defun bcPuiseuxSeries}
\getchunk{defun bcPuiseuxSeriesGen}
\getchunk{defun bcReadMatrix}
\getchunk{defun bcRealLimit}
\getchunk{defun bcRealLimitGen}
\getchunk{defun bcRealLimitGen1}
\getchunk{defun bcSadFaces}
\getchunk{defun bcSeries}
\getchunk{defun bcSeriesByFormula}
\getchunk{defun bcSeriesByFormulaGen}
\getchunk{defun bcSeriesExpansion}
\getchunk{defun bcSeriesExpansionGen}
\getchunk{defun bcSeriesGen}
\getchunk{defun bcSolve}
\getchunk{defun bcSolveEquations}
\getchunk{defun bcSolveEquationsNumerically}
\getchunk{defun bcSolveNumerically1}
\getchunk{defun bcSolveSingle}
\getchunk{defun bcString2HyString}
\getchunk{defun bcString2HyString2}
\getchunk{defun bcString2WordList}
\getchunk{defun bcSystemSolveEqns1}
\getchunk{defun bcSum}
\getchunk{defun bcSumGen}
\getchunk{defun bcSystemSolve}
\getchunk{defun bcTaylorSeries}
\getchunk{defun bcTaylorSeriesGen}
\getchunk{defun bcUnixTable}
\getchunk{defun bcVectorGen}
\getchunk{defun bcvspace}
\getchunk{defun bcwords2liststring}
\getchunk{defun beforeAfter}
\getchunk{defun BesselI}
\getchunk{defun besselIback}
\getchunk{defun besselIcheb}
\getchunk{defun BesselJ}
\getchunk{defun BesselJAsympt}
```

```
\getchunk{defun BesselJAsymptOrder}
\getchunk{defun BesselJRecur}
\getchunk{defun bpitrace}
\getchunk{defun bracketString}
\getchunk{defun break}
\getchunk{defun breaklet}
\getchunk{defun brightprint}
\getchunk{defun brightprint-0}
\getchunk{defun browse}
\getchunk{defun browseopen}
\getchunk{defun buildHtMacroTable}
\getchunk{defun buttonNames}

\getchunk{defun canFuncall?}
\getchunk{defun categoryopen}
\getchunk{defun catchCoerceFailure}
\getchunk{defun cbesseli}
\getchunk{defun cbesselj}
\getchunk{defun cgamma}
\getchunk{defun cgammaImpl}
\getchunk{defun changeHistListLen}
\getchunk{defun changeToNamedInterpreterFrame}
\getchunk{defun charDigitVal}
\getchunk{defun chebf01}
\getchunk{defun checkCondition}
\getchunk{defun checkFilter}
\getchunk{defun chkAllNonNegativeInteger}
\getchunk{defun chkDirectory}
\getchunk{defun chkNameList}
\getchunk{defun chkNonNegativeInteger}
\getchunk{defun chkOutputFileName}
\getchunk{defun chkPosInteger}
\getchunk{defun chkRange}
\getchunk{defun chyper0f1}
\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 clngamma}
\getchunk{defun clngammacase1}
\getchunk{defun clngammacase2}
\getchunk{defun clngammacase3}
\getchunk{defun clngammacase23}
\getchunk{defun clngammaImpl}
\getchunk{defun close}
\getchunk{defun closeInterpreterFrame}
\getchunk{defun cmpnote}
\getchunk{defun coerceBranch2Union}
```

```
\getchunk{defun coerceByFunction}
\getchunk{defun coerceByTable}
\getchunk{defun coerceCommuteTest}
\getchunk{defun coerceConvertMmSelection}
\getchunk{defun coerceImmediateSubDomain}
\getchunk{defun coerceInt}
\getchunk{defun coerceInt0}
\getchunk{defun coerceInt1}
\getchunk{defun coerceIntX}
\getchunk{defun coerceIntAlgebraicConstant}
\getchunk{defun coerceIntByMap}
\getchunk{defun coerceIntByMapInner}
\getchunk{defun coerceIntCommute}
\getchunk{defun coerceInteractive}
\getchunk{defun coerceIntFromUnion}
\getchunk{defun coerceIntPermute}
\getchunk{defun coerceIntSpecial}
\getchunk{defun coerceIntTableOrFunction}
\getchunk{defun coerceIntTest}
\getchunk{defun coerceIntTower}
\getchunk{defun coerceInt2Union}
\getchunk{defun coerceOrRetract}
\getchunk{defun coerceOrThrowFailure}
\getchunk{defun coerceRetract}
\getchunk{defun coerceSpadArgs2E}
\getchunk{defun coerceSpadFunValue2E}
\getchunk{defun coerceSubDomain}
\getchunk{defun coerceTraceArgs2E}
\getchunk{defun coerceTraceFunValue2E}
\getchunk{defun coerceUnion2Branch}
\getchunk{defun coercionFailure}
\getchunk{defun commandAmbiguityError}
\getchunk{defun commandError}
\getchunk{defun commandErrorIfAmbiguous}
\getchunk{defun commandErrorMessage}
\getchunk{defun commandsForUserLevel}
\getchunk{defun commandUserLevelError}
\getchunk{defun compareposns}
\getchunk{defun compareTypeLists}
\getchunk{defun compileBoot}
\getchunk{defun compiledLookup}
\getchunk{defun compiledLookupCheck}
\getchunk{defun computeDomainVariableAlist}
\getchunk{defun computeTTTranspositions}
\getchunk{defun condErrorMsg}
\getchunk{defun conLowerCaseConTran}
\getchunk{defun conOpPage}
\getchunk{defun conOpPage1}
\getchunk{defun conPage}
\getchunk{defun conPageChoose}
\getchunk{defun conPageConEntry}
\getchunk{defun conPageFastPath}
\getchunk{defun conSpecialString?}
\getchunk{defun constoken}
```

```
\getchunk{defun constructorSearch}
\getchunk{defun constructSubst}
\getchunk{defun containsVars}
\getchunk{defun containsVars1}
\getchunk{defun copyright}
\getchunk{defun countCache}
\getchunk{defun cpsi}
\getchunk{defun cPsiImpl}
\getchunk{defun cSearch}
\getchunk{defun c-to-r}

\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,fn}
\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 dbMkEvalable}
\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 dbString2Words}
\getchunk{defun dbSubConform}
\getchunk{defun dbWordFrom}
\getchunk{defun decideHowMuch}
\getchunk{defun decomposeTypeIntoTower}
\getchunk{defun defaultTargetFE}
\getchunk{defun defiostream}
\getchunk{defun deldatabase}
\getchunk{defun deleteFile}
\getchunk{defun describe}
\getchunk{defun describeFortPersistence}
```

```
\getchunk{defun describeInputLibraryArgs}
\getchunk{defun describeOutputLibraryArgs}
\getchunk{defun describeSetFortDir}
\getchunk{defun describeSetFortTmpDir}
\getchunk{defun describeSetFunctionsCache}
\getchunk{defun describeSetLinkerArgs}
\getchunk{defun describeSetNagHost}
\getchunk{defun describeSetOutputAlgebra}
\getchunk{defun describeSetOutputFormula}
\getchunk{defun describeSetOutputFortran}
\getchunk{defun describeSetOutputHtml}
\getchunk{defun describeSetOutputMathml}
\getchunk{defun describeSetOutputOpenMath}
\getchunk{defun describeSetOutputTex}
\getchunk{defun describeSetStreamsCalculate}
\getchunk{defun describeSpad2Cmd}
\getchunk{defun dewritify}
\getchunk{defun dewritify,dewritifyInner}
\getchunk{defun diffAlist}
\getchunk{defun digit?}
\getchunk{defun digitp}
\getchunk{defun disableHist}
\getchunk{defun display}
\getchunk{defun displayCondition}
\getchunk{defun displayExposedConstructors}
\getchunk{defun displayExposedGroups}
\getchunk{defun displayHiddenConstructors}
\getchunk{defun displayMacro}
\getchunk{defun displayMacros}
\getchunk{defun displayMode}
\getchunk{defun displayModemap}
\getchunk{defun displayOperations}
\getchunk{defun displayOperationsFromLisplib}
\getchunk{defun displayParserMacro}
\getchunk{defun displayProperties}
\getchunk{defun displayProperties,sayFunctionDeps}
\getchunk{defun displaySetOptionInformation}
\getchunk{defun displaySetVariableSettings}
\getchunk{defun displaySpad2Cmd}
\getchunk{defun displayType}
\getchunk{defun displayValue}
\getchunk{defun displayWorkspaceNames}
\getchunk{defun doDoitButton}
\getchunk{defun domainDescendantsOf}
\getchunk{defun domainToGenvar}
\getchunk{defun domArg}
\getchunk{defun domArg2}
\getchunk{defun doSystemCommand}
\getchunk{defun downcase}
\getchunk{defun downlink}
\getchunk{defun dqConcat}
\getchunk{defun dropInputLibrary}
\getchunk{defun dSearch}
\getchunk{defun dumbTokenize}
```

```
\getchunk{defun edit}
\getchunk{defun editFile}
\getchunk{defun editSpad2Cmd}
\getchunk{defun Else?}
\getchunk{defun Elseif?}
\getchunk{defun embededFunction}
\getchunk{defun enPile}
\getchunk{defun eofp}
\getchunk{defun eqpileTree}
\getchunk{defun erMsgCompare}
\getchunk{defun erMsgSep}
\getchunk{defun erMsgSort}
\getchunk{defun evalCategory}
\getchunk{defun evalDomain}
\getchunk{defun evaluateSignature}
\getchunk{defun evaluateType}
\getchunk{defun evaluateType1}
\getchunk{defun executeInterpreterCommand}
\getchunk{defun ExecuteInterpSystemCommand}
\getchunk{defun executeQuietCommand}
\getchunk{defun explainLinear}

\getchunk{defun fetchOutput}
\getchunk{defun fillerSpaces}
\getchunk{defun filterAndFormatConstructors}
\getchunk{defun filterListOfStrings}
\getchunk{defun filterListOfStringsWithFn}
\getchunk{defun finalExactRequest}
\getchunk{defun findnexttest}
\getchunk{defun firstTokPosn}
\getchunk{defun fixObjectForPrinting}
\getchunk{defun flatBvList}
\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 gammaRatapprox}
\getchunk{defun gammaRatkernel}
\getchunk{defun gammaStirling}
\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 getConstantFromDomain}
\getchunk{defun getConstructorDocumentation}
\getchunk{defun getdatabase}
\getchunk{defun getDependentsOfConstructor}
\getchunk{defun getDirectoryList}
\getchunk{defun getFirstWord}
\getchunk{defun getHtMacroItem}
\getchunk{defun getMapSig}
\getchunk{defun getMapSubNames}
\getchunk{defun getMsgCatAttr}
\getchunk{defun getMsgFTTag?}
\getchunk{defun getMsgInfoFromKey}
\getchunk{defun getMsgPos}
\getchunk{defun getMsgPos2}
\getchunk{defun getMsgToWhere}
\getchunk{defun getOperationAlistFromLisplib}
\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 getSubDomainPredicate}
\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 handleTokensizeSystemCommands}
\getchunk{defun hasAtt}
\getchunk{defun hasAttSig}
\getchunk{defun hasCatExpression}
\getchunk{defun hasCate}
\getchunk{defun hasCateSpecial}
\getchunk{defun hasCateSpecialNew}
\getchunk{defun hasCate1}
\getchunk{defun hasCaty}
\getchunk{defun hasCaty1}
\getchunk{defun hashable}
\getchunk{defun hasOption}
\getchunk{defun hasPair}
\getchunk{defun hasSharpVar}
\getchunk{defun hasSig}
\getchunk{defun hasSigAnd}
\getchunk{defun hasSigOr}
\getchunk{defun help}
```

```
\getchunk{defun helpSpad2Cmd}
\getchunk{defun histFileErase}
\getchunk{defun histFileName}
\getchunk{defun histInputFileName}
\getchunk{defun history}
\getchunk{defun historySpad2Cmd}
\getchunk{defun hkeys}
\getchunk{defun hput}
\getchunk{defun htAddHeading}
\getchunk{defun htAllOrNum}
\getchunk{defun htBcLinks}
\getchunk{defun htBcLispLinks}
\getchunk{defun htBcRadioButtons}
\getchunk{defun htCacheAddChoice}
\getchunk{defun htCacheOne}
\getchunk{defun htCacheSet}
\getchunk{defun htCheckList}
\getchunk{defun htCheck}
\getchunk{defun htDoneButton}
\getchunk{defun htDoNothing}
\getchunk{defun htEscapeString}
\getchunk{defun htFunctionSetLiteral}
\getchunk{defun htGlossPage}
\getchunk{defun htGlossSearch}
\getchunk{defun htGloss}
\getchunk{defun htGreekSearch}
\getchunk{defun htInitPage}
\getchunk{defun htInitPageNoScroll}
\getchunk{defun htInputStrings}
\getchunk{defun htKill}
\getchunk{defun htLispLinks}
\getchunk{defun htLispMemoLinks}
\getchunk{defun htMakeButton}
\getchunk{defun htMakeDoitButton}
\getchunk{defun htMakeDoneButton}
\getchunk{defun htMakeErrorPage}
\getchunk{defun htMakeInputList}
\getchunk{defun htMakeLabel}
\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 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 htpLabelFilter}
\getchunk{defun htpLabelInputString}
\getchunk{defun htpLabelSpadType}
\getchunk{defun htpLabelSpadValue}
\getchunk{defun htpLabelType}
\getchunk{defun htpName}
\getchunk{defun htpPageDescription}
\getchunk{defun htpRadioButtonAlist}
\getchunk{defun htpProperty}
\getchunk{defun htpPropertyList}
\getchunk{defun htProcessBcButtons}
\getchunk{defun htProcessBcStrings}
\getchunk{defun htProcessDoitButton}
\getchunk{defun htProcessDomainConditions}
\getchunk{defun htProcessDoneButton}
\getchunk{defun htProcessToggleButtons}
\getchunk{defun htpSetDomainConditions}
\getchunk{defun htpSetDomainPvarSubstList}
\getchunk{defun htpSetDomainVariableAlist}
\getchunk{defun htpSetInputAreaAlist}
\getchunk{defun htpSetLabelErrorMsg}
\getchunk{defun htpSetLabelInputString}
\getchunk{defun htpSetLabelSpadValue}
\getchunk{defun htpSetName}
\getchunk{defun htpSetPageDescription}
\getchunk{defun htpSetProperty}
\getchunk{defun htpSetRadioButtonAlist}
\getchunk{defun htQuote}
\getchunk{defun htRadioButtons}
\getchunk{defun htSay}
\getchunk{defun htSayBind}
\getchunk{defun htSetCache}
\getchunk{defun htSetExpose}
\getchunk{defun htSetFunCommandContinue}
\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 inclFname}
\getchunk{defun incLine}
\getchunk{defun incLine1}
\getchunk{defun inclmsgCannotRead}
\getchunk{defun inclmsgFileCycle}
\getchunk{defun inclmsgPrematureFin}
\getchunk{defun incLude}
\getchunk{defun incLude1}
\getchunk{defun inclmsgConActive}
\getchunk{defun inclmsgConStill}
\getchunk{defun inclmsgIfSyntax}
\getchunk{defun inclmsgNoSuchFile}
\getchunk{defun inclmsgSay}
\getchunk{defun incNConsoles}
\getchunk{defun incRenumber}
\getchunk{defun incRenumberItem}
\getchunk{defun incRenumberLine}
\getchunk{defun incRgen}
\getchunk{defun incRgen1}
\getchunk{defun incStream}
\getchunk{defun incString}
\getchunk{defun incZip}
\getchunk{defun incZip1}
\getchunk{defun init-boot/spad-reader}
```

```
\getchunk{defun initHist}
\getchunk{defun initHistList}
\getchunk{defun initial-getdatabase}
\getchunk{defun initializeInterpreterFrameRing}
\getchunk{defun initializeSetVariables}
\getchunk{defun initImPr}
\getchunk{defun initroot}
\getchunk{defun initToWhere}
\getchunk{defun insertAlist}
\getchunk{defun insertpile}
\getchunk{defun InterpExecuteSpadSystemCommand}
\getchunk{defun interpFunctionDepAlists}
\getchunk{defun interpopen}
\getchunk{defun interpret}
\getchunk{defun interpret1}
\getchunk{defun interpret2}
\getchunk{defun interpretTopLevel}
\getchunk{defun intInterpretPform}
\getchunk{defun intloop}
\getchunk{defun intloopEchoParse}
\getchunk{defun intloopInclude}
\getchunk{defun intloopInclude0}
\getchunk{defun intnplisp}
\getchunk{defun intloopProcess}
\getchunk{defun intloopProcessString}
\getchunk{defun intloopReadConsole}
\getchunk{defun intloopSpadProcess}
\getchunk{defun intloopSpadProcess,interp}
\getchunk{defun intProcessSynonyms}
\getchunk{defun ioclear}
\getchunk{defun iostat}
\getchunk{defun isDomainOrPackage}
\getchunk{defun isDomainValuedVariable}
\getchunk{defun isEqualOrSubDomain}
\getchunk{defun isExposedConstructor}
\getchunk{defun isGenvar}
\getchunk{defun isInterpOnlyMap}
\getchunk{defun isListOfIdentifiers}
\getchunk{defun isListOfIdentifiersOrStrings}
\getchunk{defun isPartialMode}
\getchunk{defun isPatternVar}
\getchunk{defun isSharpVar}
\getchunk{defun isSharpVarWithNum}
\getchunk{defun isSubForRedundantMapName}
\getchunk{defun isSystemDirectory}
\getchunk{defun isTaggedUnion}
\getchunk{defun isUncompiledMap}

\getchunk{defun justifyMyType}

\getchunk{defun kArgPage}
\getchunk{defun kArgumentCheck}
\getchunk{defun kcaPage}
\getchunk{defun kcaPage1}
```

```
\getchunk{defun kccPage}
\getchunk{defun kcdePage}
\getchunk{defun kcdPage}
\getchunk{defun kcdoPage}
\getchunk{defun kCheckArgumentNumbers}
\getchunk{defun kcnPage}
\getchunk{defun kcPage}
\getchunk{defun kcpPage}
\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 ksPage}
\getchunk{defun kcuPage}
\getchunk{defun kPage}
\getchunk{defun kSearch}
\getchunk{defun kTestPred}

\getchunk{defun lassocSub}
\getchunk{defun lastTokPosn}
\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 license}
\getchunk{defun linearFinalRequest}
\getchunk{defun lineoftoks}
\getchunk{defun linkGen}
\getchunk{defun listConstructorAbbreviations}
\getchunk{defun listDecideHowMuch}
\getchunk{defun listOfStrings2String}
\getchunk{defun listOutputter}
\getchunk{defun lncgamma}
\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 lnrgamma}
\getchunk{defun lnrgammaRatapprox}
\getchunk{defun ltrace}

\getchunk{defun macApplication}
\getchunk{defun macExpand}
\getchunk{defun macId}
\getchunk{defun macLambda}
\getchunk{defun macLambda,mac}
\getchunk{defun macLambdaParameterHandling}
\getchunk{defun macMacro}
\getchunk{defun macSubstituteId}
\getchunk{defun macSubstituteOuter}
\getchunk{defun macroExpanded}
\getchunk{defun macWhere}
\getchunk{defun macWhere,mac}
\getchunk{defun macOExpandBody}
\getchunk{defun macOGet}
\getchunk{defun macOGetName}
\getchunk{defun macOInfiniteExpansion}
\getchunk{defun macOMLambdaApply}
\getchunk{defun macOSubstituteOuter}
\getchunk{defun make-appendstream}
\getchunk{defun make-databases}
\getchunk{defun makeFullNamestring}
\getchunk{defun makeHistFileName}
\getchunk{defun makeInputFilename}
\getchunk{defun make-instream}
\getchunk{defun makeLeaderMsg}
\getchunk{defun makeMsgFromLine}
\getchunk{defun makeOrdinal}
\getchunk{defun make-outstream}
\getchunk{defun makePathname}
\getchunk{defun 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 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 monitorEvalAfter}
\getchunk{defun monitorEvalBefore}
\getchunk{defun monitorEvalTran}
\getchunk{defun monitorEvalTran1}
\getchunk{defun monitorGetValue}
\getchunk{defun monitorPrint}
\getchunk{defun monitorPrintArg}
\getchunk{defun monitorPrintArgs}
\getchunk{defun monitorPrintRest}
\getchunk{defun monitorPrintValue}
\getchunk{defun monitorX}
\getchunk{defun monitorEnter}
\getchunk{defun monitorExit}
\getchunk{defun monitorXX}
\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-show}
\getchunk{defun npAdd}
\getchunk{defun npADD}
\getchunk{defun npAmpersand}
\getchunk{defun npAmpersandFrom}
\getchunk{defun npAndOr}
```

```
\getchunk{defun npAngleBared}
\getchunk{defun npApplication}
\getchunk{defun npApplication2}
\getchunk{defun npArith}
\getchunk{defun npAssign}
\getchunk{defun npAssignment}
\getchunk{defun npAssignVariable}
\getchunk{defun npAtom1}
\getchunk{defun npAtom2}
\getchunk{defun npBacksetElse}
\getchunk{defun npBackTrack}
\getchunk{defun npBDefinition}
\getchunk{defun npBPileDefinition}
\getchunk{defun npBraced}
\getchunk{defun npBracked}
\getchunk{defun npBracketed}
\getchunk{defun npBreak}
\getchunk{defun npBy}
\getchunk{defun npCategory}
\getchunk{defun npCategoryL}
\getchunk{defun npCoerceTo}
\getchunk{defun npColon}
\getchunk{defun npColonQuery}
\getchunk{defun npComma}
\getchunk{defun npCommaBackSet}
\getchunk{defun npCompMissing}
\getchunk{defun npConditional}
\getchunk{defun npConditionalStatement}
\getchunk{defun npConstTok}
\getchunk{defun npDDInfKey}
\getchunk{defun npDecl}
\getchunk{defun npDef}
\getchunk{defun npDefaultDecl}
\getchunk{defun npDefaultItem}
\getchunk{defun npDefaultItemlist}
\getchunk{defun npDefaultValue}
\getchunk{defun npDefinition}
\getchunk{defun npDefinitionItem}
\getchunk{defun npDefinitionlist}
\getchunk{defun npDefinitionOrStatement}
\getchunk{defun npDefn}
\getchunk{defun npDefTail}
\getchunk{defun npDiscrim}
\getchunk{defun npDisjand}
\getchunk{defun npDollar}
\getchunk{defun npDotted}
\getchunk{defun npElse}
\getchunk{defun npEncAp}
\getchunk{defun npEncl}
\getchunk{defun npEnclosed}
\getchunk{defun npEqKey}
\getchunk{defun npExit}
\getchunk{defun npExpress}
\getchunk{defun npExpress1}
```

```
\getchunk{defun npExport}
\getchunk{defun npFirstTok}
\getchunk{defun npFix}
\getchunk{defun npForIn}
\getchunk{defun npFree}
\getchunk{defun npFromdom}
\getchunk{defun npFromdom1}
\getchunk{defun npGives}
\getchunk{defun npId}
\getchunk{defun npImport}
\getchunk{defun npInfGeneric}
\getchunk{defun npInfixOp}
\getchunk{defun npInfixOperator}
\getchunk{defun npInfKey}
\getchunk{defun npInline}
\getchunk{defun npInterval}
\getchunk{defun npItem}
\getchunk{defun npItem1}
\getchunk{defun npIterate}
\getchunk{defun npIterator}
\getchunk{defun npIterators}
\getchunk{defun npLambda}
\getchunk{defun npLeftAssoc}
\getchunk{defun npLet}
\getchunk{defun npLetQualified}
\getchunk{defun npList}
\getchunk{defun npListAndRecover}
\getchunk{defun npListing}
\getchunk{defun npListofFun}
\getchunk{defun npLocal}
\getchunk{defun npLocalDecl}
\getchunk{defun npLocalItem}
\getchunk{defun npLocalItemlist}
\getchunk{defun npLogical}
\getchunk{defun npLoop}
\getchunk{defun npMacro}
\getchunk{defun npMatch}
\getchunk{defun npMdef}
\getchunk{defun npMDEF}
\getchunk{defun npMDEFinition}
\getchunk{defun npMissing}
\getchunk{defun npMissingMate}
\getchunk{defun npMoveTo}
\getchunk{defun npName}
\getchunk{defun npNext}
\getchunk{defun npNull}
\getchunk{defun npParened}
\getchunk{defun npParenthesize}
\getchunk{defun npParenthesized}
\getchunk{defun npParse}
\getchunk{defun npPDefinition}
\getchunk{defun npPileBracketed}
\getchunk{defun npPileDefinitionlist}
\getchunk{defun npPileExit}
```

```
\getchunk{defun npPower}
\getchunk{defun npPP}
\getchunk{defun npPPf}
\getchunk{defun npPPff}
\getchunk{defun npPPg}
\getchunk{defun npPrefixColon}
\getchunk{defun npPretend}
\getchunk{defun npPrimary}
\getchunk{defun npPrimary1}
\getchunk{defun npPrimary2}
\getchunk{defun npProcessSynonym}
\getchunk{defun npProduct}
\getchunk{defun npPushId}
\getchunk{defun npRelation}
\getchunk{defun npRemainder}
\getchunk{defun npQualDef}
\getchunk{defun npQualified}
\getchunk{defun npQualifiedDefinition}
\getchunk{defun npQualType}
\getchunk{defun npQualTypelist}
\getchunk{defun npQuiver}
\getchunk{defun npRecoverTrap}
\getchunk{defun npRestore}
\getchunk{defun npRestrict}
\getchunk{defun npReturn}
\getchunk{defun npRightAssoc}
\getchunk{defun npRule}
\getchunk{defun npSCategory}
\getchunk{defun npSDefaultItem}
\getchunk{defun npSegment}
\getchunk{defun npSelector}
\getchunk{defun npSemiBackSet}
\getchunk{defun npSemiListing}
\getchunk{defun npSigDecl}
\getchunk{defun npSigItem}
\getchunk{defun npSigItemlist}
\getchunk{defun npSignature}
\getchunk{defun npSignatureDefinee}
\getchunk{defun npSingleRule}
\getchunk{defun npSLocalItem}
\getchunk{defun npSQualTypelist}
\getchunk{defun npStatement}
\getchunk{defun npSuch}
\getchunk{defun npSuchThat}
\getchunk{defun npSum}
\getchunk{defun npsynonym}
\getchunk{defun npSymbolVariable}
\getchunk{defun npSynthetic}
\getchunk{defun npsystem}
\getchunk{defun npState}
\getchunk{defun npTagged}
\getchunk{defun npTerm}
\getchunk{defun npTrap}
\getchunk{defun npTrapForm}
```

```
\getchunk{defun npTuple}
\getchunk{defun npType}
\getchunk{defun npTypedForm}
\getchunk{defun npTypedForm1}
\getchunk{defun npTypeStyle}
\getchunk{defun npTypified}
\getchunk{defun npTyping}
\getchunk{defun npTypeVariable}
\getchunk{defun npTypeVariablelist}
\getchunk{defun npVariable}
\getchunk{defun npVariablelist}
\getchunk{defun npVariableName}
\getchunk{defun npVoid}
\getchunk{defun npWConditional}
\getchunk{defun npWhile}
\getchunk{defun npWith}
\getchunk{defun npZeroOrMore}
\getchunk{defun NRTevalDomain}

\getchunk{defun ofCategory}
\getchunk{defun oldCompLookup}
\getchunk{defun oldHistFileName}
\getchunk{defun 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-getEndObject}
\getchunk{defun om-getError}
\getchunk{defun om-getFloat}
\getchunk{defun om-getInt}
\getchunk{defun om-getObject}
\getchunk{defun om-getString}
\getchunk{defun om-getSymbol}
\getchunk{defun om-getType}
\getchunk{defun om-getVar}
\getchunk{defun om-listCDs}
\getchunk{defun om-listSymbols}
\getchunk{defun om-makeConn}
\getchunk{defun om-openFileDev}
\getchunk{defun om-openStringDev}
```

```
\getchunk{defun om-putApp}
\getchunk{defun om-putAtp}
\getchunk{defun om-putAttr}
\getchunk{defun om-putBind}
\getchunk{defun om-putBVar}
\getchunk{defun om-putByteArray}
\getchunk{defun om-putEndApp}
\getchunk{defun om-putEndAtp}
\getchunk{defun om-putEndAttr}
\getchunk{defun om-putEndBind}
\getchunk{defun om-putEndBVar}
\getchunk{defun om-putEndError}
\getchunk{defun om-putEndObject}
\getchunk{defun om-putError}
\getchunk{defun om-putFloat}
\getchunk{defun om-putInt}
\getchunk{defun om-putObject}
\getchunk{defun om-putString}
\getchunk{defun om-putSymbol}
\getchunk{defun om-putVar}
\getchunk{defun om-Read}
\getchunk{defun om-setDevEncoding}
\getchunk{defun om-stringPtrToString}
\getchunk{defun om-stringToStringPtr}
\getchunk{defun om-supportsCD}
\getchunk{defun om-supportsSymbol}
\getchunk{defun openOutputLibrary}
\getchunk{defun openserver}
\getchunk{defun operationopen}
\getchunk{defun optionError}
\getchunk{defun /options}
\getchunk{defun optionUserLevelError}
\getchunk{defun orderBySlotNumber}
\getchunk{defun originsInOrder}

\getchunk{defun parseAndEval}
\getchunk{defun parseAndEval1}
\getchunk{defun parseAndInterpret}
\getchunk{defun parseFromString}
\getchunk{defun parseNoMacroFromString}
\getchunk{defun parseSystemCmd}
\getchunk{defun parseWord}
\getchunk{defun pathname}
\getchunk{defun pathnameDirectory}
\getchunk{defun pathnameName}
\getchunk{defun pathnameType}
\getchunk{defun pathnameTypeId}
\getchunk{defun patternVarsOf}
\getchunk{defun patternVarsOf1}
\getchunk{defun permuteToOrder}
\getchunk{defun pcounters}
\getchunk{defun pfAbSynOp}
\getchunk{defun pfAbSynOp?}
\getchunk{defun pfAdd}
```

```
\getchunk{defun pfAnd}
\getchunk{defun pfAnd?}
\getchunk{defun pfApplication}
\getchunk{defun pfApplication?}
\getchunk{defun pfApplication2Sex}
\getchunk{defun pfAssign}
\getchunk{defun pfAssign?}
\getchunk{defun pfAttribute}
\getchunk{defun pfBrace}
\getchunk{defun pfBraceBar}
\getchunk{defun pfBracket}
\getchunk{defun pfBracketBar}
\getchunk{defun pfBreak}
\getchunk{defun pfBreak?}
\getchunk{defun pfCharPosn}
\getchunk{defun pfCheckArg}
\getchunk{defun pfCheckMacroOut}
\getchunk{defun pfCheckId}
\getchunk{defun pfCheckItOut}
\getchunk{defun pfCoerceto}
\getchunk{defun pfCoerceto?}
\getchunk{defun pfCollect}
\getchunk{defun pfCollect?}
\getchunk{defun pfCollect1?}
\getchunk{defun pfCollectArgTran}
\getchunk{defun pfCollectVariable1}
\getchunk{defun pfCollect2Sex}
\getchunk{defun pfCopyWithPos}
\getchunk{defun pfDefinition}
\getchunk{defun pfDefinition?}
\getchunk{defun pfDefinition2Sex}
\getchunk{defun pfDo}
\getchunk{defun pfDo?}
\getchunk{defun pfDocument}
\getchunk{defun pfEnSequence}
\getchunk{defun pfExit}
\getchunk{defun pfExit?}
\getchunk{defun pfExport}
\getchunk{defun pfExpression}
\getchunk{defun pfFileName}
\getchunk{defun pfFix}
\getchunk{defun pfFlattenApp}
\getchunk{defun pfFree}
\getchunk{defun pfFree?}
\getchunk{defun pfForin}
\getchunk{defun pfForin?}
\getchunk{defun pfFromDom}
\getchunk{defun pfFromdom}
\getchunk{defun pfFromdom?}
\getchunk{defun pfGlobalLinePosn}
\getchunk{defun pfHide}
\getchunk{defun pfId}
\getchunk{defun pfId?}
\getchunk{defun pfIdPos}
```

```
\getchunk{defun pfIdSymbol}
\getchunk{defun pfIf}
\getchunk{defun pfIf?}
\getchunk{defun pfIfThenOnly}
\getchunk{defun pfImport}
\getchunk{defun pfInline}
\getchunk{defun pfInfApplication}
\getchunk{defun pfIterate}
\getchunk{defun pfIterate?}
\getchunk{defun pfLam}
\getchunk{defun pfLambda}
\getchunk{defun pfLambdaTran}
\getchunk{defun pfLambda?}
\getchunk{defun pfLambda2Sex}
\getchunk{defun pfLeaf}
\getchunk{defun pfLeaf?}
\getchunk{defun pfLeafPosition}
\getchunk{defun pfLeafToken}
\getchunk{defun pfLhsRule2Sex}
\getchunk{defun pfLinePosn}
\getchunk{defun pfListOf}
\getchunk{defun pfLiteralClass}
\getchunk{defun pfLiteralString}
\getchunk{defun pfLiteral2Sex}
\getchunk{defun pfLocal}
\getchunk{defun pfLocal?}
\getchunk{defun pfLoop}
\getchunk{defun pfLoop1}
\getchunk{defun pfLoop?}
\getchunk{defun pfLp}
\getchunk{defun pfMacro}
\getchunk{defun pfMacro?}
\getchunk{defun pfMapParts}
\getchunk{defun pfMLambda}
\getchunk{defun pfMLambda?}
\getchunk{defun pfname}
\getchunk{defun pfNoPosition}
\getchunk{defun pfNoPosition?}
\getchunk{defun pfNot?}
\getchunk{defun pfNothing}
\getchunk{defun pfNothing?}
\getchunk{defun pfNovalue}
\getchunk{defun pfNovalue?}
\getchunk{defun pfOp2Sex}
\getchunk{defun pfOr}
\getchunk{defun pfOr?}
\getchunk{defun pfParen}
\getchunk{defun pfPretend}
\getchunk{defun pfPretend?}
\getchunk{defun pfPushBody}
\getchunk{defun pfPushMacroBody}
\getchunk{defun pfQualType}
\getchunk{defun pfRestrict}
\getchunk{defun pfRestrict?}
```

```
\getchunk{defun pfRetractTo}
\getchunk{defun pfReturn}
\getchunk{defun pfReturn?}
\getchunk{defun pfReturnNoName}
\getchunk{defun pfReturnTyped}
\getchunk{defun pfRhsRule2Sex}
\getchunk{defun pfRule}
\getchunk{defun pfRule?}
\getchunk{defun pfRule2Sex}
\getchunk{defun pfSequence}
\getchunk{defun pfSequence?}
\getchunk{defun pfSequenceToList}
\getchunk{defun pfSequence2Sex}
\getchunk{defun pfSequence2Sex0}
\getchunk{defun pfSexpr}
\getchunk{defun pfSexpr,strip}
\getchunk{defun pfSourcePosition}
\getchunk{defun pfSourceStok}
\getchunk{defun pfSpread}
\getchunk{defun pfSuch}
\getchunk{defun pfSuchthat}
\getchunk{defun pfSuchthat?}
\getchunk{defun pfSuchThat2Sex}
\getchunk{defun pfSymb}
\getchunk{defun pfSymbol}
\getchunk{defun pfSymbol?}
\getchunk{defun pfSymbolSymbol}
\getchunk{defun pfTagged}
\getchunk{defun pfTagged?}
\getchunk{defun pfTaggedToTyped}
\getchunk{defun pfTaggedToTyped1}
\getchunk{defun pfTransformArg}
\getchunk{defun pfTuple}
\getchunk{defun pfTupleListOf}
\getchunk{defun pfTweakIf}
\getchunk{defun pfTyped}
\getchunk{defun pfTyped?}
\getchunk{defun pfTyping}
\getchunk{defun pfTuple?}
\getchunk{defun pfUnSequence}
\getchunk{defun pfWDec}
\getchunk{defun pfWDeclare}
\getchunk{defun pfWhere}
\getchunk{defun pfWhere?}
\getchunk{defun pfWhile}
\getchunk{defun pfWhile?}
\getchunk{defun pfWith}
\getchunk{defun pfWrong}
\getchunk{defun pfWrong?}
\getchunk{defun pf0ApplicationArgs}
\getchunk{defun pf0DefinitionLhsItems}
\getchunk{defun pf0FlattenSyntacticTuple}
\getchunk{defun pf0ForinLhs}
\getchunk{defun pf0FreeItems}
```

```
\getchunk{defun pf0LambdaArgs}
\getchunk{defun pf0LocalItems}
\getchunk{defun pf0LoopIterators}
\getchunk{defun pf0MLambdaArgs}
\getchunk{defun pf0SequenceArgs}
\getchunk{defun pf0TupleParts}
\getchunk{defun pf0WhereContext}
\getchunk{defun pf2Sex}
\getchunk{defun pf2Sex1}
\getchunk{defun phMacro}
\getchunk{defun phParse}
\getchunk{defun phInterpret}
\getchunk{defun phIntReportMsgs}
\getchunk{defun phiRatapprox}
\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 PiMinusLogSinPi}
\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 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 pSearch}
\getchunk{defun PsiAsymptotic}
```

```
\getchunk{defun PsiBack}
\getchunk{defun PsiXotic}
\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 rbesseli}
\getchunk{defun rbesselj}
\getchunk{defun rdefinstream}
\getchunk{defun rdefoutstream}
\getchunk{defun read}
\getchunk{defun /read}
\getchunk{defun readHiFi}
\getchunk{defun readline}
\getchunk{defun readSpadProfileIfThere}
\getchunk{defun readSpad2Cmd}
\getchunk{defun reassembleTowerIntoType}
\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 command}
\getchunk{defun regress}
\getchunk{defun remFile}
\getchunk{defun remover}
\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 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 resethashtables}
\getchunk{defun resetInCoreHist}
\getchunk{defun resetSpacers}
\getchunk{defun resetTimers}
\getchunk{defun resetWorkspaceVariables}
\getchunk{defun restart}
\getchunk{defun restart0}
\getchunk{defun restoreHistory}
\getchunk{defun retract}
\getchunk{defun retractByFunction}
\getchunk{defun retractUnderDomain}
\getchunk{defun retract2Specialization}
\getchunk{defun rgamma}
\getchunk{defun rgammaImpl}
\getchunk{defun rlngamma}
\getchunk{defun rpsi}
\getchunk{defun rPsiImpl}
\getchunk{defun rPsiW}
\getchunk{defun rread}
\getchunk{defun ruleLhsTran}
\getchunk{defun rulePredicateTran}
\getchunk{defun runspad}
\getchunk{defun rwrite}

\getchunk{defun safeWritify}
\getchunk{defun sameMsg?}
\getchunk{defun satisfiesRegularExpressions}
\getchunk{defun saveHistory}
\getchunk{defun saveMapSig}
\getchunk{defun savesystem}
\getchunk{defun saveDependentsHashTable}
\getchunk{defun saveUsersHashTable}
\getchunk{defun sayAllCacheCounts}
\getchunk{defun sayBrightly1}
\getchunk{defun sayCacheCount}
\getchunk{defun sayExample}
\getchunk{defun sayKeyedMsg}
\getchunk{defun sayKeyedMsgLocal}
\getchunk{defun sayMSG}
\getchunk{defun sayMSG2File}
\getchunk{defun sayShowWarning}
\getchunk{defun scanCheckRadix}
```

```
\getchunk{defun scanComment}
\getchunk{defun scanDictCons}
\getchunk{defun scanError}
\getchunk{defun scanEsc}
\getchunk{defun scanEscape}
\getchunk{defun scanExponent}
\getchunk{defun scanIgnoreLine}
\getchunk{defun scanInsert}
\getchunk{defun scanKeyTr}
\getchunk{defun scanNegComment}
\getchunk{defun scanNumber}
\getchunk{defun ScanOrPairVec}
\getchunk{defun ScanOrPairVec,ScanOrInner}
\getchunk{defun scanPossFloat}
\getchunk{defun scanPunct}
\getchunk{defun scanPunCons}
\getchunk{defun scanS}
\getchunk{defun scanSpace}
\getchunk{defun scanString}
\getchunk{defun scanKeyTableCons}
\getchunk{defun scanToken}
\getchunk{defun scanTransform}
\getchunk{defun scanW}
\getchunk{defun scanWord}
\getchunk{defun search}
\getchunk{defun searchCurrentEnv}
\getchunk{defun searchTailEnv}
\getchunk{defun segmentKeyedMsg}
\getchunk{defun selectOption}
\getchunk{defun selectOptionLC}
\getchunk{defun separatePiles}
\getchunk{defun serverReadLine}
\getchunk{defun set}
\getchunk{defun set1}
\getchunk{defun setdatabase}
\getchunk{defun setExpose}
\getchunk{defun setExposeAdd}
\getchunk{defun setExposeAddConstr}
\getchunk{defun setExposeAddGroup}
\getchunk{defun setExposeDrop}
\getchunk{defun setExposeDropConstr}
\getchunk{defun setExposeDropGroup}
\getchunk{defun setFortDir}
\getchunk{defun setFortPers}
\getchunk{defun setFortTmpDir}
\getchunk{defun setFunctionsCache}
\getchunk{defun setHistoryCore}
\getchunk{defun setInputLibrary}
\getchunk{defun setIOindex}
\getchunk{defun setLinkerArgs}
\getchunk{defun setMsgCatlessAttr}
\getchunk{defun setMsgForcedAttr}
\getchunk{defun setMsgForcedAttrList}
\getchunk{defun setMsgUnforcedAttr}
```

```
\getchunk{defun setMsgUnforcedAttrList}
\getchunk{defun setNagHost}
\getchunk{defun setOutputAlgebra}
\getchunk{defun setOutputCharacters}
\getchunk{defun setOutputFormula}
\getchunk{defun setOutputFortran}
\getchunk{defun setOutputLibrary}
\getchunk{defun setOutputHtml}
\getchunk{defun setOutputMathml}
\getchunk{defun setOutputOpenMath}
\getchunk{defun setOutputTex}
\getchunk{defun setStreamsCalculate}
\getchunk{defun setUpDefault}
\getchunk{defun shortenForPrinting}
\getchunk{defun show}
\getchunk{defun showdatabase}
\getchunk{defun showHistory}
\getchunk{defun showInOut}
\getchunk{defun showInput}
\getchunk{defun showSpad2Cmd}
\getchunk{defun shut}
\getchunk{defun size}
\getchunk{defun SkipEnd?}
\getchunk{defun SkipPart?}
\getchunk{defun Skipping?}
\getchunk{defun smallEnough}
\getchunk{defun spad}
\getchunk{defun spadClosure?}
\getchunk{defun spad-error-loc}
\getchunk{defun SpadInterpretStream}
\getchunk{defun spad-long-error}
\getchunk{defun spadReply}
\getchunk{defun spadrread}
\getchunk{defun spadrwrite}
\getchunk{defun spadrwrite0}
\getchunk{defun spad-save}
\getchunk{defun spad-short-error}
\getchunk{defun spadStartUpMsgs}
\getchunk{defun spad-syntax-error}
\getchunk{defun spadTrace}
\getchunk{defun spadTraceAlias}
\getchunk{defun spadTrace,g}
\getchunk{defun spadTrace,isTraceable}
\getchunk{defun spadUntrace}
\getchunk{defun spad2BootCoerce}
\getchunk{defun specialChar}
\getchunk{defun spleI}
\getchunk{defun spleI1}
\getchunk{defun split}
\getchunk{defun splitIntoOptionBlocks}
\getchunk{defun stackTraceOptionError}
\getchunk{defun startp}
\getchunk{defun startsNegComment?}
\getchunk{defun statisticsInitialization}
```

```
\getchunk{defun streamChop}
\getchunk{defun stringList2String}
\getchunk{defun stringMatches?}
\getchunk{defun string2Constructor}
\getchunk{defun StringToDir}
\getchunk{defun stripUnionTags}
\getchunk{defun strpos}
\getchunk{defun strposl}
\getchunk{defun stupidIsSpadFunction}
\getchunk{defun subMatch}
\getchunk{defun substFromAlist}
\getchunk{defun substringMatch}
\getchunk{defun subTypes}
\getchunk{defun summary}
\getchunk{defun syGeneralErrorHere}
\getchunk{defun syIgnoredFromTo}
\getchunk{defun synonym}
\getchunk{defun synonymsForUserLevel}
\getchunk{defun synonymSpad2Cmd}
\getchunk{defun sySpecificErrorAtToken}
\getchunk{defun sySpecificErrorHere}
\getchunk{defun systemCommand}

\getchunk{defun ?t}
\getchunk{defun tabbing}
\getchunk{defun templateParts}
\getchunk{defun tangle}
\getchunk{defun terminateSystemCommand}
\getchunk{defun tersyscommand}
\getchunk{defun testnumberp}
\getchunk{defun testpassed}
\getchunk{defun thisPosIsEqual}
\getchunk{defun thisPosIsLess}
\getchunk{defun throwEvalTypeMsg}
\getchunk{defun toFile?}
\getchunk{defun tokConstruct}
\getchunk{defun token-stack-show}
\getchunk{defun tokPosn}
\getchunk{defun tokTran}
\getchunk{defun tokType}
\getchunk{defun topLevelInterpEval}
\getchunk{defun trace}
\getchunk{defun trace1}
\getchunk{defun traceDomainConstructor}
\getchunk{defun traceDomainLocalOps}
\getchunk{defun tracelet}
\getchunk{defun traceOptionError}
\getchunk{defun /tracereply}
\getchunk{defun traceReply}
\getchunk{defun trace2}
\getchunk{defun trace3}
\getchunk{defun traceSpad2Cmd}
\getchunk{defun translateTrueFalse2YesNo}
\getchunk{defun translateYesNo2TrueFalse}
```

```
\getchunk{defun translateYesNoToTrueFalse}
\getchunk{defun transOnlyOption}
\getchunk{defun transTraceItem}
\getchunk{defun trunclist}
\getchunk{defun typeCheckInputAreas}

\getchunk{defun unAbbreviateKeyword}
\getchunk{defun undo}
\getchunk{defun undoChanges}
\getchunk{defun undoCount}
\getchunk{defun undoFromFile}
\getchunk{defun undoInCore}
\getchunk{defun undoLocalModemapHack}
\getchunk{defun undoSingleStep}
\getchunk{defun undoSteps}
\getchunk{defun unescapeStringsInForm}
\getchunk{defun unifyStruct}
\getchunk{defun unifyStructVar}
\getchunk{defun unparseInputForm}
\getchunk{defun untrace}
\getchunk{defun /untrace-0}
\getchunk{defun /untrace-1}
\getchunk{defun /untrace-2}
\getchunk{defun untraceDomainConstructor}
\getchunk{defun untraceDomainConstructor,keepTraced?}
\getchunk{defun untraceDomainLocalOps}
\getchunk{defun untraceMapSubNames}
\getchunk{defun unwritable?}
\getchunk{defun upcase}
\getchunk{defun updateCurrentInterpreterFrame}
\getchunk{defun updateDatabase}
\getchunk{defun updateFromCurrentInterpreterFrame}
\getchunk{defun updateHist}
\getchunk{defun updateInCoreHist}
\getchunk{defun updateSourceFiles}
\getchunk{defun userLevelErrorMessage}

\getchunk{defun validateOutputDirectory}
\getchunk{defun valueArgsEqual?}
\getchunk{defun var}
\getchunk{defun voidValue}

\getchunk{defun what}
\getchunk{defun whatCommands}
\getchunk{defun whatConstructors}
\getchunk{defun whatSpad2Cmd}
\getchunk{defun whatSpad2Cmd,fixpat}
\getchunk{defun whichCat}
\getchunk{defun with}
\getchunk{defun workfiles}
\getchunk{defun workfilesSpad2Cmd}
\getchunk{defun wrap}
\getchunk{defun write-browsedb}
\getchunk{defun write-categorydb}
```

```
\getchunk{defun writeHiFi}
\getchunk{defun writeHistModesAndValues}
\getchunk{defun writeInputLines}
\getchunk{defun write-interpdb}
\getchunk{defun write-operationdb}
\getchunk{defun write-warmdata}
\getchunk{defun writify}
\getchunk{defun writifyComplain}
\getchunk{defun writify,writifyInner}

\getchunk{defun xlCannotRead}
\getchunk{defun xlCmdBug}
\getchunk{defun xlConActive}
\getchunk{defun xlConsole}
\getchunk{defun xlConStill}
\getchunk{defun xlFileCycle}
\getchunk{defun xlIfBug}
\getchunk{defun xlIfSyntax}
\getchunk{defun xlMsg}
\getchunk{defun xlNoSuchFile}
\getchunk{defun xlOK}
\getchunk{defun xlOK1}
\getchunk{defun xlPrematureEOF}
\getchunk{defun xlPrematureFin}
\getchunk{defun xlSay}
\getchunk{defun xlSkip}
\getchunk{defun xlSkippingFin}
\getchunk{defun xSearch}

\getchunk{defun yesanswer}
\getchunk{defun ySearch}

\getchunk{postvars}
```

———-

Chapter 59

The Global Variables

59.1 Star Global Variables

NAME	SET	USE
eof	ncTopLevel	
features		restart
package		restart
standard-input		ncIntLoop
standard-output		ncIntLoop
top-level-hook	set-restart-hook	

59.1.1 *eof*

The `*eof*` variable is set to NIL in ncTopLevel.

59.1.2 *features*

The `*features*` variable from common lisp is tested for the presence of the `:unix` keyword. Apparently this controls the use of Saturn, a previous Axiom frontend. The Saturn frontend was never released as open source and so this test and the associated variables are probably not used.

59.1.3 *package*

The `*package*` variable, from common lisp, is set in restart to the BOOT package where the intepreter lives.

59.1.4 *standard-input*

The `*standard-input*` common lisp variable is used to set the curinstream variable in ncIntLoop.

This variable is an argument to serverReadLine in the intloopReadConsole function.

59.1.5 *standard-output*

The `*standard-output*` common lisp variable is used to set the curoutstream variable in ncIntLoop.

59.1.6 *top-level-hook*

The `*top-level-hook*` common lisp variable contains the name of a function to invoke when an image is started. In our case it is called restart. This is the entry point to the Axiom interpreter.

59.2 Dollar Global Variables

NAME	SET	USE
$boot	ncTopLevel	
coerceFailure		runspad
curinstream	ncIntLoop	
curoutstream	ncIntLoop	
$currentLine	restart	removeUndoLines
$dalymode		intloopReadConsole
$displayStartMsgs		restart
$e	ncTopLevel	
$erMsgToss	SpadInterpretStream	
	SpadInterpretStream	
$frameRecord	initvars	
	clearFrame	
	undoSteps	undoSteps
	recordFrame	recordFrame
$HiFiAccess	initHist	historySpad2Cmd
	historySpad2Cmd	
		setHistoryCore
$HistList	initHist	
$HistListAct	initHist	
$HistListLen	initHistList	
$HistRecord	initHistList	
$historyDirectory		makeHistFileName
		makeHistFileName
$historyFileType	initvars	histInputFileName
$InteractiveFrame	restart	ncTopLevel
	undo	recordFrame
	undoSteps	undoSteps
		reportUndo
$internalHistoryTable	initvars	
$interpreterFrameName	initializeInterpreterFrameRing	
$interpreterFrameRing	initializeInterpreterFrameRing	
$intRestart		intloop
$intTopLevel	intloop	
$IOindex	restart	historySpad2Cmd
	removeUndoLines	undoCount
$genValue	bookvol5	i-toplev
		i-analy
		i-syscmd
		i-spec1
		i-spec2
		i-map
$lastPos	SpadInterpretStream	
$libQuiet	SpadInterpretStream	
$msgDatabaseName	reroot *	
$ncMsgList	SpadInterpretStream	
$newcompErrorCount	SpadInterpretStream	
$newspad	ncTopLevel	
$nopos		SpadInterpretStream
$okToExecuteMachineCode	SpadInterpretStream	
$oldHistoryFileName	initvars	oldHistFileName
$options		history
	historySpad2Cmd	historySpad2Cmd
		undo
$previousBindings	initvars	
	clearFrame	
	recordFrame	recordFrame
$PrintCompilerMessageIfTrue	spad	

59.2.1 $boot

The `$boot` variable is set to NIL in ncTopLevel.

59.2.2 coerceFailure

The coerceFailure symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

59.2.3 $currentLine

The `$currentLine` line is set to NIL in restart. It is used in removeUndoLines in the undo mechanism.

59.2.4 $displayStartMsgs

The `$displayStartMsgs` variable is used in restart but is not set so this is likely a bug.

59.2.5 $erMsgToss

The `$erMsgToss` variable is set to NIL in SpadInterpretStream.

59.2.6 $frameRecord

`$frameRecord = [delta1, delta2,... ]` where delta(i) contains changes in the "backwards" direction. Each delta(i) has the form ((var . proplist)...) where proplist denotes an ordinary proplist. For example, an entry of the form ((x (value) (mode (Integer)))...) indicates that to undo 1 step, x's value is cleared and its mode should be set to (Integer).

A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a a system command is given which alters the environment.

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.

59.2.7 $intRestart

The `$intRestart` variable is used in intloop but has no value. This is probably a bug. While the variable's value is unchanged the system will continually reenter the SpadInterpretStream function.

59.2.8 $intTopLevel

The `$intTopLevel` is a catch tag. Throwing to this tags which is caught in the intloop will restart the SpadInterpretStream function.

59.2.9 $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.

59.2.10 $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.

59.2.11 $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.

59.2.12 $msgDatabaseName

The `$msgDatabaseName` is set to NIL in reroot.

59.2.13 $ncMsgList

The `$ncMsgList` is set to NIL in SpadInterpretStream.

59.2.14 $newcompErrorCount

The `$newcompErrorCount` is set to 0 in SpadInterpretStream.

59.2.15 $nopos

The `$nopos` variable is used in SpadInterpretStream but does not appear to have a value and is likely a bug.

59.2.16 $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.

59.2.17 $okToExecuteMachineCode

The `$okToExecuteMachineCode` is set to T in SpadInterpretStream.

59.2.18 $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.

59.2.19 $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.

59.2.20 $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.

59.2.21 $spad

The `$spad` variable is set to T in ncTopLevel.

59.2.22 $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).

59.2.23 $SpadServerName

The `$SpadServerName` is passed to the openServer function, if the function exists.

59.2.24 $systemCommandFunction

The `$systemCommandFunction` is set in SpadInterpretStream to point to the function InterpExecuteSpadSystemCommand.

59.2.25 top_level

The top_level symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

59.2.26 $quitTag

The `$quitTag` is used as a variable in a catch block. It appears that it can be thrown somewhere below ncTopLevel.

59.2.27 $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.

19 type : FrameName → Symbol
19 frameName : Frame → FrameName
20 frameInteractive : Frame → Interactive
20 frameIOIndex : Frame → IOIndex
20 frameHiFiAcecss : Frame → HiFiAccess
20 frameHistList : Frame → HistList
21 type : HistListLen → NonNegativeInteger
21 frameHistListLen : Frame → HistListLen
21 frameHistListAct : Frame → HistListAct
21 frameHistRecord : Frame → HistRecord
21 frameHistoryTable : Frame → HistoryTable
22 frameExposureData : Frame → ExposureData
22 enum : FrameArgs → (nil,drop,import,last,names,new,next)
22 frameSpad2Cmd : FrameArgs → nil
22 Frame : nil → nil
24 emptyInterpreterFrame : Symbol → Frame
24 emptyInterpreterFrame : Symbol → Frame
25 frameNames : nil → List Symbol
25 displayFrameNames : nil → nil
25 createCurrentInterpreterFrame : nil → Frame
26 updateFromCurrentInterpreterFrame : nil → nil
27 updateCurrentInterpreterFrame : nil → nil
27 frameEnvironment : FrameName → nil
28 findFrameInRing : FrameName → Union(Frame,nil)
28 changeToNamedInterpreterFrame : FrameName → nil
29 nextInterpreterFrame : nil → nil
29 previousInterpreterFrame : nil → nil
32 closeInterpreterFrame : FrameName → nil
77 embed2 : Symbol,Function,Function → Symbol
88 removeOption : Option → List Option
94 flattenOperationAlist : OperationAlist → OperationAlist
109 /untrace-reduce : Union(Atom,List) → Atom
111 transTraceItem : traceArgument → symbol
275 set-restart-hook : Void → 'restart
290 intloopReadConsole : (String Integer) → Throw
295 intloopPrefix? : String → Union(String,NIL)
297 intloopProcessString : (String,StepNo) → StepNo

298 next : (Function,Delay) → Delay
298 next1 : Delay → ParsePair
298 incString : String → Function
301 setCurrentLine : String → List(String)
301 mkprompt : Void → String
303 serverReadLine : Stream → String
321 intloopProcess : (StepNo,Boolean,Delay) → StepNo
329 incRenumber : Delay → Delay
330 incZip : (Function,Delay,Delay) → Delay
330 incZip1 : Delay → ParsePair
330 incIgen : Integer → Delay
332 incLude : (Int,List(String),Int,List(String),List(Int)) → Delay
353 incCommand? : String → Boolean
356 Delay : (Function,List(Any)) → Delay
555 StreamNull : Delay → Union(T,NIL)
1031 get-a-line : FileStream → String
1113 fracpart : or rational float → or rational float
1254 getHtMacroItem : String → Values (String NonNegativeInteger)
1312 bcGen : Command → nil
1312 doDoitButton : String,Command → nil
1313 doDoitButton : Command → nil
1313 executeInterpreterCommand : Command → nil
1486 isWrapped : t → (or t nil)

Bibliography

[Book13] Axiom Authors. *Volume 13: Proving Axiom Correct*. Axiom Project, 2016.

Link: http://axiom-developer.org/axiom-website/bookvol13.pdf

[Burg74] William H. Burge. Stream Processing Functions. *IBM Journal of Research and Development*, 19:12–25, January 1974.

Abstract: One principle of structured programming is that a program should be separated into meaningful independent subprograms, which are then combined so that the relation of the parts to the whole can be clearly established. This paper describes several alternative ways to compose programs. The main method used is to permit the programmer to denote by an expression the sequence of values taken on by a variable. The sequence is represented by a function called a stream, which is a functional analog of a coroutine. The conventional while and for loops of structured programming may be composed by a technique of stream processing (analogous to list processing), which results in more structured programs than the orignals. This technique makes it possible to structure a program in a natural way into its logically separate parts, which can then be considered independently.

[Dalm97] Stéphane Dalmas, Marc Gaëtano, and Stephen Watt. An OpenMath 1.0 Implementation. In *Proc. 1997 Int. Symp. on Symbolic and Algebraic Computation*, ISSAC'97, pages 241–248. ACM, New York, NY USA, 1997, 0-89791-875-4.

Link: http://doi.acm.org/10.1145/258726.258794

[Dewa] Mike Dewar. OpenMath: An Overview.

Link: http://www.sigsam.org/bulletin/articles/132/paper1.pdf

[Fort85] A. Fortenbacher, Richard Jenks, Michael Lucks, Robert Sutor, Barry Trager, and Stephen Watt. An Overview of the Scratchpad II Language and System. Research report, IBM, 1985.

[Jenk81] Richard D. Jenks and Barry M. Trager. A Language for Computational Algebra. In *Proc. Symp. on Symbolic and Algebraic Manipulation*, SYMSAC 1981, 1981.

Abstract: This paper reports ongoing research at the IBM Research Center on the development of a language with extensible parameterized types and generic operators for computational algebra. The language provides an abstract data type mechanism for defining algorithms which

work in as general a setting as possible. The language is based on the notions of domains and categories. Domains represent algebraic structures. Categories designate collections of domains having common operations with stated mathematical properties. Domains and categories are computed objects which may be dynamically assigned to variables, passed as arguments, and returned by functions. Although the language has been carefully tailored for the application of algebraic computation, it actually provides a very general abstract data type mechanism. Our notion of a category to group domains with common properties appears novel among programming languages (cf. image functor of RUSSELL) and leads to a very powerful notion of abstract algorithms missing from other work on data types known to the authors.

Comment: IBM Research Report 8930

[Jenk86] Richard D. Jenks, Robert S. Sutor, and Stephen M. Watt. Scratchpad II: An Abstract Datatype System for Mathematical Computation. Research Report RC 12327 (#55257), IBM Research, 1986.

Abstract: Scratchpad II is an abstract datatype language and system that is under development in the Computer Algebra Group, Mathematical Sciences Department, at the IBM Thomas J. Watson Research Center. Some features of APL that made computation particularly elegant have been borrowed. Many different kinds of computational objects and data structures are provided. Facilities for computation include symbolic integration, differentiation, factorization, solution of equations and linear algebra. Code economy and modularity is achieved by having polymorphic packages of functions that may create datatypes. The use of categories makes these facilities as general as possible.

Link: http://www.csd.uwo.ca/~watt/pub/reprints/1987-ima-spadadt.pdf

[Lisk79] Barbara Liskov, Russ Atkinson, Toby Bloom, Eliot Moss, Craig Schaffert, Bob Scheifler, and Alan Snyder. CLU Reference Manual. Technical report, Massachusetts Institute of Technology, 1979.

[Luke69a] Yudell L. Luke. *The Special Functions and their Approximations*, volume 1. Academic Press, 1969. **Algebra:**
(p**??**) package DFSFUN DoubleFloatSpecialFunctions

[Luke77] Yudell L. Luke. *Algorithms for the Computation of Mathematical Functions*. Academic Press, 1977, 978-0124599406.

[Scha86] C. Schaffert and T. Cooper. An Introduction to Trellis/Owl. *SIGPLAN Notices*, 21(11):9–16, 1986.

[Swee86] Moss E. Sweedler. Typing in Scratchpad II. Scratchpad II Newsletter 2, IBM Research, January 1986.

[Watt87] Stephen M. Watt and Richard D. Jenks. Abstract Datatypes, Multiple Views and Multiple Inheritance in Scratchpad II, 1987.

Abstract: Scratchpad II is an abstract datatype language developed at Yorktown Heights for the implementation of a new computer algebra system. It provides packages of polymorphic functions and parameterized, abstract datatypes with operator overloading and multiple inheri-

tance. To express the intricate inter-relationships between the datatypes necessary for the description of mathematical objects, a number of techniques based on the notion of *category* have been used. Categories are used to enforce relationships between type parameters and to provide the mechanism for multiple inheritance. They also allow the language to be statically type checked and the generation of efficient code. This paper describes the role of categories in Scratchpad II.

Link: https://cs.uwaterloo.ca/~smatt/pub/reprints/1987-itl-spadviews.pdf

[kuki72a] Hirondo Kuki. Complex Gamma Function with Error Control. *Communications of the ACM*, 15(4):262–267, 1972.

Abstract: An algorithm to compute the gamma function and the loggamma function of a complex variable is presented. The standard algorithm is modified in several respects to insure the continuity of the function value and to reduce accumulation of round-off errors. In addition to computation of function values, this algorithm includes an object-time estimation of round-off errors. Experimental data with regard to the effectiveness of this error control are presented. A Fortran program for the algorithm appears in the algorithms section of this issue.

[kuki72b] Hirondo Kuki. Algorithm 421: Complex Gamma Function with Error Control. *Communications of the ACM*, 15(4):271–272, 1972.

Abstract: This Fortran program computes either the gamma function or the loggamma function of a complex variable in double precision. In addition, it provides an error estimate of the computed answer. The calling sequences are: `CALL CDLGAM (X, W, E, 0)` for the loggamma, and `CALL CDLGAM (X, W, E, 1)` for the gamma, where Z is the double precision argument, W is the answer of the same type, and E is a single precision real variable. Before the call, the value of E is an estimate of the error in Z, and after the call, it is an estimate of the error in W.

Index