Compiler Modules

ZADMN0 **(compiler)** **DW +**

The master module for the compiler. Handles compile-time directives. Manages compilation of modules.

ZADMN1 **(compiler)** **DW +**

Manages and eventually compiles the "functors file" (which contains information common to a set of modules).

ZADMN2 **(compiler)** **DW**

Assembles and outputs Macro code generated by the compiler.

ZBLOCS **(compiler)** **DW**

Compiles the "blocks" code which is generated for each procedure to provide access to its clauses. This code does the clause indexing.

ZCLAUS **(compiler)** **DW**

Compiles code for a single clause. Is ultimately responsible for the layout of variables - globals, locals, temporaries.

ZCSTAT **(compiler)** **FP**

Produces compilation statistics as a courtesy to the user.

ZDCG **(compiler)** **DW**

Preprocesses grammar rules into a form suitable for compilation as ordinary clauses.

ZEVAL1 **(compiler)** **DW +**

Generates code for arithmetic expressions in general (including some pseudo-Prolog features).

ZEVAL2 **(compiler)** **DW**

Compiles code for specific binary (2-place) operators (including some pseudo-Prolog operators).

ZEVAL3 **(compiler)** **DW +**

Compiles code for specific unary (1-place) operators, plus some code of dubious correctness for N-place pseudo-Prolog operators.

ZFLAG **(compiler)** **DW**

Translates functors and predicates into corresponding internal codes, addresses etc.

ZGOAL1 **(compiler)** **DW**

Compiles code for the goals which make up the body of a clause. A number of evaluable predicates are treated specially.

ZGOAL2 **(compiler)** **DW**

Handles disjunctions and contains other procedures auxiliary to ZGOAL1. The trace code generated is buggy and obsolete.

ZGOAL3 **(compiler)** **DW**

Compiles pseudo-Prolog control primitives, but not 'if..then..else..' which is in ZGOAL1.

ZLTERS **(compiler)** **DW**

Compiles unification code for terms on the LHS of a clause.

ZMREAD **(compiler)** **DW**

Parses a list of tokens as a term, and returns this term in the special "meta-read" format used by the compiler. Essentially it avoids actually constructing the new functors, by representing the terms in decomposed form as lists. cf. ZREAD.

ZOPSC **(compiler)** **DW**

Remembers user-defined operator declarations using old-fashioned "tag". Incorporates the compiler's standard operators. cf. ZOPSI.

ZRECAL **(compiler)** **DW +**

Implements the stripped-down database facilities ("tag" and "untag") used by the compiler. cf. ZDBASE.

ZRTERS **(compiler)** **DW**

Translates terms into skeletons and also compiles code for the first term in the head of a clause (which is involved in the indexing).

ZSCAN **(compiler)** **FP**

Handles compiler command switches (coming from the user).

ZTMPCO (compiler) FP

Sets up "tempcor" files, so the compiler can call Macro to assemble its output. Defines a predicate 'run(−,−,−)' to call a program from Prolog (via the RUN UUO).

EQS2.MAC (compiler) DW

Defines the Macro symbols required for the assembly of compiler output.

Interpreter Modules

ZDBASE (interpreter) DW

Manages the internal database of interpreted clauses and other recorded items. Implements 'assert', 'retract' etc. Provides access to interpreted clauses.

ZEGALF (interpreter) DW

Implements the evaluable predicates '= =' and '\= =' (formal identity, non-identity of terms).

ZENCOD (interpreter) DW

Encodes the body of an interpreted clause as a special term to reduce the space occupied.

ZKNOW (interpreter) DW

The heart of the interpreter. Executes goals, evaluates arithmetic expressions, supports the (about to be superseded) tracing facilities.

ZLIST (interpreter) FP

Implements the listing facilities (and contains some goodies for enumerating current atoms and functors).

ZOPSI (interpreter) DW

Remembers user-defined operator declarations and incorporates the interpreter's standard operators. cf. ZOPSC.

ZREAD (interpreter) DW

Implements evaluable predicate 'read'. Parses a list of tokens according to current operator declarations, and constructs the corresponding term. cf. ZMREAD.

ZSAVE (interpreter) FP

Provides interface to Macro routines for evaluable predicates 'save' and (?) 'morespace'.

ZSTATS (interpreter) FP

Produces statistics for the interactive user.

ZSVWX (interpreter) DW +

The master module for the interpreter. Is responsible for 'consult'-ing files, for executing directives, and for translating grammar rules into ordinary clauses (for the last, cf.ZDCG).

ZUNIV (interpreter) DW

Implements evaluable predicates '=..' (alias "univ") and 'functor'.

ZWRITE (interpreter) DW

Implements evaluable predicate 'write'.

CTRAP.MAC (interpreter) FP

Handles control C interrupts.

SAVE.MAC (interpreter) FP

Implements 'save' and 'restore'.

ZNOEXT.MAC (interpreter) DW

Dummy module needed to create the naked interpreter (unextended by user-defined compiled modules).

Common Modules

ZINHTA (common) DW

Initialises the hash table, functor table and properties table (which are used inter alia by ZNAME and ZUNIV)..

ZIOCTL (common) FP

Provides interfaces for input-output evaluable predicates implemented in Macro.

ZMISC (common) DW

Odds and ends called from various places.

ZNAME (common) DW

Implements 'name'. Maintains correspondence between atoms and their names via a hash table.

ZSYNER (common) DW

Informs the user of the location of a syntax error. Supports ZREAD, ZMREAD.

ZTOKE (common) DW

Inputs a string of characters (terminated by "full-stop") and maps them into a list of tokens. Is responsible for keeping a symbol table of variables occurring in the token list. Has been drastically doctored to speed the beast up as much as possible; there is an older more readable version. Supports ZREAD, ZMREAD.

IOLIB2.MAC (common) FP

Implements the evaluable predicate 'display'.

Basic Modules

GARBGE.MAC (basic) FP

Garbage collector for the global stack. Called from SHIFT.MAC.

IDENS2.MAC (basic) DW

Symbol definitions for PLLIB2.MAC.

IDENSF.MAC (basic) FP

Symbol definitions for other Macro modules.

MHEAP.MAC (basic) FP

Manages the heap used by database evaluable predicates. Requests more space from the freespace routines in PLRUN2.MAC.

PLINI2.MAC (basic) FP + DW

After initialising various locations, initiates Prolog execution. Also sets traps for pushdown list overflow.

PLLIB2.MAC (basic) DW

Contains the Macro routines which support the compiled code. The bulk of the code is to handle special cases of unification, and is tuned for maximum speed. Routine UNIF handles the most general case of unification, and is commented (!). The other unification routines are variants of UNIF. PLLIB2 also contains code for some very basic evaluable predicates, such as '=' and 'arg'.

PLMAC2.MAC (basic) FP

Contains Macro macros used by the garbage collector etc.

PLRUN2.MAC (basic) FP

INISTA - Manages various locations which keep run-time statistics.
COREIM - Implements the evaluable predicate 'core—image'.
UUOLIB - Interfaces several UUOs to Prolog; also computes core usage (used by ZSTATS, ZCSTATS and others).
LUUOS - Local UUO handler.
LOCORE - Handles core allocation for the stacks segment.
SPACE - Manages the data segment (used for buffers, database heap and maybe other purposes...).
SIXOUT - Outputs on the TTY a sixbit quantity.
IOLIB - Macro code for the I/O evaluable predicates.
STATIC - Interfaces a LUUO (SPACE) to the routines in SPACE above (!).
PAGES - Is this really needed? (Well if you don't know, who does? :DW.).

PROLOG.BLI (basic) FP

Contains Bliss macros and symbol definitions.

SHIFT.MAC (basic) FP

Dynamically re-allocates space between the different stacks. Handles stack overflows.

TRIM.MAC (basic) FP

Interface for the Bliss implementation of the 'trimcore' routine.

TRIMCO.BLI (basic) FP

Reduces the free space on the stacks to a minimum (relocating the stacks) and reduces the total core allocation.