1. System organization

The modules which make up this system are divided into three sections: common, interpreter, and compiler.

- Common section:
  (defined in common.ccl)

  Put in this section any module that satisfies any of the following conditions:
  * Contains writable (i.e., low segment) locations
  * Contains entry points for evaluable predicates or support routines called from in-core compiled code
  * Contains entry points for run-time support routines for in-core compiled code
  * Contains predicates used by modules in both of the other sections

- Interpreter section:
  (defined in interp.ccl)

  Put in this section any module which is used only to support the running of user (interpreted or in-core compiled) programs, and which is not in the common section. Also, put here the system starting point (currently in file PL112.RAC) and system initialization procedures (like those in program J1HTAB.PL).
- Exchange section
  (defined in exchan.ccl)

Contains the module needed for basic segment exchange routines. It is definitely unhealthy to try to change these modules.
2. Loading

When appropriately loaded, the three sections merge into two executable files, `vfoo.exe` and `vplcomp.exe`. Note that the latter file must be called exactly that, `vplcomp.exe` and be placed exactly in the same place as the other, otherwise the system will not work, and the Prolog system will complain in no uncertain terms.

To start a run of Prolog, just type `R(VN) foo`, where `foo` is the name of the interpreter segment file.

To load the system, several `.mic` files are available:

- `INT.MIC` loads the interpreter segment, and saves it as `SCRA:PLCOMP.EXE`. This 

- `DINT.MIC` loads the interpreter segment with symbols (in the lower segment) and saves it as above.

- `COMP.MIC` loads the compiler segment and saves it as `SCRA:PLCOMP.EXE`. This

- `DBOTH.MIC` makes a combined one segment system, with symbols to simplify debugging, and saves it as `SCRA:DBOTH.EXE`. This
Notes on loading the system:

* Do not panic if some undefined globals appear when executing INT, DINT or CORP. This is expected. However, if *ANY* symbols (atom variables) appear as undefined, or that is serious. You have somehow miscompiled some PROLOG module. (Multiply defined symbols are not expected, and they reveal something quite wrong (maybe you forgot the main module or produced a new program module).

* If you need EDT flag, it is simpler to change using the combined system, loaded by DBOT+X. If, however, you really want to have fun, try the two-segment debug system (DINT+CORP). Note that some bug in the production system can go away in any of the debugging systems, as the memory layout cannot be exactly the same. Furthermore, with the DINT+CORP system: do not even know about compiler only symbols.

  - Doesn't know about symbols in the compiler section are not known
  - Symbols are in the low segment, so compare the segment swap; they may thus belobbered by a great education
  - If you set a breakpoint when executing the interpreter segment, even in the common section (module), it may (can't) still be there unless segments are swapped; to be on the safe side, always...
If some of the undefined globals seem definitely wrong, try to do the DBT/IT load. If they disappear, then either they are ok after all, or some module that should be in the common section that has been put somewhere else. Check it.
execute the exchange routine (entry $nruseg) in single step mode and reset break points somewhere after the crucial call...

instruction, which tells the monitor to get the other segment

3. Changing the system

If you (lucky boy!) invent a new module, first classify it according to the classification in 1. Then edit the relevant ccl file (also defined in 1.) and add a new line with the name of the new module

Try to keep the common section as small as possible, which makes the EXE files smaller (and the system at runtime!) smaller and speeds up segment exchange, by minimizing the stripping of common code. If your module is in the common section because it contains variables locations, mark them put these locations in the common section as globals and delete them from your module. If the new module contains an evaluable predicate, make two modules of from it in the case it contains much more code than just the clauses for the evaluable predicate. The new part which doesn't contain the eval predicate may not need to go into the common section.
To get a Pascal system with symbols

.R LINK
*/SYMSEG: HIGH ; symbols to hiseg
*/ (our modules, etc.)
* program /SAVE
* /GO

To sum it

.RUN program
- To have DDT from the start

. GET Program 2
. DDT 2

  < set breakpoints, etc. >

④G

- To get DDT in the middle of a session

1C to get PDSlog interrupt routine
Function (h for help): m
. DDT
  < set breakpoints, etc. >
JNST @.JBDPC③X to continue

- To get DDT after a bomb-out

. DDT 2

  < examine locations, etc. >
Shift/GC Strategy

allocations for
initial stacks

global: 1K + (something < 1 page)
local: 1K
trail: 1/2 K

α: cost ratio for moving trail: 1/10 \{ changed to 1/3 on 260478 \}
β: " " local stack: 1/10
γ: " " updating: 1/10

trail full: increment = localsize * γ
local stack full: increment = trailsize * α
global stack full: increment = localsize * β + trailsize * α

garbage collection:

successful garbage collection: reclaim > 1K

\[ S = \max \{ \text{max of successful GCs}, 1 \} \]
\[ N = \text{max of GCs} \]

Do a GC each the \( \frac{LN}{\text{NLth}} \) global overflow
GC : requires :
X, VV pointing at the chain of local frames
X pointing at the last local frame
V
VV " " " top of local stack  
V1 " " " " " global " "
PR " " " " " trail
all accessible from local & global frames complete
all trail pointers pointing at accessible stack locations

updates :
global stack
V1 pointers in local stack, global references
V1
VV1

Global stack of lo : requires : X, V, VV, V1 and TR as above
accessible local under stack frames as above
updates : local stack (shifting V and sometimes
$LOSE $VMAX $V1MAX
$TRB $TRTOP
$TR512 (sometimes)
TR , $TRO
V , VV , X

Assumes : Update

GSO N o G C
L S O
T R O