Literate Programming: Prolog Documentation with LaTeX

Gerd Neugebauer
Mainzer Str. 8
56321 Rhens (Germany)
Net: gerd@informatik.uni-koblenz.de

This document describes pl version 3.0 as of 1996/05/30.

Introduction

Inspired by the idea behind the web system I felt the need to have a system to write documented Prolog programs. But instead of having to transform the common source into program or documentation the central idea was to develop a method to have one common source which can be interpreted by a Prolog[1] system aswell as by LaTeX. To achive this goal the LaTeX commands are hidden from Prolog by enclosing them into comments.

The C-Prolog allows two kinds of comments. The first kind starts with a `%` and ends at the end of the line. This is also a comment for LaTeX. The other Prolog comment starts with `/*` and ends with `*/`. This seemed to be the right way to hide the LaTeX documentation commands from the Prolog interpreter.

The only problem was to define a starting sequence to open the first comment. This sequence must be an executable Prolog command as well as a valid LaTeX command. The first approach was to modify both — the Prolog program and the LaTeX style — to find a common statement. The pre 2.0 versions of the pl style option took this decision. One problem came up when the author tried to incorporate the module system of Prolog. This system requires that the first command in a module is the declaration of the module name possibly together with the list of predicates to be exported. This restriction led to the approach taken in version 2.0 and later. In modules nearly no change has to be made. The module declaration is defined as a valid LaTeX macro.

1 The LaTeX Documentation Driver File

The LaTeX main file ties together the whole documentation. This file should contain the `\documentclass` command together with the appropriate `\usepackage` or the `\documentsyle` command with the style option `pl`.

[1] C-Prolog, Quintus-Prolog, or ECLiPSe at that time.

The only special command required in this file is the command `\PrologInput`. This command inserts the Prolog file given as argument at the current position. The formatting directives described below are activated.
The command `\PrologInput` may occur several times in this main file. it may be mixed with simple input or include commands. `\PrologInput` can be used nested, as long as the other requirements described in this document are fulfilled.

```
\documentclass[...]{...}
\usepackage{pl}
\begin{document}
...
\PrologInput{file1.pl}
\PrologInput{file2.pl}
...
\PrologInput{filen.pl}
...
\end{document}
```

The commands meant for the Prolog files can also be used in a plain LaTeX context. Nevertheless those commands have been developed with the application of documenting Prolog in mind. The contents of the Prolog files and the additional macros provided by pl will be described in the next sections.

2 The Prolog Files

2.1 Starting Modules

Prolog files can come in two variants: either it is a Prolog module or a plain Prolog file. Modules are distinguished from files by a specific first Prolog statement. This module declaration can have the form[2]

:- module(Name, Exports).

Alternatively it may have the form

:- module_interface(Name).

These module declarations have to be the first Prolog instructions in the file.

In addition to those Prolog instructions we need to use the rule that the terminating point (.) of the module declaration is followed immediately by ␣`/*`. Note the single space which is neccesary for the Prolog parser to work properly.

Thus a Prolog module documented with pl starts e.g. with

*:- module_interface(Name). /**

Before this line there should be only Prolog comments starting with `%`. These comments are also ignored by TeX. Thus they do neither appear in the documentation nor are the evaluated by Prolog.

2.2 Starting Plain Files

Plain Prolog files are those files not starting with a module declaration. For technical reasons plain files can not start with code directly but requires a C-style comment at the beginning. The contents of this comment is typeset by LaTeX.

In generale this restriction seems to be too hard. It is always a good idea to start a file with some general remarks and comments.

[2] e.g. in Quintus-Prolog

2.3 Ending Prolog Files

The Prolog file should end with

```
\EndProlog*/
```

From the Prolog point of view the file consists now of the declaration of a module (or a dummy predicate). The rest is purely comment. From the LaTeX point of view it contains two macros and nothing in between. Thus everything in between will be interpreted as LaTeX input only.

2.4 Prolog Code

To include additional Prolog statements in this file enclose them in

```
\PL*/
/*PL
```

This forces the Prolog system to interpret the code and the LaTeX system to typeset it in a verbatim like environment.

The Prolog code may contain every characters except the string `/*PL`. For some reasons which are opaque to me spaces at the binning of a line of Prolog code are ignored. To force proper indentation you should use TAB characters at the beginning of the line.

Options for the Code Layout

The options are implemented as macros. To change them use `\renewcommand`.

`\PrologModule`
: Macro to typeset the module definition. In general this may be a sectioning command producing an appropriate title. It takes two arguments — the two arguments of the `module` declaration.

  The default is `\section{{\tt #1}}`

`\PrologFile`
: Same as above but for non module files.

  The default is `\section{{\tt #1}}`

`\PrologFont`
: Macro to select the font used for printing the listing part. This should be a non-proportional font. The default is `\small\tt`.

`\PrologListFont`
: Macro to select the font used for printing the export list. The default is `\small\tt`.

`\PrologListIndent`
: Macro to select the indentation of the export list. This should be a length. The default is `2em`.

`\PrologRuleWidth`
: Macro to select the width of the rule to seperate Prolog code from text. This should be a length. The default is `0pt`.

`\PrologNumberFont`
: Font to be used when typesetting line numbers.

The following flags can be set by simply including the commands in the LaTeX part of the document. Since they are mainly of a global nature the preamble of the driver file would be a good place for them.

`\PrologNumberLinestrue`
: Turn on line numbering.

`\PrologNumberLinesfalse`
: Turn off line numbering.

`\PrologDialect`*`Dialect`*
: Declare the dialect of the used Prolog. This is mainly important to make the syntax of the modules known to pl. *Dialect* can take one of the following values: `eclipse`, `quintus`, `sixtus`, `cprolog`, `swiprolog`, `sbprolog`, or `binprolog`.

3 Predicate Templates

Predicate templates provide a nice way to typeset a predicate head with some additional information. A usual predicate is characterized by the name/arity, arguments and the file it can be found in. Two boxes are used to contain this information. The right one contains the file name and the left one contains the predicate description. The box for the file name has a default width which will be enlarged if the file name doesn't fit into it. The predicate description if surrounded by a frame and formatted in the following way. If the predicate desciption fits in one line then it is typeset in one line. Otherwise the second and following lines are indented. This indentation tries to align beneath the first argument. If the predicate name is very long this might not be desirable. Thus the indentation has a maximal value which will not be exceeded.

The LaTeX part of the file contains the following template

`\Predicate pred/1(arg).`

Note that the `).` have *no* space in between.

Options

The options are implemented as macros. To change them use `\renewcommand`.

`\PredicateFont`
: Macro containing the font changeing command for the predicate description. The default is `\normalsize\tt`.

`\PredicateSkip`
: Macro containing the spacing before and after the predicate description. The default is `\smallskip`.

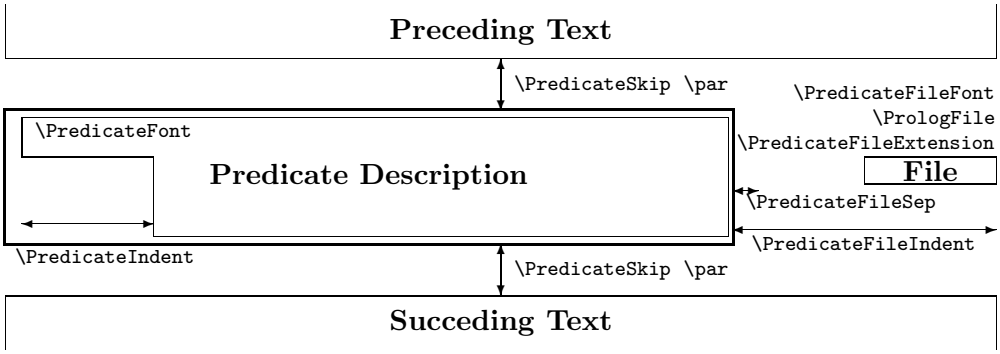

Figure 1: The `\Predicate` command

`\PredicateIndent`
: Macro containing the maximal length of the indentation of the predicate description. The default is `2em`. If this value is very large then the arguments are always aligned beneath the first one.

`\PredicateFileFont`
: Macro containing the font changeing command for the file name. The default is `\small\sf`.

`\PredicateFileWidth`
: Macro containing the minimal width of the box containing the file name. The default is `5em`. If this value is `0pt` then only the file name determines the width of the box.

`\PredicateFileExtension`
: Macro containing the extension of the prolog file. This text is typeset right after the file name stored in `\PrologFILE`. The default is empty.

```
\Predicate foo_bar/6(Argument1, Argument2, Argument3,
                     Argument4, Argument5, Argument6).
```

```
foo_bar(Argument1, Argument2, Argument3, Argument4,
        Argument5, Argument6)
```
prolog

```
foo_bar(Argument1, Argument2, Argument3,
        Argument4, Argument5, Argument6)
```
very_Long_Prolog_File

```
\Predicate this_is_a_very_long_predicate/5(
    Argument1, Argument2, Argument3, Argument4, Argument5).
```

```
this_is_a_very_long_predicate( Argument1, Argument2,
          Argument3, Argument4, Argument5)
```
prolog

3.1 The Underscore

One thing which occurs very often in program listings is the underscore `_`. The problem arises that LaTeX uses the underscore only in math mode to denote

subscripts. For the comands described above the _ can be simply used as is. Beware, don't expect math mode subscript to work there.

The macro `\WithUnderscore` is provided by pl to execute some commands where the underscore does have the meaning as plain character.

Example:

It can be highly desirable to protect the index. Otherwise any indexed predicate containing an underscore would wrack LaTeX. This can be done with a construction like the following one:

```
\WithUnderscore{\printindex}
```

3.2 Inline Code

The style option `idtt` allows to typeset some text using the teletype font (`\tt`). The text has simply to be enclosed in `|`.

The same effect can be achieved with the command `\MakeShortVerb` defined in doc.sty. To use this you have to add the style option `doc` and place the following command in the preamble:

```
\MakeShortVerb{|}
```

Example:

You type `|proc_1|` and you get `proc_1`.

Bugs and Problems

- The bugs and problems of earlier releases have been corrected.
- The documentation needs polishing.

A Sample

```
%%^^A%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%^^A This is a sample file to demonstrate the use of the \LaTeX style option
%%^^A pl.sty.
%%^^A
%%^^A The ^^A is just used to make it printable with the documentation.
%%^^A doc.sty insists on it. Otherwise a single % would have been enough.
%%^^A
%%^^A written by gene 11/94
%%^^A%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- module(sample). /*

This is a dummy module to show the possibilities of the \LaTeX{} style
option pl.
We define a predicate. It looks like

\Predicate select/3(Member, List, Rest).

This predicate describes the relation of the three arguments which fulfill
$\mbox{\it Member}\in\mbox{\it List}$\/ and $\mbox{\it Rest}=\mbox{\it
List}\backslash\mbox{\it Member}$.

And here comes the implementation:
\PL*/
select(Member,[Member|Rest],Rest).
select(Member,[Head|List],[Head|Rest]) :-
        select(Member,List,Rest).
/*PL

\Predicate in/2(Member, List).

This predicate is a reimplementation of the predicate \verb|member/2|
using the \verb|select/3| predicate.

\PL*/
in(Member,List)  :-
        select(Member,List,_).
/*PL
Now we are done with the example.
\EndProlog*/
```

4 The Module `sample.pl`

This is a dummy module to show the possibilities of the LaTeX style option pl. We define a predicate. It looks like

```
select(Member, List, Rest)
```
sample.pl

This predicate describes the relation of the three arguments which fulfill $Member \in List$ and $Rest = List \backslash Member$.

And here comes the implementation:

```
:- module(sample).
select(Member,[Member|Rest],Rest).
select(Member,[Head|List],[Head|Rest]) :-
        select(Member,List,Rest).
```

```
in(Member, List)
```
sample.pl

This predicate is a reimplementation of the predicate `member/2` using the `select/3` predicate.

```
in(Member,List)  :-
        select(Member,List,_).
```

Now we are done with the example.

5 The Implementation

```
\ifx\documentclass\relax \else
  \ProvidesPackage{pl}[\filedate\space gene (\fileversion)]
\fi
```

5.1 Options and Defaults

First of all we define the macros containing the default values for certain options. The user may redefine them with `\renewcommand` to adapt them to the personal preferences.

```
\def\PrologFont{\small\tt}
```

The macro `\PrologFont` contains the font changing command executed to typeset the Prolog code in the verbatim-like environment.

```
\def\PrologIndent{2em}
```

The macro `\PrologIndent` contains the indentation of the Prolog code in the verbatim-like environment.

```
\def\PrologNumberFont{\tiny\rm}
```

The macro `\PrologNumberFont` contains the font changing command used to typeset the line numbers (if enabled) in the verbatim-like environment.

```
\def\PrologRuleWidth{0pt}
```

The macro `\PrologRuleWidth` contains the width of the rule seperating Prolog code and text.

```
\def\PrologListFont{\small\tt}
```

The macro `\PrologListFont` contains the font changing command to typeset a Prolog list (exports).

```
\def\PrologListIndent{2em}
```

The macro `\PrologListIndent` contains the indentation for a Prolog list (exports).

```
\def\PrologModule#1#2{\section{The Module {\tt #1}}}
```

The macro `\PrologModule` contains the command which is called at the beginning of a module. The first argument is the name of the module. The second argument is the list of exports.

This macro is supposed to be redefined by the user with `\renewcommand`.

```
\def\PrologFile#1#2{\section{The File {\tt #1}}}
```

The macro `\PrologFile` contains the command which is called at the beginning of a non-module Prolog file. The first argument is the name of the module. The second argument is usually empty.

This macro is supposed to be redefined by the user with `\renewcommand`.

```
\def\PredicateFont{\tt}
```

The macro `\PredicateFont` contains the font switching command used in `\Predicate` for the body of the predicate description.

```
\def\PredicateFileFont{\small\sf}
```

The macro `\PredicateFileFont` contains the font switching command used in `\Predicate` for the file name

```
\def\PredicateSkip{\smallskip}
```

The macro `\PredicateSkip` contains the skip command executed before and after `\Predicate`.

```
\def\PredicateIndent{5em}
```

The macro `\PredicateIndent` contains length of the maximal indentation in the macro `\Predicate`.

```
\def\PredicateFileExtension{}
```

The macro `\PredicateFileExtension` contains the text appended to file name in the `\Predicate` command.

```
\def\PredicateFileWidth{5em}
```

The macro `\PredicateFileWidth` contains the minimum width of the file name in the `\Predicate` command. Shorter file names are padded to this length.

```
\def\PredicateFileSep{1em}
```

The macro `\PredicateFileSep` contains the width of the separating space between the predicate description and the file name.

```
\def\PredicateBoxSep{3pt}
```

The macro `\PredicateBoxSep` contains the amount of space between the box and contents in the `\Predicate` command. This is similar to `\fboxsep` in LaTeX.

```
\def\PredicateBoxRule{0.5pt}
```

The macro `\PredicateBoxRule` contains the line thickness of box in the `\Predicate` command. This is similar to `\fboxrule` in LaTeX.

```
\def\PredicateIndex#1{\index{#1}}
```

The macro `\PredicateIndex` contains the command which is called to put a predicate into an index. The argument is the string to index.

This macro may be redefined by the user with `\renewcommand`.

```
\newif\ifPrologNumberLines
```

switch to enable line numbering

5.2 Internals

```
\def\PrologEXPORTS{}
```

The macro `\PrologEXPORTS` contains the current list of exports.

```
\def\PrologFILE{}
```

The macro `\PrologFILE` contains the current module or file name. This is not supposed to be set by the user. Nevertheless there might be occasions where this is neccesary (e.g. in the documentation of this style option).

5.3 Configuration Commands

The different Prolog dialects have different ways to declare modules. Thus pl needs to know which dialect is currently used. This influences how `PrologInput` handles the first Prolog clause.

```
\def\PrologDialect#1{%
  \@ifundefined{PL@start@module@#1}%
                {\message{*** Prolog dialect #1 is undefined. Ignored.}}%
                {\gdef\PL@Dialect{#1}}}
\def\PL@Dialect{eclipse}
```

The command `\PrologDialect` can be used to declare the Prolog dialect used. The value is stored in the macro `\PL@Dialect` for later use. This is only done if an appropriate macro to handle a module are defined.

```
\gdef\PL@@delayed{}
```

Keep some characters which were read in advance.

```
\newcount\PL@line
```

We allocate a new counter for the line number of Prolog code (if enabled).

5.4 Typesetting Prolog Code

Prolog code is typeset is a verbatim-like way. For this purpose a modified version of the verbatim environment from the TEX book is used. For an explanation see pages 380–382 in the TEX book.

```
\gdef\PL@code@setup{\PrologFont\parskip=0ex\parindent=0pt
  \ifx\PL@@delayed\empty\else%
    \parbox{\PrologIndent}{%
      \ifPrologNumberLines \PrologNumberFont \the\PL@line%
      \global\advance\PL@line1
      \else\ \fi}\PL@@delayed%
    \gdef\PL@@delayed{}\par
  \fi%
  \def\par{\leavevmode\egroup\box0\endgraf}
  \def\do##1{\catcode`##1=12 }\dospecials
  \obeyspaces
  \obeylines
%  \catcode`\`=\other
  \catcode`\^^I=13
  \everypar{\parbox{\PrologIndent}{%
      \ifPrologNumberLines \PrologNumberFont \the\PL@line%
```

```
      \global\advance\PL@line1
      \fi
      \hfill}\PL@code@startbox}}
\def\PL@code@startbox{\setbox0=\hbox\bgroup}
{\catcode`\^^I=13
 \gdef^^I{\leavevmode\egroup
   \dimen0=\wd0 % the width so far, or since the previous tab
   \setbox1=\hbox{\PrologFont\space}\dimen1=8\wd1
   \divide\dimen0   by\dimen1
   \multiply\dimen0 by\dimen1 % compute previous multiple of tab
   \advance\dimen0  by\dimen1 % advance to next multiple of tab
   \wd0=\dimen0 \box0 \PL@code@startbox}%
}
{\obeyspaces\global\let =\ }
\def\PL*/{\PL@PL@init%
  \begingroup
  \PL@code@setup
  \PL@doPL}
```

| is temporary escape character to catch the end /*PL

```
{\catcode`\|=0 \catcode`\\=12
  |obeylines|gdef|PL@doPL^^M#1/*PL{#1|endgroup|PL@PL@exit}}
```

Initialization macro

```
\def\PL@PL@init{%
        \ifdim\PrologRuleWidth>0pt%
          \par\noindent\rule{\textwidth}{\PrologRuleWidth}\par%
        \else\medskip\par\fi}
\def\PL@PL@exit{%
        \ifdim\PrologRuleWidth>0pt%
          \vspace{-2ex}\noindent\rule{\textwidth}{\PrologRuleWidth}\par%
        \else\smallskip\par\fi}
```

Dirty hack. make : active to catch :- use a group to hide the changes

```
\def\PL@INIT{\begingroup\catcode`:=13\catcode`/=13}
\def\PL@EXIT{\endgroup}
```

we make : active and include the file

```
\gdef\PrologInput{%
  \begingroup
  \catcode`\_=12
  \PL@Input
}
\PL@INIT
\gdef\PL@Input#1{%
  \gdef\PrologFILE{#1}%
  \gdef\PrologMODULE{}%
  \gdef\PrologEXPORTS{}%
  \global\PL@line=1%
  \endgroup
  \PL@INIT%
```

```
  \let:=\PL@COLON
  \let/=\PL@SLASH
  \input{#1}%
  \gdef\PrologFILE{}%
  \gdef\PrologMODULE{}%
  \gdef\PrologEXPORTS{}%
  }
\PL@EXIT
```

5.5 End a File of Prolog Code

At the end of a file there is a `*/` which terminates the last comment for Prolog. Those two characters are stripped away by the macro `\EndProlog`.

```
\def\EndProlog#1*/{}
```

5.6 Definition for Various File Types

5.6.1 Defininitions for Non-Module Files

A file can start with `/*`. This case is handled by making the `/` active and binding it to the command `\PL@SLASH`. This command checks if the next character is a `*`. In this case the catcodes of `/` and `:` can be restored to their defaults. This is done by `\PL@EXIT`.

Finally the underscore `_` is made active and `\PL@start@star` is called to do the rest.

```
\def\PL@SLASH{\@ifnextchar*{%
    \PL@EXIT
    \PL@US@start
    \PL@SLASH@STAR}{/}}
\def\PL@SLASH@STAR*{%
  \PrologFile{\PrologFILE}{}%
  \PL@US@end}
\def\PL@COLON{\@ifnextchar-{\PL@goal}{: }}
\def\PL@goal-{%
  \PL@EXIT
  \PL@US@start
  \@ifnextchar m{\csname PL@start@module@\PL@Dialect\endcsname}%
  {\@ifnextchar t{\PL@start@true}%
    {\csname PL@start@file@\PL@Dialect\endcsname}}}
\def\PL@start@true true. /*{%
  \PrologFile{\PrologFILE}{}%
  \PL@US@end}
```

5.6.2 Defininitions for ECLiPSe-Prolog

The beginning of a module file in eclipse can be in one of the following forms:

:- module_interface(*Module*)

:- module(*Module*)

We strip away the actual predicate name getting the rest in the macro parameter #1. The complete module declaration is stored in the macro `\PL@delayed` to be inserted later.

```
\def\PL@start@module@eclipse module#1(#2). /*{%
  \global\PL@line=1
  \gdef\PL@@delayed{:- module#1(#2).}
  \gdef\PrologMODULE{#2}%
  \catcode`\,=13 %
  \PrologModule{\PrologFILE}{}%
  \PL@US@end}
```

5.6.3 Definitions for Quintus-Prolog

The beginning of a module file in Quintus is in the following form:

:- module(*Module*,*Exports*)

```
\def\PL@start@module@quintus module(#1,{%
  \global\PL@line=1
  \gdef\PrologFILE{#1}%
  \catcode`\,=13 %
  \PL@start@module@quintus@}
\def\PL@start@module@quintus@[#1]). /*{%
  \gdef\PrologEXPORTS{#1}%
  \PrologModule{\PrologFILE}{#1}%
  \PL@US@end}
```

5.6.4 Definitions for C-Prolog

I don't know if C-Prolog has a module system nowadays. The last time I checked it had none. If nobody has a better idea I use Quintus Prolog syntax in this case even it does not make any sense.

```
\let\PL@start@module@cprolog=\PL@start@module@quintus
```

5.6.5 Definitions for Sixtus-Prolog

I don't know what's used in Sixtus-Prolog. So I use the same value as for eclipse.

```
\let\PL@start@module@sixtus=\PL@start@module@eclipse
```

5.6.6 Definitions for SWI-Prolog

Fortunately the module system of SWI-Prolog is compatible with the module system of Quintus Prolog. So we just use the definition here again.

```
\let\PL@start@module@swiprolog=\PL@start@module@quintus
```

5.6.7 Definitions for SB-Prolog

I don't know if C-Prolog has a module system nowadays. The last time I checked it had none. If nobody has a better idea I use Quintus Prolog syntax in this case even it does not make any sense.

```
\let\PL@start@module@sbprolog=\PL@start@module@quintus
```

5.6.8 Definitions for bin-Prolog

I don't know what's used in bin-Prolog. So I use the same value as for eclipse.

```
\let\PL@start@module@binprolog=\PL@start@module@eclipse
```

5.7 Typeset a Boxed Predicate Description

The first step is to protect the underscores in the predicate names and the arguments.

```
\def\Predicate{\PL@US@start\Predicate@}
```

The syntax is oriented towards Prolog syntax. The name and the arity of the predicate may not contain `/` or `(`.

```
\def\Predicate@#1/#2(#3).{%
  \PredicateSkip\par\noindent%
  {\setbox1=\hbox{\PredicateFileFont \PrologFILE\PredicateFileExtension}%
   \fboxrule=\PredicateBoxRule%
   \fboxsep=\PredicateBoxSep%
   \fbox{\PredicateIndex{#1/#2}%
         \dimen255=\wd1
         \ifdim\dimen255<\PredicateFileWidth \dimen255=\PredicateFileWidth \fi
         \dimen255=-\dimen255
         \advance\dimen255 by-\PredicateFileSep
         \advance\dimen255 by \textwidth
         \parbox{\dimen255}{\raggedright
                 \setbox0=\hbox{\normalsize\PredicateFont #1(}
                 \dimen254=\wd0

                 \ifdim\dimen254>\PredicateIndent \dimen254=\PredicateIndent\fi
                 \dimen253=\dimen255 \advance\dimen253 by -\dimen254
                 \parshape=2 0mm \dimen255 \dimen254 \dimen253
                 \normalsize\PredicateFont #1\ifx\@empty#3 \else(#3)\fi
        }}%
    \hfill \box1\PredicateSkip\par
  }\PL@US@end}
```

5.8 Typeset a List of Prolog Predicates

```
\def\PrologList{\par\noindent%
  \PL@US@start
  \PrologListFont
  \catcode`\,=13%
  \parindent=\PrologListIndent\parskip=0pt\par
  \PL@List}
```

```
{\catcode`\,=13
\gdef\PL@List[#1]{%
  \def,{\par}%
  #1
  \PL@US@end\par}
}
\def\PrologListEXPORTS{\PrologList[\PrologEXPORTS]}
```

5.9 Special Treatment of the Underscore

We define two macros to activate and deactivate the underscore respectively. Those two macros have to come in pairs always. The first one opens a group to protect the changes. The closing macro simply closes the group, thus undoing the effects of the first macro.

```
\def\PL@US@start{\begingroup\catcode`\_=13 }
\def\PL@US@end{\endgroup }
\def\WithUnderscore{\begingroup\catcode`\_=13 \With@Underscore}
\def\With@Underscore#1{#1\endgroup}
```

5.10 Misc

A spin off product of this style file is a macro to include a file verbosely. Such a macro is also provided by the verbatim package and others. Nevertheless I have left it in.

```
\def\Listing#1{\par\begingroup%
  \PL@line=1%
  \PL@code@setup%
  \input{#1}%
  \endgroup}
```

6 Backward Compatibility Mode: `pcode.sty`

For backward compatibility some macros are defined in a style file under the old name `pcode.sty`. The new style file has to be acessible under the new name `pl.sty`. This file is loaded before some macros are defined.

```
\ifx\PrologFont\relax\else\input pl.sty\fi
```

In former versions mainly the Prolog dialect Quintus has been supported. Thus a boolean was enough to tell apart the dialects Quintus and eclipse — which came next. This is emulated with the next two macros.

```
\def\PrologQuintustrue{\PrologDialect{quintus}}
\def\PrologQuintusfalse{\PrologDialect{eclipse}}
```

The macro `\WithUnderscore` was named `\WithActiveUnderscore` in a former version of `pcode.sty`. The *Active* part of the name was missleading for users. Thus it has been removed. The old name is made an alias for the new one.

```
\let\WithActiveUnderscore=\WithUnderscore
```

In an ancient version of `pcode.sty` the macro `\EndProlog` was named `\StopProlog`. Thus we make an alias for the old name.

```
\let\StopProlog=\EndProlog
```

Make `\PL@INIT` usable for the user as well. I don't know where this might be used. It has been in the old version, so I put it into the compatibility mode.

```
\let\PrologInit=\PL@INIT
```