The Ciao Prolog System

A Next Generation Logic Programming Environment
REFERENCE MANUAL
The Ciao System Documentation Series
Technical Report CLIP 3/97.1
Generated/Printed on: 5 October 2004
Version 1.11#222 (2004/5/24, 13:8:7 CEST)

F. Bueno
D. Cabeza
M. Carro
M. Hermenegildo
P. López
G. Puebla
clip@clip.dia.fi.upm.es
http://www.clip.dia.fi.upm.es/
The CLIP Group
School of Computer Science
Technical University of Madrid
Copyright © F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López, and G. Puebla
This document may be freely read, stored, reproduced, disseminated, translated or quoted by any means and on any medium provided the following conditions are met:

1. Every reader or user of this document acknowledges that is aware that no guarantee is given regarding its contents, on any account, and specifically concerning veracity, accuracy and fitness for any purpose.

2. No modification is made other than cosmetic, change of representation format, translation, correction of obvious syntactic errors, or as permitted by the clauses below.

3. Comments and other additions may be inserted, provided they clearly appear as such; translations or fragments must clearly refer to an original complete version, preferably one that is easily accessed whenever possible.

4. Translations, comments and other additions or modifications must be dated and their author(s) must be identifiable (possibly via an alias).

5. This licence is preserved and applies to the whole document with modifications and additions (except for brief quotes), independently of the representation format.

6. Any reference to the "official version", "original version" or "how to obtain original versions" of the document is preserved verbatim. Any copyright notice in the document is preserved verbatim. Also, the title and author(s) of the original document should be clearly mentioned as such.

7. In the case of translations, verbatim sentences mentioned in (6.) are preserved in the language of the original document accompanied by verbatim translations to the language of the translated document. All translations state clearly that the author is not responsible for the translated work. This license is included, at least in the language in which it is referenced in the original version.

8. Whatever the mode of storage, reproduction or dissemination, anyone able to access a digitized version of this document must be able to make a digitized copy in a format directly usable, and if possible editable, according to accepted, and publicly documented, public standards.

9. Redistributing this document to a third party requires simultaneous redistribution of this licence, without modification, and in particular without any further condition or restriction, expressed or implied, related or not to this redistribution. In particular, in case of inclusion in a database or collection, the owner or the manager of the database or the collection renounces any right related to this inclusion and concerning the possible uses of the document after extraction from the database or the collection, whether alone or in relation with other documents.

Any incompatibility of the above clauses with legal, contractual or judiciary decisions or constraints implies a corresponding limitation of reading, usage, or redistribution rights for this document, verbatim or modified.
# Table of Contents

## Summary .................................................. 1

## 1 Introduction ............................................. 3
  1.1 About this manual ........................................ 3
  1.2 About the Ciao Prolog development system ............... 3
  1.3 ISO-Prolog compliance versus extensibility .............. 4
  1.4 About the name of the System ............................ 4
  1.5 Referring to Ciao ....................................... 5
  1.6 Syntax terminology and notational conventions .......... 5
    1.6.1 Predicates and their components .................. 5
    1.6.2 Characters and character strings .................. 5
    1.6.3 Predicate specs .................................. 6
    1.6.4 Modes ............................................ 6
    1.6.5 Properties and types ............................... 6
    1.6.6 Declarations ..................................... 6
    1.6.7 Operators ........................................ 7
  1.7 A tour of the manual ................................... 7
    1.7.1 PART I - The program development environment ...... 7
    1.7.2 PART II - The Ciao basic language (engine) ......... 8
    1.7.3 PART III - ISO-Prolog library (iso) ................ 8
    1.7.4 PART IV - Classic Prolog library (classic) ......... 8
    1.7.5 PART V - Annotated Prolog library (assertions) ..... 8
    1.7.6 PART VI - Ciao Prolog library miscellanea .......... 8
    1.7.7 PART VII - Ciao Prolog extensions .................. 8
    1.7.8 PART VIII - Interfaces to other languages and systems ... 9
    1.7.9 PART IX - Abstract data types .................... 9
    1.7.10 PART X - Miscellaneous standalone utilities ....... 9
    1.7.11 PART XI - Contributed libraries .................. 9
    1.7.12 PART XII - Appendices ............................ 9
  1.8 Acknowledgments ....................................... 9
  1.9 Version/Change Log (ciao) ............................... 10

## 2 Getting started on Un*x-like machines .............. 17
  2.1 Testing your Ciao Un*x installation .................... 17
  2.2 Un*x user setup ....................................... 17
  2.3 Using Ciao from a Un*x command shell .................. 18
    2.3.1 Starting/exiting the top-level shell (Un*x) ....... 18
    2.3.2 Getting help (Un*x) ................................ 18
    2.3.3 Compiling and running programs (Un*x) ............. 18
    2.3.4 Generating executables (Un*x) ..................... 19
    2.3.5 Running Ciao scripts (Un*x) ....................... 19
    2.3.6 The Ciao initialization file (Un*x) ............... 20
    2.3.7 Printing manuals (Un*x) ........................... 20
  2.4 An introduction to the Ciao emacs environment (Un*x) ... 20
  2.5 Keeping up to date (Un*x) ............................. 21
3 Getting started on Windows machines ....... 23
  3.1 Testing your Ciao Win32 installation ........... 23
  3.2 Using Ciao from the Windows explorer and command shell .... 23
    3.2.1 Starting/exiting the top-level shell (Win32) .......... 23
    3.2.2 Getting help (Win32) .................................. 24
    3.2.3 Compiling and running programs (Win32) ............. 24
    3.2.4 Generating executables (Win32) ....................... 24
    3.2.5 Running Ciao scripts (Win32) ......................... 25
    3.2.6 The Ciao initialization file (Win32) .................. 25
    3.2.7 Printing manuals (Win32) .............................. 25
  3.3 An introduction to the Ciao emacs environment (Win32) .... 25
  3.4 Keeping up to date (Win32) ............................ 26

PART I - The program development environment ........................................ 27

4 The stand-alone command-line compiler ........ 29
  4.1 Introduction to building executables .............. 29
  4.2 Paths used by the compiler during compilation ........ 30
  4.3 Running executables from the command line .......... 30
  4.4 Types of executables generated ........................ 31
  4.5 Environment variables used by Ciao executables ...... 33
  4.6 Intermediate files in the compilation process ....... 33
  4.7 Usage (ciaoc) ............................................ 34

5 The interactive top-level shell ............... 37
  5.1 Shell invocation and startup ............................ 37
  5.2 Shell interaction ......................................... 37
  5.3 Entering recursive (conjunctive) shell levels ........ 38
  5.4 Usage and interface (ciaosh) ......................... 40
    5.5 Documentation on exports (ciaosh) ................. 40
      use_module/1 (pred) ...................................... 40
      use_module/2 (pred) ..................................... 40
      ensure_loaded/1 (pred) ................................. 40
      make_exec/2 (pred) ...................................... 41
      include/1 (pred) ........................................ 41
      use_package/1 (pred) .................................... 41
      consult/1 (pred) ....................................... 41
      compile/1 (pred) ........................................ 41
      ./2 (pred) ............................................... 42
      make_po/1 (pred) ....................................... 42
      unload/1 (pred) ........................................ 42
      set_debug_mode/1 (pred) ............................... 42
      set_nodBUG_mode/1 (pred) .............................. 42
      make_actmod/2 (pred) .................................. 42
      force_lazy/1 (pred) .................................... 43
      undo_force_lazy/1 (pred) ............................. 43
      dynamic_search_path/1 (pred) ...................... 43
      (multifile)/1 (pred) .................................. 43
    5.6 Documentation on internals (ciaosh) ............... 43
      sourcenames/1 (prop) ................................. 43
6 The interactive debugger

6.1 Marking modules and files for debugging in the top-level debugger

6.2 The debugging process

6.3 Marking modules and files for debugging with the embedded debugger

6.4 The procedure box control flow model

6.5 Format of debugging messages

6.6 Options available during debugging

6.7 Calling predicates that are not exported by a module

6.8 Acknowledgements

7 Predicates controlling the interactive debugger

7.1 Usage and interface (debugger)

7.2 Documentation on exports (debugger)

db_module/1 (pred)
nodebug_module/1 (pred)
db_module_source/1 (pred)
db/0 (pred)
nodebug/0 (pred)
trace/0 (pred)
notrace/0 (pred)
spy/1 (pred)
nospy/1 (pred)
nospyall/0 (pred)
breakpt/6 (pred)
nobreakpt/6 (pred)
nobreakall/0 (pred)
list_breakpt/0 (pred)
db冬奥会/0 (pred)
leash/1 (pred)
maxdepth/1 (pred)
call_in_module/2 (pred)
adjus.db_module_state/2 (pred)

7.3 Documentation on internals (debugger)

7.4 Known bugs and planned improvements (debugger)

8 The script interpreter

8.1 How it works

8.2 Command line arguments in scripts

9 Customizing library paths and path aliases

9.1 Usage and interface (libpaths)

9.2 Documentation on exports (libpaths)

get_alias_path/0 (pred)

9.3 Documentation on multilfiles (libpaths)

9.4 Known bugs and planned improvements (libpaths)

10 CiaoMode (library)

10.1 Usage and interface (CiaoMode)
PART II - The Ciao basic language (engine)       65

11 The module system .................................................. 67
   11.1 Usage and interface (modules) ................................ 67
   11.2 Documentation on internals (modules) ....................... 67
       module/3 (decl) ........................................ 67
       module/2 (decl) ........................................ 68
       export/1 (decl) ........................................ 68
       use_module/2 (decl) ..................................... 68
       use_module/1 (decl) ..................................... 69
       import/2 (decl) ......................................... 69
       reexport/2 (decl) ....................................... 69
       reexport/1 (decl) ....................................... 69
       (meta_predicate)/1 (decl) ................................ 69
       modulename/1 (regtype) .................................. 70
       metaspec/1 (regtype) .................................... 70

12 Directives for using code in other files ........... 71
   12.1 Usage and interface (loading_code) ......................... 71
   12.2 Documentation on internals (loading_code) ............... 71
       ensure_loaded/1 (decl) ................................... 71
       include/1 (decl) ........................................ 71
       use_package/1 (decl) .................................... 71

13 Control constructs/predicates ......................... 73
   13.1 Usage and interface (basiccontrol) ......................... 73
   13.2 Documentation on exports (basiccontrol) ................. 73
       ./2 (pred) ............................................... 73
       ;/2 (pred) ............................................... 73
       -> /2 (pred) ............................................. 74
       !/0 (pred) ............................................... 74
       (+)/1 (pred) ............................................. 74
       if/3 (pred) ............................................... 74
       true/0 (pred) ............................................ 75
       fail/0 (pred) ............................................ 75
       repeat/0 (pred) .......................................... 75
       false/0 (pred) ........................................... 75
       otherwise/0 (pred) ...................................... 75
       $metachoice/1 (pred) ..................................... 75
       $metacut/1 (pred) ....................................... 75
       interpret_goal/2 (pred) .................................. 76
       interpret_compiled_goal/2 (pred) ......................... 76
       undefined_goal/1 (pred) ................................ 76
       debug_goal/1 (pred) ..................................... 76
   13.3 Documentation on internals (basiccontrol) ............... 76
       1/2 (pred) ................................................ 76
14 Basic builtin directives
14.1 Usage and interface (builtin_directives) ............... 77
14.2 Documentation on internals (builtin_directives) ....... 77

15 Basic data types and properties .............. 79
15.1 Usage and interface (basic_props) .................. 79
15.2 Documentation on exports (basic_props) ........... 79

16 Extra-logical properties for typing ............. 87
16.1 Usage and interface (term_typing) ................ 87
16.2 Documentation on exports (term_typing) ......... 87
### 17 Basic term manipulation .......................... 91
17.1 Usage and interface *(term_basic)* .................. 91
17.2 Documentation on exports *(term_basic)* .......... 91
   == /2 (pred) ........................................... 91
   arg/3 (pred) .......................................... 91
   functor/3 (pred) ....................................... 91
   =.. /2 (pred) .......................................... 92
   copy_term/2 (pred) .................................... 92
   C/3 (pred) ............................................. 93

### 18 Comparing terms ................................. 95
18.1 Usage and interface *(term_compare)* ............. 95
18.2 Documentation on exports *(term_compare)* ...... 95
   == /2 (prop) .......................................... 95
   \== /2 (pred) ......................................... 95
   @< /2 (pred) .......................................... 96
   @= < /2 (pred) ......................................... 96
   @> /2 (pred) ........................................... 96
   @>= /2 (pred) ......................................... 96
   compare/3 (pred) ....................................... 97

### 19 Basic predicates handling names of constants
                                     ................................. 99
19.1 Usage and interface *(atomic_basic)* ............. 99
19.2 Documentation on exports *(atomic_basic)* ...... 99
   name/2 (pred) .......................................... 99
   atom_codes/2 (pred) ................................... 100
   number_codes/2 (pred) ................................ 100
   number_codes/3 (pred) ................................ 101
   atom_number/2 (pred) .................................. 102
   atom_length/2 (pred) .................................. 102
   atom_concat/3 (pred) ................................... 103
   sub_atom/4 (pred) ...................................... 104

### 20 Arithmetic ......................................... 105
20.1 Usage and interface *(arithmetic)* ................ 105
20.2 Documentation on exports *(arithmetic)* ......... 105
   is/2 (pred) ............................................ 105
   < /2 (pred) ............................................ 106
   <= /2 (pred) .......................................... 106
   > /2 (pred) ............................................ 106
   >= /2 (pred) .......................................... 107
   == /2 (pred) .......................................... 107
   =\= /2 (pred) .......................................... 107
   arithexpression/1 (regtype) .......................... 108
# Chapter 21 Basic file/stream handling

## 21.1 Usage and interface (streams_basic)

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>open/3 (pred)</td>
<td>111</td>
</tr>
<tr>
<td>open/4 (pred)</td>
<td>112</td>
</tr>
<tr>
<td>open_option_list/1 (regtype)</td>
<td>112</td>
</tr>
<tr>
<td>close/1 (pred)</td>
<td>113</td>
</tr>
<tr>
<td>set_input/1 (pred)</td>
<td>113</td>
</tr>
<tr>
<td>current_input/1 (pred)</td>
<td>113</td>
</tr>
<tr>
<td>set_output/1 (pred)</td>
<td>113</td>
</tr>
<tr>
<td>current_output/1 (pred)</td>
<td>114</td>
</tr>
<tr>
<td>character_count/2 (pred)</td>
<td>114</td>
</tr>
<tr>
<td>line_count/2 (pred)</td>
<td>114</td>
</tr>
<tr>
<td>line_position/2 (pred)</td>
<td>114</td>
</tr>
<tr>
<td>flush_output/1 (pred)</td>
<td>115</td>
</tr>
<tr>
<td>flush_output/0 (pred)</td>
<td>115</td>
</tr>
<tr>
<td>clearerr/1 (pred)</td>
<td>115</td>
</tr>
<tr>
<td>current_stream/3 (pred)</td>
<td>115</td>
</tr>
<tr>
<td>stream_code/2 (pred)</td>
<td>116</td>
</tr>
<tr>
<td>absolute_file_name/2 (pred)</td>
<td>116</td>
</tr>
<tr>
<td>absolute_file_name/7 (pred)</td>
<td>117</td>
</tr>
<tr>
<td>sourcename/1 (regtype)</td>
<td>117</td>
</tr>
<tr>
<td>stream/1 (regtype)</td>
<td>118</td>
</tr>
<tr>
<td>stream_alias/1 (regtype)</td>
<td>118</td>
</tr>
<tr>
<td>io_mode/1 (regtype)</td>
<td>118</td>
</tr>
</tbody>
</table>

## 21.2 Documentation on exports (streams_basic)

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_search_path/2 (pred)</td>
<td>119</td>
</tr>
<tr>
<td>library_directory/1 (pred)</td>
<td>119</td>
</tr>
</tbody>
</table>

## 21.3 Documentation on multifiles (streams_basic)

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
</table>
| io_basic
| file_search_path/2 (pred) | 119 |
| library_directory/1 (pred) | 119 |

# Chapter 22 Basic input/output

## 22.1 Usage and interface (io_basic)

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_code/2 (pred)</td>
<td>121</td>
</tr>
<tr>
<td>get_code/1 (pred)</td>
<td>121</td>
</tr>
<tr>
<td>get1_code/2 (pred)</td>
<td>122</td>
</tr>
<tr>
<td>get1_code/1 (pred)</td>
<td>122</td>
</tr>
<tr>
<td>peek_code/2 (pred)</td>
<td>122</td>
</tr>
<tr>
<td>peek_code/1 (pred)</td>
<td>123</td>
</tr>
<tr>
<td>skip_code/2 (pred)</td>
<td>123</td>
</tr>
<tr>
<td>skip_code/1 (pred)</td>
<td>123</td>
</tr>
<tr>
<td>skip_line/1 (pred)</td>
<td>123</td>
</tr>
<tr>
<td>skip_line/0 (pred)</td>
<td>123</td>
</tr>
<tr>
<td>put_code/2 (pred)</td>
<td>123</td>
</tr>
<tr>
<td>put_code/1 (pred)</td>
<td>124</td>
</tr>
<tr>
<td>nl/1 (pred)</td>
<td>124</td>
</tr>
<tr>
<td>nl/0 (pred)</td>
<td>124</td>
</tr>
<tr>
<td>tab/2 (pred)</td>
<td>124</td>
</tr>
<tr>
<td>tab/1 (pred)</td>
<td>125</td>
</tr>
<tr>
<td>code_class/2 (pred)</td>
<td>125</td>
</tr>
<tr>
<td>getct/2 (pred)</td>
<td>125</td>
</tr>
<tr>
<td>getct1/2 (pred)</td>
<td>125</td>
</tr>
<tr>
<td>display/2 (pred)</td>
<td>126</td>
</tr>
<tr>
<td>display/1 (pred)</td>
<td>126</td>
</tr>
<tr>
<td>displayq/2 (pred)</td>
<td>127</td>
</tr>
<tr>
<td>displayq/1 (pred)</td>
<td>127</td>
</tr>
</tbody>
</table>
23 Exception handling .......................... 129
   23.1 Usage and interface (exceptions) ....................... 129
   23.2 Documentation on exports (exceptions) ................. 129
      catch/3 (pred) ........................................ 129
      intercept/3 (pred) .................................. 129
      throw/1 (pred) ....................................... 130
      halt/0 (pred) .......................................... 130
      halt/1 (pred) .......................................... 130
      abort/0 (pred) ........................................ 130

24 Changing system behaviour and various flags .................... 131
   24.1 Usage and interface (prolog_flags) ..................... 132
   24.2 Documentation on exports (prolog_flags) ................ 132
      set_prolog_flag/2 (pred) ............................ 132
      current_prolog_flag/2 (pred) ....................... 132
      prolog_flag/3 (pred) ................................ 132
      push_prolog_flag/2 (pred) ......................... 133
      pop_prolog_flag/1 (pred) ........................... 133
      prompt/2 (pred) ...................................... 133
      gc/0 (pred) ........................................... 134
      nogc/0 (pred) ........................................ 134
      fileerrors/0 (pred) .................................. 134
      nofileerrors/0 (pred) ................................ 134
   24.3 Documentation on multilfiles (prolog_flags) .......... 134
      define_flag/3 (pred) .................................. 134

25 Fast/concurrent update of facts ............................. 137
   25.1 Usage and interface (data_facts) ....................... 137
   25.2 Documentation on exports (data_facts) ................. 137
      asserta_fact/1 (pred) .............................. 137
      asserta_fact/2 (pred) ................................ 137
      assertz_fact/1 (pred) .............................. 138
      assertz_fact/2 (pred) ................................ 138
      current_fact/1 (pred) .............................. 138
      current_fact/2 (pred) ................................ 139
      retract_fact/1 (pred) ................................ 139
      retractall_fact/1 (pred) ............................ 139
      current_fact_nb/1 (pred) ......................... 140
      retract_fact_nb/1 (pred) ........................... 140
      close_predicate/1 (pred) ......................... 140
      open_predicate/1 (pred) ........................... 140
      set_fact/1 (pred) ................................... 140
      erase/1 (pred) ....................................... 141
   25.3 Documentation on internals (data_facts) ............... 141
      (data)/1 (decl) ...................................... 141
      (concurrent)/1 (decl) ................................ 141
      reference/1 (regtype) ................................ 141
26 Exploring the syntax .......................... 143
26.1 Usage and interface (syntax_extensions) .......... 143
26.2 Documentation on internals (syntax_extensions) .... 143
   op/3 (decl) ........................................ 143
   new_declaration/1 (DECL) ........................................ 143
   new_declaration/2 (DECL) ........................................ 143
   load_compilation_module/1 (DECL) ................. 144
   add_sentence_trans/1 (DECL) .................................... 144
   add_term_trans/1 (DECL) .................................... 144
   add_goal_trans/1 (DECL) .................................. 145
   add_clause_trans/1 (DECL) .................................. 145
   translation_predname/1 (REGTYPE) ................. 145

27 Message printing primitives ...................... 147
27.1 Usage and interface (io_aux) .................... 147
27.2 Documentation on exports (io_aux) ............... 147
   message/2 (Pred) ....................................... 147
   message_lns/4 (Pred) ................................... 147
   error/1 (Pred) .......................................... 148
   warning/1 (Pred) ....................................... 148
   note/1 (Pred) ........................................... 148
   message/1 (Pred) ....................................... 148
   debug/1 (Pred) .......................................... 148
   inform_user/1 (Pred) .................................. 149
   display_string/1 (Pred) .................................. 149
   display_lis/1 (Pred) ................................... 149
   display_term/1 (Pred) ................................ 149
27.3 Known bugs and planned improvements (io_aux) .... 149

28 Attributed variables .............................. 151
28.1 Usage and interface (attributes) ................. 151
28.2 Documentation on exports (attributes) .......... 151
   attach_attribute/2 (Pred) ............................. 151
   get_attribute/2 (Pred) ................................. 151
   update_attribute/2 (Pred) ............................ 151
   detach_attribute/1 (Pred) ............................ 152
28.3 Documentation on multi-fies (attributes) ........ 152
   verify_attribute/2 (Pred) ............................. 152
   combine_attributes/2 (Pred) ......................... 152
28.4 Other information (attributes) ................... 152

29 Gathering some basic internal info .............. 155
29.1 Usage and interface (system_info) .............. 155
29.2 Documentation on exports (system_info) ........ 155
   get_arch/1 (Pred) ...................................... 155
   get_os/1 (Pred) ......................................... 155
   this_module/1 (Pred) .................................. 156
   current_module/1 (Pred) ............................... 156
   ciaoilibdir/1 (Pred) .................................. 156
29.3 Documentation on internals (system_info) ....... 157
   internal_module_id/1 (Prop) ........................... 157
### 30 Other predicates and features defined by default

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.1 Usage and interface (default_predicates)</td>
<td></td>
<td>159</td>
</tr>
<tr>
<td>30.2 Documentation on exports (default_predicates)</td>
<td></td>
<td>159</td>
</tr>
</tbody>
</table>

- `op/3 (udreexp)`
- `current_op/3 (udreexp)`
- `append/3 (udreexp)`
- `delete/3 (udreexp)`
- `select/3 (udreexp)`
- `nth/3 (udreexp)`
- `last/2 (udreexp)`
- `reverse/2 (udreexp)`
- `length/2 (udreexp)`
- `ensure/1 (udreexp)`
- `use/1 (udreexp)`
- `default_predicates` (exported)
- `default_predicates` (module)
- `assert/2 (udreexp)`
- `retract/1 (udreexp)`
- `assertz/1 (udreexp)`
- `asserta/2 (udreexp)`
- `assert/1 (udreexp)`
- `assert/1 (udreexp)`
- `assertz/2 (udreexp)`
- `asserta/2 (udreexp)`
- `asserta/1 (udreexp)`
- `second_prompt/2 (udreexp)`
- `read_top_level/3 (udreexp)`
- `read_term/3 (udreexp)`
- `read_term/2 (udreexp)`
- `read/2 (udreexp)`
- `read/1 (udreexp)`
- `printable_char/1 (udreexp)`
- `prettyvars/1 (udreexp)`
- `numbervars/3 (udreexp)`
- `portray_clause/1 (udreexp)`
- `portray_clause/2 (udreexp)`
- `write_list1/1 (udreexp)`
- `print/1 (udreexp)`
- `print/2 (udreexp)`
- `write_canonical/1 (udreexp)`
- `write_canonical/2 (udreexp)`
- `writeeq/1 (udreexp)`
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>shell/0 (udreexp)</td>
<td>168</td>
</tr>
<tr>
<td>cd/1 (udreexp)</td>
<td>168</td>
</tr>
<tr>
<td>working_directory/2 (udreexp)</td>
<td>168</td>
</tr>
<tr>
<td>make_dirpath/1 (udreexp)</td>
<td>168</td>
</tr>
<tr>
<td>make_dirpath/2 (udreexp)</td>
<td>168</td>
</tr>
<tr>
<td>make_directory/1 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>make_directory/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>umask/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>current_executable/1 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>current_host/1 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>get_pid/1 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>extract_paths/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>copy_file/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>del_env/1 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>set_env/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>current_env/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>getenvstr/2 (udreexp)</td>
<td>169</td>
</tr>
<tr>
<td>datetime_struct/1 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>datetime/9 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>datetime/1 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>time/1 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>pause/1 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>new_atom/1 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>garbage_collect/0 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>current_atom/1 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>predicate_property/2 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>statistics/2 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>statistics/0 (udreexp)</td>
<td>170</td>
</tr>
<tr>
<td>close_file/1 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>told/0 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>telling/1 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>tell/1 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>seen/0 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>seeing/1 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>see/1 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>current_key/2 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>recorded/3 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>recordz/3 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>recorda/3 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>ttydisplay_string/1 (udreexp)</td>
<td>171</td>
</tr>
<tr>
<td>ttydisplayq/1 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttydisplay/1 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttyflush/0 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttytab/1 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttyskip/1 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttyput/1 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttynl/0 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttyget1/1 (udreexp)</td>
<td>172</td>
</tr>
<tr>
<td>ttyget/1 (udreexp)</td>
<td>172</td>
</tr>
</tbody>
</table>

PART III - ISO-Prolog library (iso) ............. 173
31 ISO-Prolog package .................................... 175
  31.1 Usage and interface (iso) ......................... 175

32 All solutions predicates ......................... 177
  32.1 Usage and interface (aggregates) ............... 177
  32.2 Documentation on exports (aggregates) .......... 177
  setof/3 (pred) ................................... 177
  bagof/3 (pred) .................................. 178
  findall/3 (pred) .................................. 179
  findall/4 (pred) .................................. 179
  findall/5 (pred) .................................. 179
  findall/6 (pred) .................................. 179
  ^/2 (pred) ........................................ 180

33 Dynamic predicates ............................... 181
  33.1 Usage and interface (dynamic) .................... 181
  33.2 Documentation on exports (dynamic) .......... 181
  asserta/1 (pred) .................................. 181
  asserta/2 (pred) .................................. 182
  assertz/1 (pred) .................................. 182
  assertz/2 (pred) .................................. 182
  assert/1 (pred) ................................... 182
  assert/2 (pred) ................................... 182
  retract/1 (pred) .................................. 183
  retractall/1 (pred) ................................ 183
  abolish/1 (pred) ................................... 183
  clause/2 (pred) ................................... 184
  clause/3 (pred) ................................... 184
  current_predicate/1 (pred) ......................... 184
  current_predicate/2 (pred) ......................... 184
  (dynamic)/1 (pred) ................................ 185
  (data)/1 (pred) ................................... 185
  wellformed_body/3 (pred) ......................... 185
  33.3 Documentation on multi-files (dynamic) ....... 185
    do_on_abolish/1 (pred) .......................... 185

34 Term input .......................................... 187
  34.1 Usage and interface (read) ...................... 187
  34.2 Documentation on exports (read) ............... 187
    read/1 (pred) ................................... 187
    read/2 (pred) ................................... 187
    read_term/2 (pred) ............................... 188
    read_term/3 (pred) ............................... 188
    read_top_level/3 (pred) ......................... 188
    second_prompt/2 (pred) ......................... 188
  34.3 Documentation on multi-files (read) .......... 188
    define_flag/3 (pred) ............................. 188
  34.4 Documentation on internals (read) ............. 189
    read_option/1 (regtype) ......................... 189
  34.5 Known bugs and planned improvements (read) ... 189
35 Term output .......................... 191
  35.1 Usage and interface (write) .................... 191
  35.2 Documentation on exports (write) .......... 191
      write_term/3 (pred) .......................... 191
      write_term/2 (pred) ......................... 192
      write_option/1 (prop) ...................... 192
      write/2 (pred) ............................... 192
      write/1 (pred) ............................... 193
      writeq/2 (pred) .............................. 193
      writeq/1 (pred) .............................. 193
      write/2 (pred) ............................... 192
      write/1 (pred) ............................... 193
      writeq/2 (pred) .............................. 193
      writeq/1 (pred) .............................. 193
      write/2 (pred) ............................... 192
      write/1 (pred) ............................... 193
      writeq/2 (pred) .............................. 193
      writeq/1 (pred) .............................. 193
      write/2 (pred) ............................... 192
      write/1 (pred) ............................... 193
      writeq/2 (pred) .............................. 193
      writeq/1 (pred) .............................. 193
  35.3 Documentation on mult-files (write) .......... 196
      define_flag/3 (pred) ....................... 196
      portray_attribute/2 (pred) .................. 196
      portray/1 (pred) ............................ 196
  36 Defining operators .......................... 197
  36.1 Usage and interface (operators) .......... 197
  36.2 Documentation on exports (operators) .... 197
      op/3 (pred) .................................. 197
      current_op/3 (pred) ......................... 198
      current_prefixop/3 (pred) ................. 198
      current_infixop/4 (pred) .................. 198
      current_postfixop/3 (pred) ............... 198
  37 The Iso Byte Char module ................. 199
  37.1 Usage and interface (iso_byte_char) .... 199
  37.2 Documentation on exports (iso_byte_char) 199
      char_code/2 (pred) .......................... 199
      atom_chars/2 (pred) ........................ 199
      number_chars/2 (pred) ..................... 200
      get_byte/1 (pred) ........................... 200
      get_byte/2 (pred) ........................... 201
      peek_byte/1 (pred) .......................... 201
      peek_byte/2 (pred) .......................... 201
      put_byte/1 (pred) ........................... 201
      put_byte/2 (pred) ........................... 202
      get_char/1 (pred) ........................... 202
      get_char/2 (pred) ........................... 202
      peek_char/1 (pred) .......................... 202
      peek_char/2 (pred) .......................... 203
      put_char/1 (pred) ........................... 203
      put_char/2 (pred) ........................... 203
38 Miscellaneous ISO Prolog predicates ........ 205
  38.1 Usage and interface (iso_misc) ............ 205
  38.2 Documentation on exports (iso_misc) ....... 205
    \= /2 (pred) .................................. 205
    once/1 (pred) .................................. 205
    compound/1 (pred) ............................. 205
    sub_atom/5 (pred) ............................. 205
    unify_with_occurs_check/2 (pred) ............ 206

39 Incomplete ISO Prolog predicates .......... 207
  39.1 Usage and interface (iso_incomplete) ....... 207
  39.2 Documentation on exports (iso_incomplete)... 207
    close/2 (pred) .............................. 207
    stream_property/2 (pred) ..................... 207

PART IV - Classic Prolog library (classic) ..... 209

40 Definite clause grammars ................. 211
  40.1 Usage and interface (dcg) ................. 213

41 Definite clause grammars (expansion) ...... 215
  41.1 Usage and interface (dcg_expansion) ...... 215
  41.2 Documentation on exports (dcg_expansion) . 215
    phrase/2 (pred) ............................. 215
    phrase/3 (pred) ............................. 215
    dcg_translation/2 (pred) .................... 215

42 Formatted output ............................ 217
  42.1 Usage and interface (format) .............. 217
  42.2 Documentation on exports (format) ......... 218
    format/2 (pred) ............................ 218
    format/3 (pred) ............................ 218
    format_control/1 (regtype) ................ 218

43 List processing ............................. 223
  43.1 Usage and interface (lists) ............... 223
  43.2 Documentation on exports (lists) ......... 223
    nonsingle/1 (pred) ........................ 223
    append/3 (pred) ............................ 223
    reverse/2 (pred) ........................... 223
    reverse/3 (pred) ........................... 224
    delete/3 (pred) ............................ 224
    delete_non_ground/3 (pred) ............... 224
    select/3 (pred) ............................ 224
    length/2 (pred) ............................ 224
    nth/3 (pred) ............................... 225
    add_after/4 (pred) ........................ 225
    add_before/4 (pred) ....................... 226
    list1/2 (prop) ............................. 226
    dlist/3 (pred) ............................. 226
    list_concat/2 (pred) ...................... 226
    list_insert/2 (pred) ...................... 226
    insert_last/3 (pred) ...................... 226
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains_ro/2 (pred)</td>
<td>226</td>
</tr>
<tr>
<td>contains1/2 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>nocontainsx/2 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>last/2 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>list_lookup/3 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>list_lookup/4 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>intset_insert/3 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>intset_delete/3 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>intset_in/2 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>intset_sequence/3 (pred)</td>
<td>227</td>
</tr>
<tr>
<td>intersection/3 (pred)</td>
<td>228</td>
</tr>
<tr>
<td>union/3 (pred)</td>
<td>228</td>
</tr>
<tr>
<td>difference/3 (pred)</td>
<td>228</td>
</tr>
<tr>
<td>sublist/2 (prop)</td>
<td>228</td>
</tr>
<tr>
<td>subordlist/2 (prop)</td>
<td>228</td>
</tr>
<tr>
<td>equal_lists/2 (pred)</td>
<td>229</td>
</tr>
<tr>
<td>list_to_list_of_lists/2 (pred)</td>
<td>229</td>
</tr>
<tr>
<td>powerset/2 (pred)</td>
<td>229</td>
</tr>
<tr>
<td>cross_product/2 (pred)</td>
<td>229</td>
</tr>
</tbody>
</table>

### 44 Sorting lists ........................................ 231

44.1 Usage and interface (sort) .......................... 231
44.2 Documentation on exports (sort) .................... 231
  sort/2 (pred) ............................................. 231
  keysort/2 (pred) ......................................... 231
  keylist/1 (regtype) ........................................ 232
44.3 Documentation on internals (sort) .................. 232
  keypair/1 (regtype) ........................................ 232

### 45 compiler (library) ................................. 233

45.1 Usage and interface (compiler) ...................... 233
45.2 Documentation on exports (compiler) ............... 233
  make_po/1 (pred) .......................................... 233
  ensure_loaded/1 (pred) ..................................... 233
  ensure_loaded/2 (pred) ..................................... 233
  use_module/1 (pred) ....................................... 233
  use_module/2 (pred) ....................................... 233
  use_module/3 (pred) ....................................... 233
  unload/1 (pred) ............................................ 234
  set_debug_mode/1 (pred) ................................... 234
  set_nodbug_mode/1 (pred) ................................. 234
  set_debug_module/1 (pred) ................................ 234
  set_nodbug_module/1 (pred) ............................... 234
  set_debug_module_source/1 (pred) ....................... 234
  mode_of_module/2 (pred) .................................. 234
  module_of/2 (pred) ....................................... 234

### 46 Enumeration of integers inside a range ........ 235

46.1 Usage and interface (between) ...................... 235
46.2 Documentation on exports (between) ............... 235
  between/3 (pred) ........................................... 235
47 Operating system utilities .................. 237

47.1 Usage and interface (system) .................. 237
47.2 Documentation on exports (system) .............. 237
   pause/1 (pred) ........................................ 237
   time/1 (pred) ......................................... 237
   datime/1 (pred) ....................................... 238
   datime/9 (pred) ....................................... 238
   datime_struct/1 (regtype) ........................... 239
   getenvstr/2 (pred) ................................... 239
   setenvstr/2 (pred) ................................... 240
   current_env/2 (pred) ................................ 240
   set_env/2 (pred) ...................................... 240
   del_env/1 (pred) ..................................... 240
   copy_file/2 (pred) ................................... 240
   extract_paths/2 (pred) ................................ 241
   get_pid/1 (pred) ...................................... 241
   current_host/1 (pred) ................................. 241
   current_executable/1 (pred) .......................... 242
   umask/2 (pred) ........................................ 242
   make_directory/2 (pred) ............................... 242
   make_directory/1 (pred) ............................... 243
   make_dirpath/2 (pred) ................................ 243
   make_dirpath/1 (pred) ................................ 243
   working_directory/2 (pred) ............................ 243
   cd/1 (pred) ............................................ 244
   shell/0 (pred) ......................................... 244
   shell/1 (pred) ......................................... 244
   shell/2 (pred) ......................................... 244
   system/1 (pred) ....................................... 245
   system/2 (pred) ....................................... 245
   popen/3 (pred) ........................................ 245
   popen_mode/1 (regtype) ............................... 245
   exec/4 (pred) .......................................... 245
   exec/3 (pred) .......................................... 246
   directory_files/2 (pred) .............................. 246
   mktemp/2 (pred) ...................................... 246
   file_exists/1 (pred) .................................. 247
   file_exists/2 (pred) .................................. 247
   file_property/2 (pred) ............................... 247
   file_properties/6 (pred) ............................. 248
   modif_time/2 (pred) .................................. 248
   modif_time/0/2 (pred) ................................ 248
   fmode/2 (pred) ........................................ 249
   chmod/2 (pred) ....................................... 249
   chmod/3 (pred) ....................................... 249
   delete_file/1 (pred) ................................ 250
   delete_directory/1 (pred) ............................ 250
   rename_file/2 (pred) ................................ 250
   cyg2win/3 (pred) ..................................... 250

47.3 Documentation on multifiles (system) .......... 251
   define_flag/3 (pred) ................................ 251

47.4 Known bugs and planned improvements (system) . 251
48 Prolog system internal predicates .............. 253
  48.1 Usage and interface (prolog_sys) .............. 253
  48.2 Documentation on exports (prolog_sys) ......... 253
     statistics/0 (pred) .......................... 253
     statistics/2 (pred) .......................... 253
     predicate_property/2 (pred) .................. 255
     current_atom/1 (pred) ......................... 255
     garbage_collect/0 (pred) ...................... 255
     new_atom/1 (pred) ............................ 255
  48.3 Documentation on internals (prolog_sys) ....... 256
     time_option/1 (regtype) ...................... 256
     memory_option/1 (regtype) ................... 256
     garbage_collection_option/1 (regtype) ....... 256
     symbol_option/1 (regtype) ................... 256
     time_result/1 (regtype) ...................... 256
     click_result/1 (regtype) ..................... 256
     clockfreq_result/1 (regtype) ................. 257
     memory_result/1 (regtype) ................... 257
     gc_result/1 (regtype) ....................... 257
     symbol_result/1 (regtype) ................... 257
  48.4 Known bugs and planned improvements (prolog_sys) ...... 257

49 DEC-10 Prolog file IO .......................... 259
  49.1 Usage and interface (dec10_io) ................. 259
  49.2 Documentation on exports (dec10_io) ............ 259
     see/1 (pred) ................................. 259
     seeing/1 (pred) .............................. 259
     seen/0 (pred) ............................... 259
     tell/1 (pred) .............................. 259
     telling/1 (pred) ............................ 259
     told/0 (pred) ............................... 259
     close_file/1 (pred) ......................... 259

50 Quintus-like internal database .................... 261
  50.1 Usage and interface (old_database) ............. 261
  50.2 Documentation on exports (old_database) ....... 261
     recorda/3 (pred) ........................... 261
     recordz/3 (pred) ........................... 261
     recorded/3 (pred) .......................... 262
     current_key/2 (pred) ....................... 262

51 ttyout (library) .................................. 263
  51.1 Usage and interface (ttyout) .................... 263
  51.2 Documentation on exports (ttyout) ............... 263
     ttyget/1 (pred) .............................. 263
     ttyget1/1 (pred) ............................. 263
     tynl/0 (pred) ............................... 263
     ttyput/1 (pred) .............................. 263
     ttykip/1 (pred) .............................. 263
     ttytab/1 (pred) .............................. 264
     ttyflush/0 (pred) ........................... 264
     ttydisplay/1 (pred) ......................... 264
     ttydisplayq/1 (pred) ....................... 264
     ttyskip/0 (pred) ............................. 264
     ttydisplay_string/1 (pred) ................. 264
PART V - Annotated Prolog library (assertions) ........................................ 267

53 The Ciao assertion package .................... 269

54 Types and properties related to assertions .. 277
55 Declaring regular types ......................... 283
  55.1 Defining properties .......................... 283
  55.2 Usage and interface (regtypes) ............. 286
  55.3 Documentation on new declarations (regtypes) 286
    regtype/1 (decl) .................................. 286
    regtype/2 (decl) .................................. 287

56 Properties which are native to analyzers .... 289
  56.1 Usage and interface (native_props) .......... 289
  56.2 Documentation on exports (native_props) .. 289
    covered/2 (prop) .................................. 289
    linear/1 (prop) .................................. 289
    mshare/1 (prop) .................................. 290
    nonground/1 (prop) ................................ 290
    fails/1 (prop) .................................. 290
    not_fails/1 (prop) ................................ 290
    possibly_fails/1 (prop) ......................... 290
    covered/1 (prop) .................................. 291
    not_covered/1 (prop) ............................. 291
    is_det/1 (prop) .................................. 291
    non_det/1 (prop) .................................. 291
    possibly_nondet/1 (prop) ....................... 291
    mut_exclusive/1 (prop) ......................... 291
    not_mut_exclusive/1 (prop) ....................... 291
    size_lb/2 (prop) .................................. 292
    size_ub/2 (prop) .................................. 292
    steps_lb/2 (prop) .................................. 292
    steps_ub/2 (prop) .................................. 292
    steps/2 (prop) .................................. 292
    finite_solutions/1 (prop) ....................... 293
    terminates/1 (prop) ................................ 293
    indep/1 (prop) .................................. 293
    indep/2 (prop) .................................. 293
    instance/2 (udreexp) ............................ 293

57 ISO-Prolog modes ................................. 295
  57.1 Usage and interface (isomodes) .............. 295
  57.2 Documentation on new modes (isomodes) .... 295
    (+)/1 (modedef) .................................. 295
    (@)/1 (modedef) .................................. 295
    (-)/1 (modedef) .................................. 295
    (?)/1 (modedef) .................................. 295
    */1 (modedef) .................................. 295
    (+)/2 (modedef) .................................. 296
    (@)/2 (modedef) .................................. 296
    (-)/2 (modedef) .................................. 296
    (?)/2 (modedef) .................................. 296
    */2 (modedef) .................................. 296
58 Classical Prolog modes ................. 297
  58.1 Usage and interface (basicmodes) ................. 297
  58.2 Documentation on new modes (basicmodes) ................. 297
  (+)/1 (modedef) ........................................ 297
  (-)/1 (modedef) ........................................ 297
  (?)/1 (modedef) ........................................ 297
  (Θ)/1 (modedef) ........................................ 297
  in/1 (modedef) ........................................ 298
  out/1 (modedef) ........................................ 298
  go/1 (modedef) ........................................ 298
  (+)/2 (modedef) ........................................ 298
  (-)/2 (modedef) ........................................ 298
  (?)/2 (modedef) ........................................ 298
  (Θ)/2 (modedef) ........................................ 299
  in/2 (modedef) ........................................ 299
  out/2 (modedef) ........................................ 299
  go/2 (modedef) ........................................ 299

PART VI - Ciao Prolog library miscellanea ......... 301

59 Structured stream handling ................. 303
  59.1 Usage and interface (streams) ................. 303
  59.2 Documentation on exports (streams) ................. 303
    open_null_stream/1 (pred) .............................. 303
    open_input/2 (pred) .............................. 303
    close_input/1 (pred) .............................. 303
    open_output/2 (pred) .............................. 303
    close_output/1 (pred) .............................. 303

60 Dictionaries ................................. 305
  60.1 Usage and interface (dict) ................. 305
  60.2 Documentation on exports (dict) ................. 305
    dictionary/1 (prop) .............................. 305
    dictionary/5 (pred) .............................. 305
    dic_node/2 (pred) .............................. 305
    dic_lookup/3 (pred) .............................. 305
    dic_lookup/4 (pred) .............................. 306
    dic_get/3 (pred) .............................. 306
    dic_replace/4 (pred) .............................. 306

61 String processing ................................. 307
  61.1 Usage and interface (strings) ................. 307
  61.2 Documentation on exports (strings) ................. 307
    get_line/2 (pred) .............................. 307
    get_line/1 (pred) .............................. 307
    write_string/2 (pred) .............................. 307
    write_string/1 (pred) .............................. 308
    whitespace/2 (pred) .............................. 308
    whitespace0/2 (pred) .............................. 308
    string/3 (pred) .............................. 309
  61.3 Documentation on internals (strings) ................. 309
    line/1 (prop) .............................. 309
62 Printing status and error messages ........ 311

62.1 Usage and interface (messages) .................. 311
62.2 Documentation on exports (messages) ............ 311
  error_message/1 (pred) ............................. 311
  error_message/2 (pred) ............................. 311
  error_message/3 (pred) ............................. 312
  warning_message/1 (pred) .......................... 312
  warning_message/2 (pred) .......................... 312
  warning_message/3 (pred) .......................... 312
  note_message/1 (pred) ............................... 312
  note_message/2 (pred) ............................... 313
  note_message/3 (pred) ............................... 313
  simple_message/1 (pred) ............................ 313
  simple_message/2 (pred) ............................ 313
  optional_message/2 (pred) .......................... 314
  optional_message/3 (pred) .......................... 314
  debug_message/1 (pred) .............................. 314
  debug_message/2 (pred) .............................. 314
  debug_goal/2 (pred) ................................. 314
  debug_goal/3 (pred) ................................. 315
  debug_goal/2 (pred) ................................. 315
  debug_goal/3 (pred) ................................. 315
62.3 Documentation on multifiles (messages) .......... 315
  issue_debug_messages/1 (pred) ...................... 315
62.4 Documentation on internals (messages) .......... 315
  location/1 (regtype) ................................. 315
62.5 Known bugs and planned improvements (messages) .. 315

63 Accessing and redirecting the stream aliases .... 317

63.1 Usage and interface (io_alias_redirection) ...... 317
63.2 Documentation on exports (io_alias_redirection) .. 317
  set_stream/3 (pred) .................................. 317
  get_stream/2 (pred) .................................. 317

64 Atom to term conversion .......................... 319

64.1 Usage and interface (atom2term) ................... 319
64.2 Documentation on exports (atom2term) ............. 319
  atom2term/2 (pred) .................................. 319
  string2term/2 (pred) ................................ 319
  parse_term/3 (pred) .................................. 319
64.3 Known bugs and planned improvements (atom2term) .. 319

65 ctrlcclean (library) .............................. 321

65.1 Usage and interface (ctrlcclean) .................... 321
65.2 Documentation on exports (ctrlcclean) .......... 321
  ctrlc_clean/1 (pred) .................................. 321
  delete_on_ctrlc/2 (pred) ............................ 321
  ctrlcclean/0 (pred) .................................. 321

66 errhandle (library) ............................... 323

66.1 Usage and interface (errhandle) ................... 323
66.2 Documentation on exports (errhandle) ............ 323
  error_protect/1 (pred) .................................. 323
  handle_error/2 (pred) ................................ 323
67 Fast reading and writing of terms .......... 325
   67.1 Usage and interface (fastrw) ................. 325
   67.2 Documentation on exports (fastrw) .......... 325
          fast_read/1 (pred) 325
          fast_write/1 (pred) 325
          fast_read/2 (pred) 325
          fast_write/2 (pred) 325
          fast_write_to_string/3 (pred) 325
   67.3 Known bugs and planned improvements (fastrw) ...... 326

68 File name manipulation .................... 327
   68.1 Usage and interface (filenames) ........... 327
   68.2 Documentation on exports (filenames) .... 327
          no_path_file_name/2 (pred) 327
          file_name_extension/3 (pred) 328
          basename/2 (pred) 328
          extension/2 (pred) 328

69 Symbolic filenames ....................... 329
   69.1 Usage and interface (symfnames) .......... 329
   69.2 Documentation on exports (symfnames) ..... 329
          open/3 (pred) 329
          alias_file/1 (pred) 330
          file_alias/2 (pred) 330
   69.3 Documentation on multfiles (symfnames) .... 330
   69.4 Other information (symfnames) .......... 330

70 File I/O utilities ....................... 331
   70.1 Usage and interface (file_utils) .......... 331
   70.2 Documentation on exports (file_utils) ... 331
          file_terms/2 (pred) 331
          copy_stdout/1 (pred) 332
          file_to_string/2 (pred) 332
          stream_to_string/2 (pred) 332

71 File locks ......................... 333
   71.1 Usage and interface (file_locks) .......... 333
   71.2 Documentation on exports (file_locks) .... 333
          lock_file/3 (pred) 333
          unlock_file/2 (pred) 333
   71.3 Known bugs and planned improvements (file_locks) ...... 333

72 Term manipulation utilities ............. 335
   72.1 Usage and interface (terms) ............... 335
   72.2 Documentation on exports (terms) ....... 335
          copy_args/3 (pred) 335
          arg/2 (pred) 335
          atom_concat/2 (pred) 335
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>Term checking utilities</td>
<td>337</td>
</tr>
<tr>
<td>73.1</td>
<td>Usage and interface (terms_check)</td>
<td>337</td>
</tr>
<tr>
<td>73.2</td>
<td>Documentation on exports (terms_check)</td>
<td>337</td>
</tr>
<tr>
<td>73.3</td>
<td>Other information (terms_check)</td>
<td>338</td>
</tr>
<tr>
<td>74</td>
<td>Sets of variables in terms</td>
<td>339</td>
</tr>
<tr>
<td>74.1</td>
<td>Usage and interface (terms_vars)</td>
<td>339</td>
</tr>
<tr>
<td>74.2</td>
<td>Documentation on exports (terms_vars)</td>
<td>339</td>
</tr>
<tr>
<td>75</td>
<td>A simple pretty-printer for Ciao programs</td>
<td>341</td>
</tr>
<tr>
<td>75.1</td>
<td>Usage and interface (pretty_print)</td>
<td>341</td>
</tr>
<tr>
<td>75.2</td>
<td>Documentation on exports (pretty_print)</td>
<td>341</td>
</tr>
<tr>
<td>76</td>
<td>Pretty-printing assertions</td>
<td>345</td>
</tr>
<tr>
<td>76.1</td>
<td>Usage and interface (assert_write)</td>
<td>345</td>
</tr>
<tr>
<td>76.2</td>
<td>Documentation on exports (assert_write)</td>
<td>345</td>
</tr>
<tr>
<td>77</td>
<td>The Ciao library browser</td>
<td>347</td>
</tr>
<tr>
<td>77.1</td>
<td>Usage and interface (librowser)</td>
<td>347</td>
</tr>
<tr>
<td>77.2</td>
<td>Documentation on exports (librowser)</td>
<td>348</td>
</tr>
<tr>
<td>77.3</td>
<td>Documentation on internals (librowser)</td>
<td>350</td>
</tr>
</tbody>
</table>
78 Code translation utilities .......................... 351
  78.1 Usage and interface (expansion_tools) .................. 351
  78.2 Documentation on exports (expansion_tools) ........... 351
      imports_meta_pred/3 (pred) ...................... 351
      body_expander/6 (pred) ........................... 351
      arg_expander/6 (pred) ............................ 352
  78.3 Documentation on internals (expansion_tools) .......... 353
      expander_pred/1 (prop) .......................... 353
  78.4 Known bugs and planned improvements (expansion_tools)
      ........................................................................ 353

79 Low-level concurrency/multithreading primitives
                                             ........................................... 355
  79.1 Usage and interface (concurrency) ................... 355
  79.2 Documentation on exports (concurrency) ............. 355
      eng_call/4 (pred) ................................. 355
      eng_call/3 (pred) .................................. 356
      eng_backtrack/2 (pred) ............................ 356
      eng_cut/1 (pred) ................................... 357
      eng_release/1 (pred) ............................... 357
      eng_wait/1 (pred) .................................. 357
      eng_kill/1 (pred) ................................... 357
      eng_killothers/0 (pred) ............................ 358
      eng_self/1 (pred) ................................... 358
      goal_id/1 (pred) .................................... 358
      eng_goal_id/1 (pred) ............................... 358
      eng_status/0 (pred) ................................ 358
      lock_atom/1 (pred) .................................. 358
      unlock_atom/1 (pred) ............................... 358
      atom_lock_state/2 (pred) ......................... 359
      (concurrent)/1 (pred) .............................. 359
  79.3 Known bugs and planned improvements (concurrency) ... 360

80 All solutions concurrent predicates .......... 361
  80.1 Usage and interface (conc_aggregates) ................ 361
  80.2 Documentation on exports (conc_aggregates) ........ 361
      findall/3 (pred) .................................... 361
  80.3 Known bugs and planned improvements (conc_aggregates)
      ........................................................................ 361

81 The socket interface ............................... 363
  81.1 Usage and interface (sockets) ......................... 363
  81.2 Documentation on exports (sockets) ................... 363
      connect_to_socket/3 (pred) ....................... 363
      socket_recv/2 (pred) .............................. 363
      socket_type/1 (regtype) ......................... 364
      shutdown_type/1 (regtype) ....................... 364
      hostname_address/2 (pred) ....................... 364
      socket_shutdown/2 (pred) ......................... 364
      socket_recv_code/3 (pred) ....................... 365
      socket_send/2 (pred) .............................. 365
      select_socket/5 (pred) ............................ 365
      socket_accept/2 (pred) ............................ 366
      bind_socket/3 (pred) .............................. 366
      connect_to_socket_type/4 (pred) ............... 366
82  Sockets I/O ............................. 367
   82.1  Usage and interface (sockets_io) ....................... 367
   82.2  Documentation on exports (sockets_io) ............. 367
          serve_socket/3 (pred) ........................... 367
          safe_write/2 (pred) ............................ 367

83  The Ciao Make Package ................. 369
   83.1  Usage and interface (make) ........................ 369
   83.2  Other information (make) ........................ 369
          83.2.1  The Dependency Rules .................... 369
          83.2.2  Specifying Paths ....................... 371
          83.2.3  Documenting Rules ..................... 371
          83.2.4  An Example of a Makefile .............. 371

84  Predicates Available When Using The Make
    Package ...................................... 375
   84.1  Usage and interface (make_rt) .................... 375
   84.2  Documentation on exports (make_rt) .............. 375
          make/1 (pred) .................................. 375
          target/1 (regtype) ............................ 375
          make_option/1 (pred) .......................... 375
          verbose_message/2 (pred) .................... 376
          call_unknown/1 (pred) ........................ 376
          dyn_load_cfg_module_into_make/1 (pred) .... 376

85  system_extra (library) .................. 377
   85.1  Usage and interface (system_extra) ............... 377
   85.2  Documentation on exports (system_extra) ....... 377
          del_dir_if_empty/1 (pred) .................... 377
          move_files/2 (pred) .......................... 377
          move_file/2 (pred) ............................ 378
          copy_files/2 (pred) ........................... 378
          cat/2 (pred) ................................... 378
          cat_append/2 (pred) ........................... 378
          convert_permissions/4 (pred) ............... 378
          symbolic_link/2 (pred) ....................... 378
          symbolic_link/3 (pred) ....................... 378
          delete_files/1 (pred) ........................ 378
          delete_files_nofail/1 (pred) ................. 378
          del_file_nofail/1 (pred) ..................... 378
          del_file_nofail/2 (pred) ..................... 379
          del_endings_nofail/2 (pred) ................... 379
          ls/3 (pred) .................................... 379
          ls/2 (pred) .................................... 379
          filter-alist_pattern/3 (pred) ............... 379
          (-)/1 (pred) ................................... 380
          do/2 (pred) .................................... 380
          set_perms/2 (pred) ............................ 380
          readf/2 (pred) .................................. 380
          datime_string/1 (pred) ....................... 380
          datime_string/2 (pred) ....................... 380
          datime_atom/1 (pred) ........................ 380
          datime_atom/2 (pred) ........................ 380
          all_values/2 (pred) ........................... 380
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_tr_nl/2 (pred)</td>
<td>380</td>
</tr>
<tr>
<td>call unknown/1 (pred)</td>
<td>380</td>
</tr>
<tr>
<td>replace_strings_in_file/3 (pred)</td>
<td>380</td>
</tr>
<tr>
<td>writef/3 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>writef/2 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>add_suffix/3 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>add_prefix/3 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>write_list/3 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>write_dir/2 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>delete_fileless/1 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>etags/2 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>etagss/2 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>get_dirs/2 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>get_rec_dirs/2 (pred)</td>
<td>381</td>
</tr>
<tr>
<td>get_rec_dirs/3 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>get_rec_dirs/4 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>lss/3 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>lss/4 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>lss/6 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>copy_files_rec/3 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>filter_dirs/3 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>touch/1 (pred)</td>
<td>382</td>
</tr>
<tr>
<td>cyg2win/3 (udreexp)</td>
<td>382</td>
</tr>
<tr>
<td>rename_file/2 (udreexp)</td>
<td>382</td>
</tr>
<tr>
<td>delete_directory/1 (udreexp)</td>
<td>382</td>
</tr>
<tr>
<td>delete_file/1 (udreexp)</td>
<td>382</td>
</tr>
<tr>
<td>chmod/3 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>chmod/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>fmode/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>modif_time/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>modif_time/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>file_properties/6 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>file_property/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>file_exists/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>file_exists/1 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>mktemp/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>directory_files/2 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>exec/3 (udreexp)</td>
<td>383</td>
</tr>
<tr>
<td>exec/4 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>popen_mode/1 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>popen/3 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>system/2 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>system/1 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>shell/2 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>shell/1 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>shell/0 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>cd/1 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>working_directory/2 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>make_dirpath/1 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>make_dirpath/2 (udreexp)</td>
<td>384</td>
</tr>
<tr>
<td>make_directory/1 (udreexp)</td>
<td>385</td>
</tr>
<tr>
<td>make_directory/2 (udreexp)</td>
<td>385</td>
</tr>
<tr>
<td>umask/2 (udreexp)</td>
<td>385</td>
</tr>
<tr>
<td>current_executable/1 (udreexp)</td>
<td>385</td>
</tr>
<tr>
<td>current_host/1 (udreexp)</td>
<td>385</td>
</tr>
<tr>
<td>get_pid/1 (udreexp)</td>
<td>385</td>
</tr>
</tbody>
</table>
extract_paths/2 (udreexp) .................................. 385
copy_file/2 (udreexp) ..................................... 385
del_env/1 (udreexp) ...................................... 385
set_env/2 (udreexp) ...................................... 385
current_env/2 (udreexp) .................................. 385
setenvstr/2 (udreexp) .................................... 385
getenvstr/2 (udreexp) .................................... 386
datetime_struct/1 (udreexp) .............................. 386
datetime/9 (udreexp) ..................................... 386
datetime/1 (udreexp) ..................................... 386
time/1 (udreexp) .......................................... 386
pause/1 (udreexp) ......................................... 386

PART VII - Ciao Prolog extensions ................. 387

86 Pure Prolog package ................................. 389
86.1 Usage and interface (pure) ......................... 390
86.2 Known bugs and planned improvements (pure) .. 390

87 Multiple Argument Indexing ...................... 391
87.1 Usage and interface (indexer) ..................... 391
87.2 Documentation on internals (indexer) .......... 391
index/1 (decl) ........................................... 391
indexspecs/1 (regtype) .................................. 392
argspec/1 (regtype) ..................................... 392
hash_term/2 (pred) ...................................... 392

88 Higher-order .......................................... 393
88.1 Usage and interface (hiord_rt) .................. 393
88.2 Documentation on exports (hiord_rt) ........... 393
call/1 (pred) ............................................ 393
call/2 (pred) ............................................ 393
SYSCALL/1 (pred) ........................................ 394
$nodebug_call/1 (pred) .................................. 394
$meta_call/1 (pred) ..................................... 394

89 Higher-order predicates ............................ 395
89.1 Usage and interface (hiordlib) ................... 395
89.2 Documentation on exports (hiordlib) ........... 395
map/3 (pred) ............................................. 395
foldl/4 (pred) ............................................ 395
minimum/3 (pred) ........................................ 395

90 Terms with named arguments -records/feature terms ........................................ 397
90.1 Usage and interface (argnames) .................. 397
90.2 Documentation on new declarations (argnames) .. 397
argnames/1 (decl) ...................................... 397
90.3 Other information (argnames) ...................... 398
90.3.1 Using argument names in a toy database ....... 398
90.3.2 Complete code for the zebra example .......... 399
90.4 Known bugs and planned improvements (argnames) .... 400
# Functional notation

- Usage and interface *(functions)* ........................................ 401
- Known bugs and planned improvements *(functions)* ............... 402

# Global (library)

- Usage and interface *(global)* ........................................... 403
- Documentation on exports *(global)* .................................. 403
  - `set_global/2 (pred)` ................................................. 403
  - `get_global/2 (pred)` ............................................... 403
  - `push_global/2 (pred)` .............................................. 403
  - `pop_global/2 (pred)` ................................................ 403
  - `del_global/1 (pred)` ................................................ 403

# Independent and-parallel execution

- Usage and interface *(andprolog)* .................................... 405
- Documentation on internals *(andprolog)* .......................... 405
  - `& /2 (pred)` ................................................................ 405
  - `active_agents/1 (pred)` .............................................. 405
  - `indep/2 (pred)` .......................................................... 405
  - `indep/1 (pred)` .......................................................... 406
- Known bugs and planned improvements *(andprolog)* ............ 406

# Andorra execution

- Usage and interface *(andorra)* ........................................ 407
- Documentation on new declarations *(andorra)* .................... 407
  - `determinate/2 (decl)` ............................................... 408
  - `detcond/1 (regtype)` .................................................. 408
  - `path/1 (regtype)` ...................................................... 409
- Other information *(andorra)* ........................................ 409

# Call on determinate

- Usage and interface *(det_hook_rt)* ................................ 411
- Documentation on exports *(det_hook_rt)* .......................... 411
  - `det_try/3 (pred)` ...................................................... 411
- Other information *(det_hook_rt)* ................................... 412
- Known bugs and planned improvements *(det_hook_rt)* ......... 412

# Miscellaneous predicates

- Usage and interface *(odd)* ............................................. 413
- Documentation on exports *(odd)* .................................... 413
  - `setarg/3 (pred)` ....................................................... 413
  - `undo/1 (pred)` .......................................................... 413

# Delaying predicates *(freeze)*

- Usage and interface *(freeze)* ........................................ 415
- Documentation on exports *(freeze)* ................................ 415
  - `freeze/2 (pred)` ...................................................... 415
  - `frozen/2 (pred)` ....................................................... 415
- Documentation on multi-locales *(freeze)* ......................... 416
  - `$combine_attr/4 (pred)` .......................................... 416
  - `$check_attr/3 (pred)` .............................................. 416
98  Delaying predicates (when) .......................... 417
  98.1 Usage and interface (when) ......................... 418
  98.2 Documentation on exports (when) .................. 418
  wakeu_ac/1 (pred) .................................. 418
  98.3 Documentation on multiple (when) .................. 419
  $combine_attr/4 (pred) ............................. 419
  $check_attr/3 (pred) ............................... 419
  98.4 Known bugs and planned improvements (when) .... 419

99  Active modules (high-level distributed execution) .......................... 421
  99.0.1 Active module name servers ...................... 422
  99.0.2 Active modules as agents ....................... 423
  99.1 Usage and interface (actmods) ...................... 424
  99.2 Documentation on new declarations (actmods) ...... 424
  use_active_module/2 (decl) .......................... 424
  99.3 Known bugs and planned improvements (actmods) .. 424

100 Breadth-first execution .............................. 425
  100.1 Usage and interface (bf) .......................... 425
  100.2 Known bugs and planned improvements (bf) ...... 426

101 Iterative-deepening execution ........................ 427
  101.1 Usage and interface (id) ......................... 428

102 Constraint programming over rationals ............ 429
  102.1 Usage and interface (clpq) ....................... 429
  102.2 Other information (clpq) ....................... 429
  102.2.1 Some CLP(Q) examples ....................... 429
  102.3 Known bugs and planned improvements (clpq) .... 430

103 Constraint programming over reals ............... 431
  103.1 Usage and interface (clpr) ....................... 431
  103.2 Other information (clpr) ....................... 431
  103.2.1 Some CLP(R) examples ....................... 431
  103.3 Known bugs and planned improvements (clpr) ... 433

104 Fuzzy Prolog ........................................ 435
  104.1 Usage and interface (fuzzy) ....................... 435
  104.2 Documentation on new declarations (fuzzy) ...... 436
  aggr/1 (decl) ..................................... 436
  104.3 Documentation on exports (fuzzy) ............... 436
  :# /2 (pred) ...................................... 436
  fuzzy_predicate/1 (pred) ............................ 436
  fuzzy/1 (pred) ..................................... 436
  fnot/1 (pred) ...................................... 437
  :" /2 (pred) ...................................... 437
  fuzzybody/1 (prop) .................................. 438
  faggregator/1 (regtype) ............................. 438
  => /4 (pred) ...................................... 438
  104.4 Other information (fuzzy) ....................... 439
  104.5 Known bugs and planned improvements (fuzzy) ... 439
## 105 Object oriented programming

- **105.1 Early examples** .......................................................... 441
- **105.2 Recommendations on when to use objects** .................. 445
- **105.3 Limitations on object usage** ...................................... 445

## 106 Declaring classes and interfaces

- **106.1 Usage and interface (class)** ........................................ 447
- **106.2 Documentation on new declarations (class)** ............... 448
  - export/1 (decl) .............................................................. 448
  - public/1 (decl) ............................................................ 448
  - inheritable/1 (decl) ...................................................... 448
  - (data)/1 (decl) ............................................................ 448
  - (dynamic)/1 (decl) ....................................................... 449
  - (concurrent)/1 (decl) ................................................... 449
  - inherit_class/1 (decl) ................................................... 449
  - implements/1 (decl) ...................................................... 450
  - virtual/1 (decl) .......................................................... 450
- **106.3 Documentation on exports (class)** ............................. 451
  - inherited/1 (pred) ......................................................... 451
  - self/1 (pred) .............................................................. 451
  - constructor/0 (pred) ..................................................... 451
  - destructor/0 (pred) ....................................................... 452
- **106.4 Other information (class)** ....................................... 452
  - **106.4.1 Class and Interface error reporting at compile time** .... 453
  - **106.4.2 Class and Interface error reporting at run time** ........ 456
  - **106.4.3 Normal Prolog module system interaction** ............... 456
- **106.5 Known bugs and planned improvements (class)** .......... 457

## 107 Compile-time usage of objects

- **107.1 Usage and interface (objects)** ................................... 459
- **107.2 Documentation on new declarations (objects)** ............ 459
  - use_class/1 (decl) ...................................................... 459
  - instance_of/2 (decl) .................................................... 459
  - new/2 (decl) ............................................................. 460
- **107.3 Other information (objects)** .................................... 460
  - **107.3.1 Error reporting at compile time (objects)** ............ 461
  - **107.3.2 Error reporting at run time (objects)** ................. 463

## 108 Run time usage of objects

- **108.1 Usage and interface (objects_rt)** ............................... 465
- **108.2 Documentation on exports (objects_rt)** ..................... 465
  - new/2 (pred) .............................................................. 465
  - instance_of/2 (pred) .................................................... 466
  - derived_from/2 (pred) .................................................. 467
  - interface/2 (pred) ....................................................... 467
  - instance_codes/2 (pred) .............................................. 468
  - destroy/1 (pred) ........................................................ 468
  - use_class/1 (pred) ...................................................... 468
  - constructor/1 (prop) ................................................... 468
  - class_name/1 (prop) .................................................... 469
  - interface_name/1 (prop) .............................................. 469
  - instance_id/1 (prop) ................................................... 469
  - class_source/1 (prop) .................................................. 469
interface_source/1 (prop) .................................. 469
method_spec/1 (prop) ........................................ 469
virtual_method_spec/1 (prop) ............................... 469
108.3 Known bugs and planned improvements (objects_rt) .... 470

109 The Ciao Remote Services Package .......... 471
109.1 Usage and interface (remote) ...................... 471
109.2 Documentation on exports (remote) ............... 471
(Θ)/2 (udreexp) ............................................. 471
server_stop/1 (udreexp) .................................... 471
server_stop/1 (udreexp) .................................... 471
server_trace/1 (udreexp) ................................... 471
server_trace/1 (udreexp) ................................... 471
server_notrace/1 (udreexp) ................................. 471
server_notrace/1 (udreexp) ................................. 471
109.3 Known bugs and planned improvements (remote) .... 472

PART VIII - Interfaces to other languages and systems ................. 473

110 Foreign Language Interface ....................... 475
110.1 Declaration of Types .................................. 475
110.2 Equivalence between Ciao Prolog and C types ............ 475
110.3 Equivalence between Ciao Prolog and C modes ............ 476
110.4 Custom access to Prolog from C ....................... 476
110.4.1 Term construction .................................... 477
110.4.2 Testing the Type of a Term ......................... 478
110.4.3 Term navigation ...................................... 478
110.4.4 Testing for Equality and Performing Unification .... 479
110.4.5 Raising Exceptions ................................. 479
110.4.6 Creating and disposing of memory chunks ............ 480
110.4.7 Calling Prolog from C ............................... 480
110.5 Examples ................................................. 480
110.5.1 Mathematical functions ............................. 481
110.5.2 Addresses and C pointers ........................... 481
110.5.3 Lists of bytes and buffers ........................... 481
110.5.4 Lists of integers .................................... 482
110.5.5 Strings and atoms .................................... 482
110.5.6 Arbitrary Terms ..................................... 483
110.5.7 Exceptions ............................................ 483
110.5.8 Testing number types and using unbound length integers .............................................. 484
110.6 Usage and interface (foreign_interface) ............... 484
111 Foreign Language Interface Properties ..... 485
111.1 Usage and interface (foreign_interface_properties) ..... 485
111.2 Documentation on exports (foreign_interface_properties)
   ........................................................................ 485
   int_list/1 (regtype) ........................................... 485
   byte_list/1 (regtype) ......................................... 485
   byte/1 (regtype) ............................................... 485
   null/1 (regtype) ............................................... 485
   address/1 (regtype) ........................................... 485
   any_term/1 (regtype) ........................................ 486
   native/1 (prop) ............................................... 486
   native/2 (prop) ............................................... 486
   size_of/3 (prop) .............................................. 486
   foreign/1 (prop) ............................................. 486
   foreign/2 (prop) ............................................. 486
   returns/2 (prop) ............................................. 486
   do_not_free/2 (prop) ...................................... 486
111.3 Documentation on internals (foreign_interface_properties)
   ........................................................................ 487
   use_foreign_source/1 (decl) .............................. 487
   use_foreign_source/2 (decl) .............................. 487
   use_foreign_library/1 (decl) ............................. 487
   use_foreign_library/2 (decl) ............................. 487
   extra_compiler_opts/1 (decl) ............................ 487
   extra_compiler_opts/2 (decl) ............................ 487
   use_compiler/1 (decl) ...................................... 488
   use_compiler/2 (decl) ...................................... 488
   extra_linker_opts/1 (decl) ............................... 488
   extra_linker_opts/2 (decl) ............................... 488
   use_linker/1 (decl) ......................................... 488
   use_linker/2 (decl) ......................................... 489
   foreign_inline/2 (decl) ................................... 489
111.4 Known bugs and planned improvements
   (foreign_interface_properties) ............................ 489

112 Utilities for on-demand compilation of foreign files................................. 491
112.1 Usage and interface (foreign_compilation) ........................................... 491
112.2 Documentation on exports (foreign_compilation) ............................... 491
   compiler_and_opts/2 (pred) ............................... 491
   linker_and_opts/2 (pred) .................................. 491

113 Foreign Language Interface Builder .......... 493
113.1 Usage and interface (build_foreign_interface) ................................. 493
113.2 Documentation on exports (build_foreign_interface) ....................... 493
   build_foreign_interface/1 (pred) ....................... 493
   rebuild_foreign_interface/1 (pred) .................... 493
   build_foreign_interface.explicit.decls/2 (pred) .... 494
   rebuild_foreign_interface.explicit.decls/2 (pred) ... 494
   build_foreign_interface.object/1 (pred) .......... 494
   rebuild_foreign_interface.object/1 (pred) ....... 495
   do_interface/1 (pred) ..................................... 495
114 Interface to daVinci .......................... 497
114.1 Usage and interface (davinci) ................. 497
114.2 Documentation on exports (davinci) ............ 497
davinci/0 (pred) .................................. 497
topd/0 (pred) ...................................... 497
davinci_get/1 (pred) ................................ 497
davinci_get_all/1 (pred) ............................ 497
davinci_put/1 (pred) ................................ 498
davinci_quit/0 (pred) ................................ 498
davinci_get/1 (pred) ................................ 498
davinci_get_all/1 (pred) ............................ 498
davinci_put/1 (pred) ................................ 498
davinci_quit/0 (pred) ................................ 498
114.3 Documentation on internals (davinci) ......... 498
davinci_command/1 (prop) ......................... 498
ugraph/1 (prop) .................................... 499
lgraph/1 (prop) ..................................... 499
115 The Tcl/Tk interface .............................. 501
115.1 Usage and interface (tcltk) ..................... 504
115.2 Documentation on exports (tcltk) .............. 504
tcl/new/1 (pred) ................................... 504
tcl/eval/3 (pred) .................................... 504
tcl/delete/1 (pred) .................................. 505
tcl/event/3 (pred) .................................. 505	cInterpreter/1 (registype) ......................... 505
tclCommand/1 (registype) .......................... 505
tk/event_loop/1 (pred) ............................. 506
tk/main_loop/1 (pred) ............................... 506
tk/new/2 (pred) .................................... 506
tk/next_event/2 (pred) ............................. 507
116 Low level interface library to Tcl/Tk ...... 509
116.1 Usage and interface (tcltk_low_level) .......... 509
116.2 Documentation on exports (tcltk_low_level) ... 509
new_interp/1 (pred) .................................. 509
new_interp/2 (pred) .................................. 509
new_interp_file/1 (pred) ......................... 510
tcltk/2 (pred) ...................................... 510
tcltk_raw_code/2 (pred) ......................... 510
receive_result/2 (pred) ............................ 510
send_term/2 (pred) .................................. 511
receive_event/2 (pred) ............................ 511
receive_list/2 (pred) ............................... 511
receive_confirm/2 (pred) ......................... 512
delete/1 (pred) .................................... 512
116.3 Documentation on internals (tcltk_low_level) ... 512
core/1 (pred) ...................................... 512
### 117 The Tcl/Tk Class Interface

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>117.1</td>
<td>Usage and interface (\texttt{window_class})</td>
</tr>
<tr>
<td>117.2</td>
<td>Documentation on multifactors (\texttt{window_class})</td>
</tr>
</tbody>
</table>

- \texttt{$\text{class}$/1 (pred)}
- \texttt{class$super/2 (pred)}
- \texttt{class$initial\_state/3 (pred)}
- \texttt{class$virtual/6 (pred)}
- \texttt{class$attr\_template/4 (pred)}
- \texttt{class$default\_cons/1 (pred)}
- \texttt{class$constructor/4 (pred)}
- \texttt{class$destructor/3 (pred)}
- \texttt{class$implements/2 (pred)}

### 118 widget\_class (library)

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>118.1</td>
<td>Usage and interface (\texttt{widget_class})</td>
</tr>
<tr>
<td>118.2</td>
<td>Documentation on exports (\texttt{widget_class})</td>
</tr>
</tbody>
</table>

- \texttt{text\_characters/1 (pred)}
- \texttt{font\_type/1 (pred)}
- \texttt{background\_color/1 (pred)}
- \texttt{borderwidth\_value/1 (pred)}
- \texttt{foreground\_color/1 (pred)}
- \texttt{highlight\_background\_color/1 (pred)}
- \texttt{highlight\_color/1 (pred)}
- \texttt{width\_value/1 (pred)}
- \texttt{relief\_type/1 (pred)}
- \texttt{side\_type/1 (pred)}
- \texttt{expand\_value/1 (pred)}
- \texttt{fill\_type/1 (pred)}
- \texttt{pady\_value/1 (pred)}
- \texttt{row\_value/1 (pred)}
- \texttt{rowspan\_value/1 (pred)}
- \texttt{column\_value/1 (pred)}
- \texttt{columnspan\_value/1 (pred)}
- \texttt{event\_type\_widget/1 (pred)}
- \texttt{action\_widget/3 (pred)}
- \texttt{action\_widget/1 (pred)}
- \texttt{creation\_options/1 (pred)}
- \texttt{creation\_position/1 (pred)}
- \texttt{creation\_position\_grid/1 (pred)}
- \texttt{creation\_bind/1 (pred)}

### 119 menu\_class (library)

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>119.1</td>
<td>Usage and interface (\texttt{menu_class})</td>
</tr>
<tr>
<td>119.2</td>
<td>Documentation on exports (\texttt{menu_class})</td>
</tr>
</tbody>
</table>

- \texttt{name\_menu/1 (pred)}
- \texttt{menu\_data/1 (pred)}
- \texttt{label\_value/1 (pred)}
- \texttt{tearoff\_value/1 (pred)}
- \texttt{tcl\_name/1 (pred)}
- \texttt{creation\_options/1 (pred)}
- \texttt{creation\_options\_entry/1 (pred)}
- \texttt{creation\_menu\_name/1 (pred)}
## 120 canvas_class (library) ........................................ 529

120.1 Usage and interface (canvas_class) ......................... 529
120.2 Documentation on exports (canvas_class) .................. 529

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape/1</td>
<td>529</td>
</tr>
<tr>
<td>option/1</td>
<td>529</td>
</tr>
<tr>
<td>canvas_class/0</td>
<td>529</td>
</tr>
<tr>
<td>canvas_class/1</td>
<td>530</td>
</tr>
<tr>
<td>destructor/0</td>
<td>530</td>
</tr>
<tr>
<td>show/0</td>
<td>530</td>
</tr>
<tr>
<td>hide_/0</td>
<td>530</td>
</tr>
<tr>
<td>show_shape/1</td>
<td>530</td>
</tr>
<tr>
<td>hide_shape/1</td>
<td>530</td>
</tr>
<tr>
<td>width_value/1</td>
<td>531</td>
</tr>
<tr>
<td>height_value/1</td>
<td>531</td>
</tr>
<tr>
<td>side_type/1</td>
<td>531</td>
</tr>
<tr>
<td>expand_value/1</td>
<td>532</td>
</tr>
<tr>
<td>fill_type/1</td>
<td>532</td>
</tr>
<tr>
<td>padx_value/1</td>
<td>532</td>
</tr>
<tr>
<td>pady_value/1</td>
<td>533</td>
</tr>
</tbody>
</table>

120.3 Documentation on multi-files (canvas_class) ............ 533

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$class$/1</td>
<td>533</td>
</tr>
<tr>
<td>class$super/2</td>
<td>533</td>
</tr>
<tr>
<td>class$initial_state/3</td>
<td>533</td>
</tr>
<tr>
<td>class$virtual/6</td>
<td>533</td>
</tr>
<tr>
<td>class$attr_template/4</td>
<td>534</td>
</tr>
<tr>
<td>class$default_cons/1</td>
<td>534</td>
</tr>
<tr>
<td>class$constructor/4</td>
<td>534</td>
</tr>
<tr>
<td>class$destructor/3</td>
<td>534</td>
</tr>
<tr>
<td>class$implements/2</td>
<td>534</td>
</tr>
</tbody>
</table>

## 121 button_class (library) ................................. 535

121.1 Usage and interface (button_class) ....................... 535
121.2 Documentation on exports (button_class) ................ 535

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_button/1</td>
<td>535</td>
</tr>
</tbody>
</table>

## 122 checkbutton_class (library) .......................... 537

122.1 Usage and interface (checkbutton_class) ............... 537
122.2 Documentation on exports (checkbutton_class) ......... 537

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_value/1</td>
<td>537</td>
</tr>
</tbody>
</table>

## 123 radiobutton_class (library) .......................... 539

123.1 Usage and interface (radiobutton_class) ............... 539
123.2 Documentation on exports (radiobutton_class) ......... 539

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_value/1</td>
<td>539</td>
</tr>
</tbody>
</table>

## 124 entry_class (library) .................................. 541

124.1 Usage and interface (entry_class) ....................... 541
124.2 Documentation on exports (entry_class) ............... 541

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>textvariable_entry/1</td>
<td>541</td>
</tr>
<tr>
<td>textvariablevalue_string/1</td>
<td>541</td>
</tr>
<tr>
<td>textvariablevalue_number/1</td>
<td>542</td>
</tr>
<tr>
<td>justify_entry/1</td>
<td>542</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>125</td>
<td>label_class (library)</td>
</tr>
<tr>
<td>125.1</td>
<td>Usage and interface (label_class)</td>
</tr>
<tr>
<td>125.2</td>
<td>Documentation on exports (label_class)</td>
</tr>
<tr>
<td></td>
<td>textvariable_label/1 (pred)</td>
</tr>
<tr>
<td>126</td>
<td>menubutton_class (library)</td>
</tr>
<tr>
<td>126.1</td>
<td>Usage and interface (menubutton_class)</td>
</tr>
<tr>
<td>126.2</td>
<td>Documentation on exports (menubutton_class)</td>
</tr>
<tr>
<td></td>
<td>menu_name/1 (pred)</td>
</tr>
<tr>
<td>127</td>
<td>menu_entry_class (library)</td>
</tr>
<tr>
<td>127.1</td>
<td>Usage and interface (menu_entry_class)</td>
</tr>
<tr>
<td>127.2</td>
<td>Documentation on exports (menu_entry_class)</td>
</tr>
<tr>
<td></td>
<td>set_name/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>set_action/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>label_value/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>menu_name/1 (pred)</td>
</tr>
<tr>
<td>128</td>
<td>shape_class (library)</td>
</tr>
<tr>
<td>128.1</td>
<td>Usage and interface (shape_class)</td>
</tr>
<tr>
<td>128.2</td>
<td>Documentation on multi-files (shape_class)</td>
</tr>
<tr>
<td></td>
<td>$class$/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$super/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$initial_state/3 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$virtual/6 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$attr_template/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$default_cons/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$constructor/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$destructor/3 (pred)</td>
</tr>
<tr>
<td></td>
<td>$class$implements/2 (pred)</td>
</tr>
<tr>
<td>129</td>
<td>arc_class (library)</td>
</tr>
<tr>
<td>129.1</td>
<td>Usage and interface (arc_class)</td>
</tr>
<tr>
<td>129.2</td>
<td>Documentation on exports (arc_class)</td>
</tr>
<tr>
<td></td>
<td>coord/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>width/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>height/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>center/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>angle_start/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>style_type/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>outline_color/1 (pred)</td>
</tr>
<tr>
<td>130</td>
<td>oval_class (library)</td>
</tr>
<tr>
<td>130.1</td>
<td>Usage and interface (oval_class)</td>
</tr>
<tr>
<td>130.2</td>
<td>Documentation on exports (oval_class)</td>
</tr>
<tr>
<td></td>
<td>coord/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>width/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>height/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>center/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>outline_color/1 (pred)</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>131</td>
<td>poly_class (library)</td>
</tr>
<tr>
<td>131.1</td>
<td>Usage and interface (poly_class)</td>
</tr>
<tr>
<td>131.2</td>
<td>Documentation on multfiles (poly_class)</td>
</tr>
<tr>
<td></td>
<td>$class$/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$super/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$initial_state/3 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$virtual/6 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$attr_template/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$default_cons/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$constructor/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$destructor/3 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$implements/2 (pred)</td>
</tr>
<tr>
<td>132</td>
<td>line_class (library)</td>
</tr>
<tr>
<td>132.1</td>
<td>Usage and interface (line_class)</td>
</tr>
<tr>
<td>132.2</td>
<td>Documentation on multfiles (line_class)</td>
</tr>
<tr>
<td></td>
<td>$class$/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$super/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$initial_state/3 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$virtual/6 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$attr_template/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$default_cons/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$constructor/4 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$destructor/3 (pred)</td>
</tr>
<tr>
<td></td>
<td>class$implements/2 (pred)</td>
</tr>
<tr>
<td>133</td>
<td>text_class (library)</td>
</tr>
<tr>
<td>133.1</td>
<td>Usage and interface (text_class)</td>
</tr>
<tr>
<td>133.2</td>
<td>Documentation on exports (text_class)</td>
</tr>
<tr>
<td></td>
<td>coord/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>point/2 (pred)</td>
</tr>
<tr>
<td></td>
<td>text_characters/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>anchor/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>font_type/1 (pred)</td>
</tr>
<tr>
<td></td>
<td>justify_text/1 (pred)</td>
</tr>
<tr>
<td>134</td>
<td>The PiLLoW Web programming library</td>
</tr>
<tr>
<td>134.1</td>
<td>Installing PiLLoW</td>
</tr>
<tr>
<td>134.2</td>
<td>Usage and interface (pillow)</td>
</tr>
</tbody>
</table>
138  Persistent predicate database .......................... 589
   138.1  Introduction to persistent predicates ................. 589
   138.2  Persistent predicates, files, and relational databases 589
   138.3  Using file-based persistent predicates ............... 590
   138.4  Implementation Issues ................................ 590
   138.5  Defining an initial database .......................... 591
   138.6  Using persistent predicates from the top level ....... 591
   138.7  Usage and interface (persdbrt) ....................... 592
   138.8  Documentation on exports (persdbrt) ................. 592
         asserta_fact/1 (pred) .................................. 592
         assertz_fact/1 (pred) .................................. 592
         retract_fact/1 (pred) .................................. 593
         retractall_fact/1 (pred) ............................... 593
         asserta_fact/1 (pred) .................................. 593
         assertz_fact/1 (pred) .................................. 593
         retract_fact/1 (pred) .................................. 593
         retractall_fact/1 (pred) ............................... 594
         initialize_db/0 (pred) .................................. 594
         make_persistent/2 (pred) ............................... 594
         update_files/0 (pred) .................................. 594
         update_files/1 (pred) .................................. 595
   138.9  Documentation on multifiles (persdbrt) ............... 595
         persistent_dir/2 (pred) ................................ 595
         $is_persistent/2 (pred) ................................ 595
   138.10 Documentation on internals (persdbrt) ............... 595
         persistent/2 (decl) ..................................... 595
         keyword/1 (pred) ....................................... 596
         directoryname/1 (regtype) .............................. 596
   138.11 Known bugs and planned improvements (persdbrt) ... 596
139  Using the persdb library ................................. 597
   139.1  An example of persistent predicates (static version) 597
   139.2  An example of persistent predicates (dynamic version) 597
   139.3  A simple application / a persistent queue .......... 598
140  Filed predicates .......................................... 599
   140.1  Usage and interface (factsdb_rt) ...................... 599
   140.2  Documentation on exports (factsdb_rt) ............... 599
         asserta_fact/1 (pred) .................................. 599
         assertz_fact/1 (pred) .................................. 600
         call/1 (pred) .......................................... 600
         current_fact/1 (pred) .................................. 600
         retract_fact/1 (pred) .................................. 600
   140.3  Documentation on multifiles (factsdb_rt) .......... 601
         $factsdb cached_goal/3 (pred) .......................... 601
         persistent_dir/2 (pred) ................................ 601
         file_alias/2 (pred) ..................................... 601
   140.4  Documentation on internals (factsdb_rt) ............ 601
         facts/2 (decl) .......................................... 601
         keyword/1 (pred) ....................................... 602
   140.5  Known bugs and planned improvements (factsdb_rt) ... 602
141 SQL persistent database interface ........... 603
141.1 Implementation of the Database Interface .......... 603
141.2 Example(s) .................................................. 603
141.3 Usage and interface (persdbt_mysql) .............. 606
141.4 Documentation on exports (persdbt_mysql) ....... 607
     init_sql_persdb/0 (pred) .......... 607
     dbassertz_fact/1 (pred) .......... 607
     dbretract_fact/1 (pred) .......... 607
     dbc Retract_fact/1 (pred) ........ 607
     dbretractall_fact/1 (pred) ...... 607
     make_sql_persistent/3 (pred) ... 608
     dbfindall/4 (pred) ............... 608
     dbcall/2 (pred) ................... 608
     sql_query/3 (pred) ............... 609
     sql_get_tables/2 (pred) .......... 609
     sql_table_types/3 (pred) ........ 610
     socketname/1 (regtype) .......... 610
     dbname/1 (regtype) ............... 610
     user/1 (regtype) .................. 610
     passwd/1 (regtype)................. 611
     projterm/1 (regtype) ............. 611
     querybody/1 (regtype) ............ 611
     sqltype/1 (udreexp) .............. 611
141.5 Documentation on multi®les (persdbt_mysql) ...... 611
     sql_persistent_location/2 (pred) .. 611
141.6 Documentation on internals (persdbt_mysql) ...... 612
     tuple/1 (regtype) ................. 612
     dbconnection/1 (regtype) ......... 612
     sql_persistent/3 (decl) .......... 612
     db_query/4 (pred) ................ 612
     db_query_one_tuple/4 (pred) ...... 613
     sql_query_one_tuple/3 (pred) .... 613
141.7 Known bugs and planned improvements (persdbt_mysql)
     ......................................................... 614
142 Prolog to SQL translator ...................... 615
142.1 Usage and interface (pl2sql) ..................... 616
142.2 Documentation on exports (pl2sql) ............... 616
     pl2sqlstring/3 (pred) ............ 616
     querybody/1 (regtype) .......... 616
     projterm/1 (regtype) .......... 617
     sqlstring/1 (regtype) .......... 617
     pl2sqlterm/3 (pred) ............. 618
     sqlterm2string/2 (pred) .......... 618
     sqltype/1 (udreexp) .......... 618
142.3 Documentation on multi®les (pl2sql) .............. 618
     sql_relation/3 (pred) .......... 618
     sql_attribute/4 (pred) .......... 619
142.4 Documentation on internals (pl2sql) .............. 619
     query_generation/3 (pred) ....... 619
     translate_conjunction/5 (pred) .. 620
     translate_goal/5 (pred) .......... 620
     translate_arithmetic_function/5 (pred) .. 620
     translate_comparison/5 (pred) ... 621
     aggregate_function/3 (pred) ..... 621
     comparison/2 (pred) .......... 621
143 Low-level socket interface to SQL/ODBC databases .......................... 623
143.1 Usage and interface (mysql_client) ........................................ 623
143.2 Documentation on exports (mysql_client) .................................. 623
  mysql_connect/5 (pred) .......................................................... 623
  dbconnection/1 (regtype) ...................................................... 623
  mysql_query/3 (pred) .......................................................... 623
  mysql_query_one_tuple/3 (pred) ............................................. 623
  dbqueryconnection/1 (regtype) ............................................. 624
  mysql_free_query_connection/1 (pred) .................................. 624
  mysql_fetch/2 (pred) .......................................................... 624
  mysql_get_tables/2 (pred) .................................................. 624
  mysql_table_types/3 (pred) .................................................. 624
  mysql_disconnect/1 (pred) .................................................... 624
144 Types for the Low-level interface to SQL databases ................................. 625
144.1 Usage and interface (db_client_types) ...................................... 625
144.2 Documentation on exports (db_client_types) ................................ 625
  socketname/1 (regtype) ....................................................... 625
  dbname/1 (regtype) ............................................................ 625
  user/1 (regtype) ............................................................... 625
  passwd/1 (regtype) ............................................................ 625
  answertableterm/1 (regtype) ............................................... 626
  tuple/1 (regtype) .............................................................. 626
  answertupleterm/1 (regtype) ............................................... 626
  sqlstring/1 (regtype) ......................................................... 626
145 sqltypes (library) ................................................................. 627
145.1 Usage and interface (sqltypes) .............................................. 627
145.2 Documentation on exports (sqltypes) ....................................... 627
  sqltype/1 (regtype) ............................................................ 627
  accepted_type/2 (pred) ....................................................... 627
  get_type/2 (pred) .............................................................. 627
  type_compatible/2 (pred) .................................................... 628
  type_union/3 (pred) ........................................................... 628
  sybasetype/1 (regtype) ....................................................... 628
  sybase2sqltypes_list/2 (pred) ............................................. 629
  sybase2sqltype/2 (pred) .................................................... 629
  postgres2sqltypes_list/2 (pred) .......................................... 629
  postgres2sqltype/2 (pred) .................................................. 629
146 persdbtr_sql (library) ............................................................. 631
146.1 Usage and interface (persdbtr_sql) ....................................... 631
146.2 Documentation on exports (persdbtr_sql) .................................. 631
  sql_persistent_tr/2 (pred) .................................................. 631
  sql_goal_tr/2 (pred) .......................................................... 631
  dbId/2 (pred) ................................................................. 631
147 pl2sqlinsert (library) ............... 633
  147.1 Usage and interface (pl2sqlinsert) ............... 633
  147.2 Documentation on exports (pl2sqlinsert) ....... 633
    pl2sqlInsert/2 (pred) .................. 633
  147.3 Documentation on multi-files (pl2sqlinsert) .... 633
    sql_relation/3 (pred) ........................ 633
    sql_attribute/4 (pred) .................... 633

148 Prolog to Java Interface .................... 635
  148.1 Prolog to Java Interface Structure .................... 635
    148.1.1 Prolog side of the Java interface .................... 635
    148.1.2 Java side .................................. 635
  148.2 Java event handling from Prolog ...................... 636
  148.3 Java exception handling from Prolog ................. 637
  148.4 Usage and interface (javart) ...................... 637
    java_start/0 (pred) .............................. 637
    java_start/1 (pred) .............................. 637
    java_start/2 (pred) .............................. 638
    java_stop/0 (pred) ................................ 638
    java_connect/2 (pred) ............................ 638
    java_disconnect/0 (pred) .......................... 638
    machine_name/1 (regtype) ....................... 638
    java_constructor/1 (regtype) .................... 639
    java_object/1 (regtype) ......................... 639
    java_event/1 (regtype) ........................... 639
    prolog_goal/1 (regtype) ......................... 639
    java_field/1 (regtype) ........................... 639
    java_use_module/1 (pred) ....................... 639
    java_create_object/2 (pred) ........................ 639
    java_delete_object/1 (pred) ........................ 640
    java_invoke_method/2 (pred) ..................... 640
    java_method/1 (regtype) .......................... 640
    java_get_value/2 (pred) ........................... 641
    java_set_value/2 (pred) ........................... 641
    java_add_listener/3 (pred) ....................... 641
    java_remove_listener/3 (pred) ..................... 642

149 Java to Prolog Interface ...................... 643
  149.1 Usage and interface (jtopl) ...................... 643
  149.2 Documentation on exports (jtopl) ................... 643
    prolog_server/0 (pred) ....................... 643
    prolog_server/1 (pred) ....................... 644
    prolog_server/2 (pred) ....................... 644
    shell_s/0 (pred) .............................. 644
    query_solutions/2 (pred) ........................ 644
    query_requests/2 (pred) ....................... 645
    running_queries/2 (pred) ..................... 645
150 Low-level Prolog to Java socket connection

150.1 Usage and interface (javasock) .......................... 647
150.2 Documentation on exports (javasock) ...................... 647
   bind_socket_interface/1 (pred) .......................... 647
   start_socket_interface/2 (pred) ......................... 647
   stop_socket_interface/0 (pred) ......................... 648
   join_socket_interface/0 (pred) ......................... 648
   java_query/2 (pred) .................................. 648
   java_response/2 (pred) ................................ 648
   prolog_query/2 (pred) ................................ 648
   prolog_response/2 (pred) ................................ 649
   is_connected_to_java/0 (pred) ......................... 649
   java_debug/1 (pred) ................................ 649
   java_debug_redo/1 (pred) ............................ 649
   start_threads/0 (pred) .............................. 649

151 Calling emacs from Prolog ....................... 651

151.1 Usage and interface (emacs) ............................ 652
151.2 Documentation on exports (emacs) ....................... 652
   emacs_edit/1 (pred) .................................. 652
   emacs_edit_nowait/1 (pred) ............................ 652
   emacs_eval/1 (pred) .................................. 652
   emacs_eval_nowait/1 (pred) ............................ 652
   elisp_string/1 (regtype) ............................. 653

152 linda (library) .................. 655

152.1 Usage and interface (linda) ............................ 655
152.2 Documentation on exports (linda) ....................... 655
   linda_client/1 (pred) .................................. 655
   close_client/0 (pred) .................................. 655
   in/1 (pred) ........................................... 655
   in/2 (pred) ........................................... 655
   in_noblock/1 (pred) .................................. 655
   out/1 (pred) ........................................... 655
   rd/1 (pred) ........................................... 656
   rd/2 (pred) ........................................... 656
   rd_noblock/1 (pred) .................................. 656
   rd_findall/3 (pred) .................................. 656
   linda_timeouts/2 (pred) ................................ 656
   halt_server/0 (pred) .................................. 656
   open_client/2 (pred) .................................. 656
   in_stream/2 (pred) ................................... 656
   out_stream/2 (pred) ................................ 656

PART IX - Abstract data types ..................... 657
153 Extendable arrays with logarithmic access time

153.1 Usage and interface (arrays) .................................. 659
153.2 Documentation on exports (arrays) ......................... 659
   new_array/1 (pred) ........................................... 659
   is_array/1 (pred) ............................................ 659
   aref/3 (pred) .................................................. 659
   arefa/3 (pred) ................................................ 659
   arefl/3 (pred) .................................................. 660
   areset/4 (pred) ............................................... 660
   array_to_list/2 (pred) ....................................... 660

154 counters (library) ................................. 661
154.1 Usage and interface (counters) .......................... 661
154.2 Documentation on exports (counters) ................. 661
   setcounter/2 (pred) ........................................... 661
   getcounter/2 (pred) ......................................... 661
   inccounter/2 (pred) ......................................... 661

155 Identity lists ................................. 663
155.1 Usage and interface (idlists) .......................... 663
155.2 Documentation on exports (idlists) .................. 663
   member_0/2 (pred) ......................................... 663
   memberchk/2 (pred) ......................................... 663
   list_insert/2 (pred) ....................................... 663
   add_after/4 (pred) ......................................... 663
   add_before/4 (pred) ....................................... 664
   delete/3 (pred) ............................................. 664
   subtract/3 (pred) .......................................... 664
   union_idlists/3 (pred) .................................... 664

156 Lists of numbers ................................. 665
156.1 Usage and interface (numlists) ....................... 665
156.2 Documentation on exports (numlists) ............... 665
   get_primes/2 (pred) ....................................... 665
   intlist/1 (regtype) ........................................ 665
   numlist/1 (regtype) ....................................... 665
   sum_list/2 (pred) .......................................... 665
   sum_list/3 (pred) .......................................... 666
   sum_list_of_lists/2 (pred) ................................ 666
   sum_list_of_lists/3 (pred) .............................. 666

157 Pattern (regular expression) matching .... 667
157.1 Usage and interface (patterns) ...................... 667
157.2 Documentation on exports (patterns) .............. 667
   match_pattern/2 (pred) .................................... 667
   match_pattern/3 (pred) .................................... 667
   case_insensitive_match/2 (pred) ...................... 667
   letter_match/2 (pred) ..................................... 668
   pattern/1 (regtype) ....................................... 668
   match_pattern_pred/2 (pred) ............................ 668
158 Graphs ................................................. 669
158.1 Usage and interface (graphs) ......................... 669
158.2 Documentation on exports (graphs) .................. 669
dgraph/1 (regtype) ....................................... 669
dlgraph/1 (regtype) ....................................... 669
dgraph_to_ugraph/2 (pred) ................................ 669
dlgraph_to_lgraph/2 (pred) ................................ 670
edges_to_ugraph/2 (pred) .................................. 670
dgraph-to_lgraph/2 (pred) .................................. 670
158.3 Documentation on internals (graphs) ............... 671
pair/1 (regtype) .......................................... 671
triple/1 (regtype) ........................................ 671

159 Unweighted graph-processing utilities ............. 673
159.1 Usage and interface (ugraphs) ....................... 673
159.2 Documentation on exports (ugraphs) ............... 673
vertices_edges_to_ugraph/3 (pred) ....................... 673
neighbors/3 (pred) ........................................ 673
edges/2 (pred) ............................................. 674
del_vertices/3 (pred) ...................................... 674
vertices/2 (pred) .......................................... 674
add_vertices/3 (pred) ...................................... 674
add_edges/3 (pred) ......................................... 674
transpose/2 (pred) ......................................... 674
rooted_subgraph/3 (pred) .................................. 675
point_to/3 (pred) .......................................... 675
ugraph/1 (regtype) ........................................ 675

160 wgraphs (library) ...................................... 677
160.1 Usage and interface (wgraphs) ....................... 677
160.2 Documentation on exports (wgraphs) ............... 677
vertices_edges_to_wgraph/3 (pred) ....................... 677

161 Labeled graph-processing utilities ............... 679
161.1 Usage and interface (lgraphs) ....................... 679
161.2 Documentation on exports (lgraphs) ............... 679
lgraph/2 (regtype) ........................................ 679
vertices_edges_to_lgraph/3 (pred) ....................... 679

162 queues (library) ...................................... 681
162.1 Usage and interface (queues) ......................... 681
162.2 Documentation on exports (queues) ................. 681
q_empty/1 (pred) ........................................... 681
q_insert/3 (pred) .......................................... 681
q_member/2 (pred) .......................................... 681
q_delete/3 (pred) .......................................... 681

163 Random numbers ................................. 683
163.1 Usage and interface (random) ......................... 683
163.2 Documentation on exports (random) ................. 683
random/1 (pred) ........................................... 683
random/3 (pred) ........................................... 683
srandom/1 (pred) ......................................... 684
164 Set Operations ........................................ 685
   164.1 Usage and interface (sets) ....................... 685
   164.2 Documentation on exports (sets) ............... 685
      insert/3 (pred) .................................. 685
      ord_delete/3 (pred) ............................. 685
      ord_member/2 (pred) ............................. 685
      ord_test_member/3 (pred) ....................... 686
      ord_subtract/3 (pred) ........................... 686
      ord_intersection/3 (pred) ...................... 686
      ord_intersection_diff/4 (pred) ................. 686
      ord_intersect/2 (pred) .......................... 686
      ord_subset/2 (pred) ............................. 687
      ord_subset_diff/3 (pred) ....................... 687
      ord_union/3 (pred) .............................. 687
      ord_union_diff/4 (pred) ....................... 687
      ord_union_symdiff/4 (pred) ..................... 688
      merge/3 (pred) .................................. 688
      ord_disjoint/2 (pred) ........................... 688
      setproduct/3 (pred) .............................. 688

165 Variable name dictionaries ....................... 689
   165.1 Usage and interface (vndict) ................. 689
   165.2 Documentation on exports (vndict) ............ 689
      null_dict/1 (regtype) ......................... 689
      create_dict/2 (pred) ........................... 689
      complete_dict/3 (pred) ....................... 689
      complete_vars_dict/3 (pred) ................... 690
      prune_dict/3 (pred) ............................ 690
      sort_dict/2 (pred) .............................. 690
      dict2varnamesl/2 (pred) ....................... 690
      varnames2dict/2 (pred) ........................ 690
      find_name/4 (pred) ............................. 691
      rename/2 (pred) ................................ 691
      varname/1 (regtype) ............................ 691
      varnamesl/1 (regtype) ......................... 691
      varnamedict/1 (regtype) ........................ 691
      vars_names_dict/3 (pred) ..................... 691

PART X - Miscellaneous standalone utilities ...... 693

166 A Program to Help Cleaning your Directories .... 695
   166.1 Usage (cleandirs) ......................... 695
   166.2 Known bugs and planned improvements (cleandirs) 695

167 Printing the declarations and code in a file .... 697
   167.1 Usage (fileinfo) .......................... 697
   167.2 More detailed explanation of options (fileinfo) 697

168 Printing the contents of a bytecode file ....... 699
   168.1 Usage (viewpo) ............................ 699
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>169</td>
<td>Crossed-references of a program</td>
<td>701</td>
</tr>
<tr>
<td>170</td>
<td>callgraph (library)</td>
<td>703</td>
</tr>
<tr>
<td>170.1</td>
<td>Usage and interface (callgraph)</td>
<td>703</td>
</tr>
<tr>
<td>170.2</td>
<td>Documentation on exports (callgraph)</td>
<td>703</td>
</tr>
<tr>
<td></td>
<td>call_graph/2 (pred)</td>
<td>703</td>
</tr>
<tr>
<td></td>
<td>reachability/4 (pred)</td>
<td>703</td>
</tr>
<tr>
<td>171</td>
<td>Gathering the dependent files for a file</td>
<td>705</td>
</tr>
<tr>
<td>171.1</td>
<td>Usage (get_deps)</td>
<td>705</td>
</tr>
<tr>
<td>172</td>
<td>Finding differences between two Prolog files</td>
<td>707</td>
</tr>
<tr>
<td>172.1</td>
<td>Usage (pldiff)</td>
<td>707</td>
</tr>
<tr>
<td>172.2</td>
<td>Known bugs and planned improvements (pldiff)</td>
<td>707</td>
</tr>
<tr>
<td>173</td>
<td>The Ciao lpmake scripting facility</td>
<td>709</td>
</tr>
<tr>
<td>173.1</td>
<td>General operation</td>
<td>709</td>
</tr>
<tr>
<td>173.2</td>
<td>Format of the Configuration File</td>
<td>710</td>
</tr>
<tr>
<td>173.3</td>
<td>lpmake usage</td>
<td>710</td>
</tr>
<tr>
<td>173.4</td>
<td>Acknowledgments (lpmake)</td>
<td>710</td>
</tr>
<tr>
<td>174</td>
<td>Find out which architecture we are running on</td>
<td>711</td>
</tr>
<tr>
<td>174.1</td>
<td>Usage (ciao_get_arch)</td>
<td>711</td>
</tr>
<tr>
<td>174.2</td>
<td>More details (ciao_get_arch)</td>
<td>711</td>
</tr>
<tr>
<td>175</td>
<td>Print out WAM code</td>
<td>713</td>
</tr>
<tr>
<td>175.1</td>
<td>Usage (compiler_output)</td>
<td>713</td>
</tr>
<tr>
<td>176</td>
<td>Compile Ciao in an arbitrary subdirectory</td>
<td>715</td>
</tr>
<tr>
<td>176.1</td>
<td>Usage (auto_compile_ciao)</td>
<td>715</td>
</tr>
<tr>
<td>176.2</td>
<td>More details (auto_compile_ciao)</td>
<td>715</td>
</tr>
<tr>
<td>177</td>
<td>Gathering all modules in a directory</td>
<td>717</td>
</tr>
<tr>
<td>177.1</td>
<td>Usage (collect_modules)</td>
<td>717</td>
</tr>
<tr>
<td>177.2</td>
<td>More details (collect_modules)</td>
<td>717</td>
</tr>
<tr>
<td><strong>PART XI</strong> - Contributed libraries</td>
<td></td>
<td>719</td>
</tr>
<tr>
<td>178</td>
<td>Programming MYCIN rules</td>
<td>721</td>
</tr>
<tr>
<td>178.1</td>
<td>Usage and interface (mycin)</td>
<td>721</td>
</tr>
<tr>
<td>178.2</td>
<td>Documentation on new declarations (mycin)</td>
<td>721</td>
</tr>
<tr>
<td></td>
<td>export/1 (decl)</td>
<td>721</td>
</tr>
<tr>
<td>178.3</td>
<td>Known bugs and planned improvements (mycin)</td>
<td>721</td>
</tr>
</tbody>
</table>
A Chart Library .......................... 723

179.1 Bar charts ................................. 723
179.2 Line graphs ............................... 725
179.3 Scatter graphs ......................... 725
179.4 Tables .................................. 726
179.5 Overview of widgets .................... 727
179.6 Usage and interface (chartlib) ......... 728
179.7 Documentation on exports (chartlib) .... 728

barchart1/7 (udreexp) ....................... 728
barchart1/9 (udreexp) ....................... 728
percentbarchart1/7 (udreexp) ............... 728
barchart2/7 (udreexp) ....................... 728
barchart2/11 (udreexp) ..................... 728
percentbarchart2/7 (udreexp) ............... 728
barchart3/7 (udreexp) ....................... 728
barchart3/9 (udreexp) ....................... 729
percentbarchart3/7 (udreexp) ............... 729
barchart4/7 (udreexp) ....................... 729
barchart4/11 (udreexp) ..................... 729
percentbarchart4/7 (udreexp) ............... 729
multibarchart/8 (udreexp) .................. 729
multibarchart/10 (udreexp) ................. 729
tablewidget1/4 (udreexp) ................... 729
tablewidget1/5 (udreexp) ................... 729
tablewidget2/4 (udreexp) ................... 729
tablewidget2/5 (udreexp) ................... 729
tablewidget3/4 (udreexp) ................... 730
tablewidget3/5 (udreexp) ................... 730
tablewidget4/4 (udreexp) ................... 730
tablewidget4/5 (udreexp) ................... 730
graph_b1/9 (udreexp) ....................... 730
graph_b1/13 (udreexp) ...................... 730
graph_w1/9 (udreexp) ....................... 730
graph_w1/13 (udreexp) ...................... 730
scattergraph_b1/8 (udreexp) ............... 730
scattergraph_b1/12 (udreexp) .............. 730
scattergraph_w1/8 (udreexp) ............... 730
scattergraph_w1/12 (udreexp) .............. 730
graph_b2/9 (udreexp) ....................... 731
graph_b2/13 (udreexp) ...................... 731
graph_w2/9 (udreexp) ....................... 731
graph_w2/13 (udreexp) ...................... 731
scattergraph_b2/8 (udreexp) ............... 731
scattergraph_b2/12 (udreexp) .............. 731
scattergraph_w2/8 (udreexp) ............... 731
scattergraph_w2/12 (udreexp) .............. 731
chartlib_text_error_protect/1 (udreexp) ... 731
chartlib_visual_error_protect/1 (udreexp) .. 731
Constraint programming over finite domains

180.1 Usage and interface (fd) ........................................ 734
180.2 Documentation on exports (fd) ................................. 734
   fd_item/1 (regtype) .............................................. 734
   fd_range/1 (regtype) ............................................ 734
   fd_subrange/1 (regtype) .......................................... 734
   fd_store/1 (regtype) ............................................. 735
   fd_store_entity/1 (regtype) ...................................... 735
   labeling/1 (pred) ................................................ 735
   pitm/2 (pred) ..................................................... 735
   choose_var/3 (pred) .............................................. 735
   choose_free_var/2 (pred) ......................................... 736
   choose_var_and/2 (pred) .......................................... 736
   retrieve_range/2 (pred) ........................................... 736
   retrieve_store/2 (pred) .......................................... 736
   glb/2 (pred) ...................................................... 737
   lub/2 (pred) ...................................................... 737
   bounds/3 (pred) .................................................. 737
   retrieve_list_of_values/2 (pred) ................................. 738

XDR handle library .............................................. 739
181.1 Usage and interface (xdr_handle) ............................. 739
181.2 Documentation on exports (xdr_handle) ........................ 739
   xdr_tree/3 (pred) ................................................. 739
   xdr_tree/1 (pred) ................................................. 739
   xdr_node/1 (regtype) ............................................ 740
   xdr2html/4 (pred) ................................................. 740
   xdr2html/2 (pred) ................................................. 740
   unfold_tree/2 (pred) ............................................ 740
   unfold_tree_dic/3 (pred) ......................................... 741
   xdr_xpath/2 (pred) ................................................ 741

XML query library .............................................. 743
182.1 Usage and interface (xml_path) ................................. 744
182.2 Documentation on exports (xml_path) ........................... 744
   xml_search/3 (pred) .............................................. 744
   xml_parse/3 (pred) ................................................. 744
   xml_parse_match/3 (pred) ....................................... 745
   xml_search_match/3 (pred) ...................................... 745
   xml_index_query/3 (pred) ....................................... 745
   xml_index_to_file/2 (pred) ..................................... 746
   xml_index/1 (pred) ............................................... 746
   xml_query/3 (pred) .............................................. 746
182.3 Documentation on internals (xml_path) .......................... 747
   canonic_xml_term/1 (regtype) .................................. 747
   canonic_xml_item/1 (regtype) .................................. 747
   tag_attrib/1 (regtype) .......................................... 747
   canonic_xml_query/1 (regtype) .................................. 747
   canonic_xml_subquery/1 (regtype) .............................. 747
183 Low level Interface between Prolog and blt

- Usage and interface (bltclass) ........................................ 749
- Documentation on exports (bltclass) ................................ 749
  new_interp/1 (pred) .................................................. 749
  tcltk_raw_code/2 (pred) ........................................... 749
  bltwish_interp/1 (regtype) ........................................ 750
  interp_file/2 (pred) .................................................. 750

184 Error Handler for Chartlib ................................. 751

- Usage and interface (chartlib_errhandle) .................. 751
- Documentation on exports (chartlib_errhandle) .......... 751
  chartlib_text_error_protect/1 (pred) ......................... 751
  chartlib_visual_error_protect/1 (pred) ...................... 751
- Documentation on internals (chartlib_errhandle) ....... 752
  handler_type/1 (regtype) ........................................ 752
  error_message/2 (pred) ........................................... 752
  error_file/2 (pred) .................................................. 752

185 Color and Pattern Library ................................. 753

- Usage and interface (color_pattern) ....................... 753
- Documentation on exports (color_pattern) ............... 753
  color/1 (regtype) ................................................ 753
  color/2 (pred) .................................................... 754
  pattern/1 (regtype) ............................................ 755
  pattern/2 (pred) ................................................... 755
  random_color/1 (pred) ............................................ 755
  random_lightcolor/1 (pred) .................................... 755
  random_darkcolor/1 (pred) .................................... 756
  random_pattern/1 (pred) ......................................... 756

186 Barchart widgets - 1 ................................. 757

- Usage and interface (genbar1) ................................. 757
- Documentation on exports (genbar1) ......................... 757
  barchart1/7 (pred) ............................................... 757
  barchart1/9 (pred) ............................................... 758
  percentbarchart1/7 (pred) .................................... 759
  yelement/1 (regtype) ............................................ 760
  axis_limit/1 (regtype) .......................................... 761
  header/1 (regtype) ............................................... 761
  title/1 (regtype) ................................................... 761
  footer/1 (regtype) ................................................. 761
- Documentation on internals (genbar1) ..................... 761
  xbarelement1/1 (regtype) ...................................... 761

187 Barchart widgets - 2 ................................. 763

- Usage and interface (genbar2) ................................. 763
- Documentation on exports (genbar2) ......................... 763
  barchart2/7 (pred) ............................................... 763
  barchart2/11 (pred) ............................................... 764
  percentbarchart2/7 (pred) .................................... 765
  xbarelement2/1 (regtype) ...................................... 765
188  Depict barchart widgets - 3 767

188.1 Usage and interface (genbar3) 767
188.2 Documentation on exports (genbar3) 767
   barchart3/7 (pred) 767
   barchart3/9 (pred) 768
   percentbarchart3/7 (pred) 768

188.3 Documentation on internals (genbar3) 769
   xbarelement3/1 (regtype) 769

189  Depict barchart widgets - 4 771

189.1 Usage and interface (genbar4) 771
189.2 Documentation on exports (genbar4) 771
   barchart4/7 (pred) 771
   barchart4/11 (pred) 772
   percentbarchart4/7 (pred) 772

189.3 Documentation on internals (genbar4) 773
   xbarelement4/1 (regtype) 773

190  Depic line graph 775

190.1 Usage and interface (gengraph1) 776
190.2 Documentation on exports (gengraph1) 776
   graph_b1/9 (pred) 776
   graph_b1/13 (pred) 777
   graph_w1/9 (pred) 777
   graph_w1/13 (pred) 778
   scattergraph_b1/8 (pred) 779
   scattergraph_b1/12 (pred) 780
   scattergraph_w1/8 (pred) 780
   scattergraph_w1/12 (pred) 781
   vector/1 (regtype) 782
   smooth/1 (regtype) 782
   attributes/1 (regtype) 782
   symbol/1 (regtype) 783
   size/1 (regtype) 783

191  Line graph widgets 785

191.1 Usage and interface (gengraph2) 785
191.2 Documentation on exports (gengraph2) 786
   graph_b2/9 (pred) 786
   graph_b2/13 (pred) 786
   graph_w2/9 (pred) 787
   graph_w2/13 (pred) 788
   scattergraph_b2/8 (pred) 789
   scattergraph_b2/12 (pred) 789
   scattergraph_w2/8 (pred) 790
   scattergraph_w2/12 (pred) 791
198 ProVRML - a Prolog interface for VRML . . . 811
198.1 Usage and interface (provrml) ..................... 811
   vrml_web_to_terms/2 (pred) .......................... 811
   vrml_file_to_terms/2 (pred) .......................... 812
   vrml_web_to_terms_file/2 (pred) .................... 812
   vrml_file_to_terms_file/2 (pred) .................... 812
   terms_file_to_vrml/2 (pred) .......................... 812
   terms_file_to_vrml_file/2 (pred) .................... 813
   terms_to_vrml_file/2 (pred) .......................... 813
   terms_to_vrml/2 (pred) ............................... 813
   vrml_to_terms/2 (pred) ............................... 814
   vrml_in_out/2 (pred) ................................. 814
   vrml_http_access/2 (pred) ........................... 814
198.2 Documentation on exports (provrml) ................. 811
   vrml_web_to_terms/2 (pred) .......................... 811
   vrml_file_to_terms/2 (pred) .......................... 812
   vrml_web_to_terms_file/2 (pred) .................... 812
   vrml_file_to_terms_file/2 (pred) .................... 812
   terms_file_to_vrml/2 (pred) .......................... 812
   terms_file_to_vrml_file/2 (pred) .................... 813
   terms_to_vrml_file/2 (pred) .......................... 813
   terms_to_vrml/2 (pred) ............................... 813
   vrml_to_terms/2 (pred) ............................... 814
   vrml_in_out/2 (pred) ................................. 814
   vrml_http_access/2 (pred) ........................... 814
198.3 Documentation on internals (provrml) .................. 814
   read_page/2 (pred) .................................... 814

199 boundary (library) ................................. 817
199.1 Usage and interface (boundary) ...................... 817
199.2 Documentation on exports (boundary) ............... 817
   boundary_check/3 (pred) .............................. 817
   boundary_rotation_first/2 (pred) .................... 817
   boundary_rotation_last/2 (pred) ..................... 818
   reserved_words/1 (pred) ............................... 818
   children_nodes/1 (pred) ............................... 818

200 dictionary (library) ............................... 819
200.1 Usage and interface (dictionary) .................... 819
200.2 Documentation on exports (dictionary) ............. 819
   dictionary/6 (pred) .................................... 819

201 dictionary_tree (library) ......................... 821
201.1 Usage and interface (dictionary_tree) .............. 821
201.2 Documentation on exports (dictionary_tree) ....... 821
   create_dictionaries/1 (pred) ......................... 821
   is_dictionaries/1 (pred) ............................. 821
   get_definition_dictionary/2 (pred) .................. 821
   get_prototype_dictionary/2 (pred) ................... 822
   dictionary_lookup/5 (pred) ........................... 822
   dictionary_insert/5 (pred) ........................... 822
   merge_tree/2 (pred) .................................... 823

202 error (library) .................................... 825
202.1 Usage and interface (error) .......................... 825
202.2 Documentation on exports (error) ................... 825
   error_vrml/1 (pred) ................................... 825
   output_error/1 (pred) .................................. 825

203 field_type (library) .............................. 827
203.1 Usage and interface (field_type) .................... 827
203.2 Documentation on exports (field_type) .............. 827
   fieldType/1 (pred) .................................... 827
204 field_value (library) ........................................... 829
  204.1 Usage and interface (field_value) ......................... 829
  204.2 Documentation on exports (field_value) .................. 829
       fieldValue/6 (pred) .................................... 829
       mfstringValue/5 (pred) ................................ 829
       parse/1 (prop) .......................................... 829

205 field_value_check (library) ................................. 831
  205.1 Usage and interface (field_value_check) .................. 831
  205.2 Documentation on exports (field_value_check) .......... 831
       fieldValue_check/8 (pred) ............................. 831
       mfstringValue/7 (pred) ................................ 831

206 generator (library) ........................................... 833
  206.1 Usage and interface (generator) ........................... 833
  206.2 Documentation on exports (generator) ................. 833
       generator/2 (pred) ...................................... 833
       nodeDeclaration/4 (pred) ............................... 833

207 generator_util (library) ................................. 835
  207.1 Usage and interface (generator_util) ..................... 835
  207.2 Documentation on exports (generator_util) ............ 835
       reading/4 (pred) ........................................ 835
       reading/5 (pred) ........................................ 836
       reading/6 (pred) ........................................ 836
       open_node/6 (pred) ....................................... 836
       close_node/5 (pred) ..................................... 836
       close_nodeGut/4 (pred) ................................. 836
       open_PROTO/4 (pred) .................................... 836
       close_PROTO/6 (pred) ................................... 836
       open_EXTERNPROTO/5 (pred) ............................ 836
       close_EXTERNPROTO/6 (pred) ............................ 836
       open_DEF/5 (pred) ....................................... 836
       close_DEF/5 (pred) ....................................... 837
       open_Script/5 (pred) ..................................... 837
       close_Script/5 (pred) ................................... 837
       decompose_field/3 (pred) ............................... 837
       indentation_list/2 (pred) ................................ 837
       start_vrmlScene/4 (pred) ................................ 837
       remove_comments/4 (pred) ................................ 837

208 internal_types (library) ................................. 839
  208.1 Usage and interface (internal_types) ..................... 839
  208.2 Documentation on exports (internal_types) ............ 839
       bound/1 (regtype) ....................................... 839
       bound_double/1 (regtype) ............................... 839
       dictionary/1 (regtype) ................................ 839
       environment/1 (regtype) ................................ 840
       parse/1 (regtype) ........................................ 840
       tree/1 (regtype) .......................................... 840
       whitespace/1 (regtype) ................................ 840
213  possible (library) .......................... 869
    213.1  Usage and interface (possible) ...................... 869
    213.2  Documentation on exports (possible) .......... 869

214  tokeniser (library) ......................... 871
    214.1  Usage and interface (tokeniser) ................... 871
    214.2  Documentation on exports (tokeniser) .......... 871
          tokeniser/2 (pred) ......................... 871
          token_read/3 (pred) .................... 871
215 Double linked list .......................... 873
  215.1 Usage and interface (ddlset) ................. 873
  215.2 Documentation on exports (ddlset) ............. 873
  null_list/1 (pred) .................................. 873
  next/2 (pred) ........................................ 873
  prev/2 (pred) ........................................ 873
  insert/3 (pred) ...................................... 874
  insert_top/3 (pred) .................................. 874
  insert_after/3 (pred) ................................ 874
  delete/2 (pred) ...................................... 874
  delete_top/2 (pred) .................................. 874
  delete_after/2 (pred) ................................ 874
  top/2 (pred) ......................................... 874
  forward/2 (pred) ..................................... 874
  length/2 (pred) ...................................... 875
  length_next/2 (pred) ................................ 875
  length_prev/2 (pred) ................................ 875
  ddlset/1 (regtype) .................................. 875
  215.3 Other information (ddlset) ...................... 875
    215.3.1 Using insert_after .......................... 875
    215.3.2 More Complex example ...................... 876

216 Measuring features from predicates (time cost
or memory used...) ..................... 877
  216.1 Usage and interface (time_analyzer) ................. 877
  216.2 Documentation on exports (time_analyzer) ............ 877
    performance/3 (pred) ................................ 877
    benchmark/6 (pred) .................................. 878
    compare_benchmark/7 (pred) ......................... 878
    benchmark2/6 (pred) ................................ 879
    compare_benchmark2/7 (pred) ....................... 880
    sub_times/3 (pred) .................................. 880
    div_times/2 (pred) .................................. 880
    cost/3 (pred) ........................................ 880
    generate_plot/3 (udreexp) .......................... 881
    generate_plot/2 (udreexp) .......................... 881
    set_general_options/1 (udreexp) .................... 881
    get_general_options/1 (udreexp) .................... 881

217 Printing graph using gnuplot as auxiliary tool.  .... 883
  217.1 Usage and interface (gnuplot) ..................... 883
  217.2 Documentation on exports (gnuplot) .................. 883
    get_general_options/1 (pred) ....................... 883
    set_general_options/1 (pred) ....................... 883
    generate_plot/2 (pred) ............................. 884
    generate_plot/3 (pred) ............................. 884
Regular Type Definition Index .................. 925
Declaration Definition Index...................... 927
Concept Definition Index......................... 929
Global Index ...................................... 931
Summary

**Ciao** is a *public domain, next generation* multi-paradigm programming environment with a unique set of features:

- **Ciao** offers a complete Prolog system, supporting **ISO-Prolog**, but its novel modular design allows both **restricting** and **extending** the language. As a result, it allows working with **fully declarative subsets** of Prolog and also to **extend** these subsets (or ISO-Prolog) both syntactically and semantically. Most importantly, these restrictions and extensions can be activated separately on each program module so that several extensions can coexist in the same application for different modules.

- **Ciao** also supports (through such extensions) programming with functions, higher-order (with predicate abstractions), constraints, and objects, as well as feature terms (records), persistence, several control rules (breadth-first search, iterative deepening, ...), concurrency (threads-engines), a good base for distributed execution (agents), and parallel execution. Libraries also support WWW programming, sockets, external interfaces (C, Java, TclTk, relational databases, etc.), etc.

- **Ciao** offers support for *programming in the large* with a robust module/object system, module-based separate/incremental compilation (automatically — no need for makefiles), an assertion language for declaring (**optional**) program properties (including types and modes, but also determinacy, non-failure, cost, etc.), automatic static inference and static/dynamic checking of such assertions, etc.

- **Ciao** also offers support for *programming in the small* producing small executables (including only those builtins used by the program) and support for writing scripts in Prolog.

- The **Ciao** programming environment includes a classical top-level and a rich emacs interface with an embeddable source-level debugger and a number of execution visualization tools.

- The **Ciao** compiler (which can be run outside the top level shell) generates several forms of architecture-independent and stand-alone executables, which run with speed, efficiency and executable size which are very competitive with other commercial and academic Prolog/CLP systems. Library modules can be compiled into compact bytecode or C source files, and linked statically, dynamically, or autoloaded.

- The novel modular design of Ciao enables, in addition to modular program development, effective global program analysis and static debugging and optimization via source to source program transformation. These tasks are performed by the **Ciao preprocessor** (ciaopp, distributed separately).

- The **Ciao** programming environment also includes **lpdoc**, an automatic documentation generator for LP/CLP programs. It processes Prolog files adorned with (**Ciao**) assertions and machine-readable comments and generates manuals in many formats including **postscript, pdf, texinfo, info, HTML, man**, etc., as well as on-line help, ascii README files, entries for indices of manuals (**info, WWW, ...**), and maintains WWW distribution sites.

**Ciao** is distributed under the GNU General Public License.

This documentation corresponds to version 1.11#222 (2004/5/24, 13:8:7 CEST).
1 Introduction

1.1 About this manual

This is the Reference Manual for the Ciao Prolog development system. It contains basic information on how to install Ciao Prolog and how to write, debug, and run Ciao Prolog programs from the command line, from inside GNU emacs, or from a windowing desktop. It also documents all the libraries available in the standard distribution.

This manual has been generated using the LPdoc semi-automatic documentation generator for LP/CLP programs [HC97,Her00]. lpdoc processes Prolog files (and files in other CLP languages) adorned with assertions and machine-readable comments, which should be written in the Ciao assertion language [PBH97,PBH00]. From these, it generates manuals in many formats including postscript, pdf, texinfo, info, HTML, man, etc., as well as on-line help, ascii README files, entries for indices of manuals (info, WWW, ...), and maintains WWW distribution sites.

The big advantage of this approach is that it is easier to keep the on-line and printed documentation in sync with the source code [Knu84]. As a result, this manual changes continually as the source code is modified. Because of this, the manual has a version number. You should make sure the manual you are reading, whether it be printed or on-line, coincides with the version of the software that you are using.

The approach also implies that there is often a variability in the degree to which different libraries or system components are documented. Many libraries offer abundant documentation, but a few will offer little. The latter is due to the fact that we tend to include libraries in the manual if the code is found to be useful, even if they may still contain sparse documentation. This is because including a library in the manual will at the bare minimum provide formal information (such as the names of exported predicates and their arity, which other modules it loads, etc.), create index entries, pointers for on-line help in the electronic versions of the manuals, and command-line completion capabilities inside emacs. Again, the manual is being updated continuously as the different libraries (and machine-readable documentation in them) are improved.

1.2 About the Ciao Prolog development system

The Ciao system is a full programming environment for developing programs in the Prolog language and in several other languages which are extensions and modifications of Prolog in several interesting and useful directions. The programming environment offers a number of tools such as the Ciao standalone compiler (ciaoc), a traditional-style top-level interactive shell (ciaosh or ciao), an interpreter of scripts written in Prolog (ciao-shell), a Prolog emacs mode (which greatly helps the task of developing programs with support for editing, debugging, version/change tracking, etc.), numerous libraries, a powerful program preprocessor (ciaopp [BdlBH99,BLGPH04,HBPLG99], which supports static debugging and optimization from program analysis via source to source program transformation), and an automatic documentation generator (lpdoc) [HC97,Her00]. A number of execution visualization tools [CGH93,CH00d,CH00c] are also available.

This manual documents the first four of the tools mentioned above [see PART I - The program development environment], and the Ciao Prolog language and libraries. The ciaopp and lpdoc tools are documented in separate manuals.

The Ciao language [see PART II - The Ciao basic language (engine)] has been designed from the ground up to be small, but to also allow extensions and restrictions in a modular way. The first objective allows producing small executables (including only those builtins used by the program), providing basic support for pure logic programming, and being able to write
scripts in Prolog. The second one allows supporting standard ISO-Prolog [see PART III - ISO-Prolog library (iso)], as well as powerful extensions such as constraint logic programming, functional logic programming, and object-oriented logic programming [see PART VII - Ciao Prolog extensions], and restrictions such as working with pure horn clauses.

The design of Ciao has also focused on allowing modular program development, as well as automatic program manipulation and optimization. Ciao includes a robust module system [CH00a], module-based automatic incremental compilation [CH99], and modular global program analysis, debugging and optimization [PH99], based on a rich assertion language [see PART V - Annotated Prolog library (assertions)] for declaring (optional) program properties (including types and modes), which can be checked either statically or dynamically. The program analysis, static debugging and optimization tasks related to these assertions are performed by the ciaoopp preprocessor, as mentioned above. These assertions (together with special comment-style declarations) are also the ones used by the 1pdoc autodocumenter to generate documentation for programs (the comment-style declarations are documented in the 1pdoc manual).

Ciao also includes several other features and utilities, such as support for several forms of executables, concurrency (threads), distributed and parallel execution, higher-order, WWW programming [PiLLoW [CHV96b]], interfaces to other languages like C and Java, database interfaces, graphical interfaces, etc., etc. [see PARTS VI to XI].

1.3 ISO-Prolog compliance versus extensibility

One of the innovative features of Ciao is that it has been designed to subsume ISO-Prolog (International Standard ISO/IEC 13211-1, PROLOG: Part 1-General Core [DEDC96]), while at the same time extending it in many important ways. The intention is to ensure that all ISO-compliant Prolog programs run correctly under Ciao. At the same time, the Ciao module system (see [PART II - The Ciao basic language (engine)] and [CH00a] for a discussion of the motivations behind the design) allows selectively avoiding the loading of most ISO-builtsins (and changing some other ISO characteristics) when not needed, so that it is possible to work with purer subsets of Prolog and also to build small executables. Also, this module system makes it possible to develop extensions using these purer subsets (or even the full ISO-standard) as a starting point. Using these features, the Ciao distribution includes libraries which significantly extend the language both syntactically and semantically.

Compliance with ISO is still not complete: currently there are some minor deviations in, e.g., the treatment of characters, the syntax, some of the arithmetic functions, and part of the error system. On the other hand, Ciao has been reported by independent sources (members of the standardization body) to be one of the most conforming Prologs at the moment of this writing, and the first one to be able to compile all the standard-conforming test cases. Also, Ciao does not offer a strictly conforming mode which rejects uses of non-ISO features. However, in order to aid programmers who wish to write standard compliant programs, library predicates that correspond to those in the ISO-Prolog standard are marked specially in the manuals, and differences between the Ciao and the prescribed ISO-Prolog behaviours, if any, are commented appropriately.

The intention of the Ciao developers is to progressively complete the compliance of Ciao with the published parts of the ISO standard as well as with other reasonable extensions of the standard may be published in the future. However, since one of the design objectives of Ciao is to address some shortcomings of previous implementations of Prolog and logic programming in general, we also hope that some of the better ideas present in the system will make it eventually into the standards.

1.4 About the name of the System

After reading the previous sections the sharp reader may have already seen the logic behind the 'Ciao Prolog' name. Ciao is an interesting word which means both hello and goodbye. Ciao
Prolog intends to be a really good, all-round, freely available ISO-Prolog system which can be used as a classical Prolog, in both academic and industrial environments (and, in particular, to introduce users to Prolog and to constraint and logic programming – the *hello* part). But Ciao is also a new-generation, multiparadigm programming language and program development system which goes well beyond Prolog and other classical logic programming languages. And it has the advantage (when compared to other systems) that it does so while keeping full Prolog compatibility when needed.

### 1.5 Referring to Ciao

If you find Ciao or any of its components useful, we would appreciate very much if you added a reference to this manual (i.e., the Ciao reference manual [BCC97]) in your work. The following is an appropriate BiBTeX entry with the relevant data:

```latex
@techreport{ciao-reference-manual-tr,
  author = {F. Bueno and D. Cabeza and M. Carro and M. Hermenegildo and P. López-García and G. Puebla},
  title = {The Ciao Prolog system. Reference manual},
  institution = {School of Computer Science, Technical University of Madrid (UPM)},
  year = 1997,
  month = {August},
  number = {{CLIP}3/97.1},
  note = {Available from http://www.clip.dia.fi.upm.es/}
}
```

### 1.6 Syntax terminology and notational conventions

This manual is not meant to be an introduction to the Prolog language. The reader is referred to standard textbooks on Prolog such as [SS86,CM81,Apt97,Hog84]. However, we would like to refresh herein some concepts for the sake of establishing terminology. Also, we will briefly introduce a few of the extensions that Ciao brings to the Prolog language.

#### 1.6.1 Predicates and their components

In Prolog, procedures are called *predicates* and predicate calls *literals*. They all have the classical syntax of procedures (and of logic predications and of mathematical functions). Predicates are identified in this manual by a keyword 'PREDICATE' at the right margin of the place where they are documented.

Prolog instructions are expressions made up of control constructs (Chapter 13 [Control constructs/predicates], page 73) and literals, and are called *goals*. Literals are also (atomic) goals.

A predicate definition is a sequence of clauses. A clause has the form ‘*H* :- *B*.’ (ending in ‘.’), where *H* is syntactically the same as a literal and is called the clause *head*, and *B* is a goal and is called the clause *body*. A clause with no body is written ‘*H*.’ and is called a *fact*. Clauses with body are also called *rules*. A Prolog program is a sequence of predicate definitions.

#### 1.6.2 Characters and character strings

We adopt the following convention for delineating character strings in the text of this manual: when a string is being used as a Prolog atom it is written thus: *user* or ‘*user*’; but in all other circumstances double quotes are used (as in "*hello*.")
When referring to keyboard characters, printing characters are written thus: \( \text{\texttt{a}} \), while control characters are written like this: \( \text{\texttt{\textasciicircum A}} \). Thus \( \text{\texttt{\textasciicircum C}} \) is the character you get by holding down the \( \text{\texttt{CTL}} \) key while you type \( \text{\texttt{c}} \). Finally, the special control characters carriage-return, line-feed and space are often abbreviated to \( \text{\texttt{RET}} \), \( \text{\texttt{LFD}} \) and \( \text{\texttt{SPC}} \) respectively.

### 1.6.3 Predicate specs

Predicates in Prolog are distinguished by their name and their arity. We will call name/arity a *predicate spec*. The notation name/arity is therefore used when it is necessary to refer to a predicate unambiguously. For example, \texttt{concatenate/3} specifies the predicate which is named “concatenate” and which takes 3 arguments.

(Note that different predicates may have the same name and different arity. Conversely, of course, they may have the same arity and different name.)

### 1.6.4 Modes

When documenting a predicate, we will often describe its usage with a mode spec which has the form \texttt{name(Arg1, ..., ArgN)} where each Arg may be preceded by a *mode*. A mode is a functor which is wrapped around an argument (or prepended if defined as an operator). Such a mode allows documenting in a compact way the instantiation state on call and exit of the argument to which it is applied. The set of modes which can be used in Ciao is not fixed. Instead, arbitrary modes can be defined by in programs using the \texttt{modedef/1} declarations of the Ciao assertion language (Chapter 53 [The Ciao assertion package], page 269 for details). Modes are identified in this manual by a keyword ‘MODE’.

Herein, we will use the set of modes defined in the Ciao isomodes library, which is essentially the same as those used in the ISO-Prolog standard (Chapter 57 [ISO-Prolog modes], page 295).

### 1.6.5 Properties and types

Although Ciao Prolog is not a typed language, it allows writing (and using) types, as well as (more general) properties. There may be properties of the states and of the computation. Properties of the states allow expressing characteristics of the program variables during computation, like in \texttt{sorted(X)} (X is a sorted list). Properties of the computation allow expressing characteristics of a whole computation, like in \texttt{is_det(p(X,Y))} (such calls yield only one solution). Properties are just a special form of predicates (Chapter 55 [Declaring regular types], page 283) and are identified in this manual by a keyword ‘PROPERTY’.

Ciao types are regular types (Chapter 55 [Declaring regular types], page 283), which are a special form of properties themselves. They are identified in this manual by a keyword ‘REG-TYPE’.

### 1.6.6 Declarations

A *declaration* provides information to one of the Ciao environment tools. Declarations are interspersed in the code of a program. Usually the target tool is either the compiler (telling it that a predicate is dynamic, or a meta-predicate, etc.), the preprocessor (which understands declarations of properties and types, assertions, etc.), or the autodocumenter (which understands the previous declarations and also certain “comment” declarations).

A declaration has the form \( :- D \), where \( D \) is syntactically the same as a literal. Declarations are identified in this manual by a keyword ‘DECLARATION’.

In Ciao users can define (and document) new declarations. New declarations are typically useful when defining extensions to the language (which in Ciao are called packages). Such extensions are often implemented as expansions (see Chapter 26 [Extending the syntax], page 143).
There are many such extensions in Ciao. The **functions** library, which provides functional syntax, is an example. The fact that in Ciao expansions are local to modules (as operators, see below) makes it possible to use a certain language extension in one module without affecting other modules.

### 1.6.7 Operators

An **operator** is a functor (or predicate name) which has been declared as such, thus allowing its use in a prefix, infix, or suffix fashion, instead of the standard procedure-like fashion. E.g., declaring `+` as an infix operator allows writing `X+Y` instead of `'+'(X,Y)` (which may still, of course, be written).

*Operators in Ciao are local to the module/file where they are declared.* However, some operators are standard and allowed in every program (see Chapter 36 [Defining operators], page 197). This manual documents the operator declarations in each (library) module where they are included. As with expansions, the fact that in Ciao operators are local to modules makes it possible to use a certain language extension in one module without affecting other modules.

### 1.7 A tour of the manual

The rest of the introductory chapters after this one provide a first “getting started” introduction for newcomers to the Ciao system. The rest of the chapters in the manual are organized into a sequence of major parts as follows:

#### 1.7.1 PART I - The program development environment

This part documents the components of the basic Ciao program development environment. They include:

- **ciaoc**: the standalone compiler, which creates executables without having to enter the interactive top-level.
- **ciaosh**: (also invoked simply as **ciao**) is an interactive top-level shell, similar to the one found on most Prolog systems (with some enhancements).
- **debugger.pl**: a Byrd box-type debugger, similar to the one found on most Prolog systems (also with some enhancements, such as source-level debugging). This is not a standalone application, but is rather included in **ciaosh**, as is done in other Prolog systems. However, it is also embeddable, in the sense that it can be included as a library in executables, and activated dynamically and conditionally while such executables are running.
- **ciao-shell**: an interpreter/compiler for **Prolog scripts** (i.e., files containing Prolog code which run without needing explicit compilation).
- **ciao.el**: **a complete program development environment**, based on GNU emacs, with syntax coloring, direct access to all the tools described above (as well as the preprocessor and the documenter), atomic location of errors, source-level debugging, context-sensitive access to on-line help/manuals, etc. The use of this environment is very highly recommended!

The Ciao program development environment also includes **ciaopp**, the preprocessor, and **lpdoc**, the documentation generator, which are described in separate manuals.
1.7.2 PART II - The Ciao basic language (engine)

This part documents the Ciao basic builtins. These predefined predicates and declarations are available in every program, unless the pure package is used (by using a :- module(_,_,[pure]). declaration or :- use_package(pure).). These predicates are contained in the engine directory within the lib library. The rest of the library predicates, including the packages that provide most of the ISO-Prolog builtins, are documented in subsequent parts.

1.7.3 PART III - ISO-Prolog library (iso)

This part documents the iso package which provides to Ciao programs (most of) the ISO-Prolog functionality, including the ISO-Prolog builtins not covered by the basic library.

1.7.4 PART IV - Classic Prolog library (classic)

This part documents some Ciao libraries which provide additional predicates and functionalities that, despite not being in the ISO standard, are present in many popular Prolog systems. This includes definite clause grammars (DCGs), “Quintus-style” internal database, list processing predicates, DEC-10 Prolog-style input/output, formatted output, dynamic loading of modules, activation of operators at run-time, etc.

1.7.5 PART V - Annotated Prolog library (assertions)

Ciao allows annotating the program code with assertions. Such assertions include type and instantiation mode declarations, but also more general properties as well as comments in the style of the literate programming. These assertions document predicates (and modules and whole applications) and can be used by the Ciao preprocessor/compiler while debugging and optimizing the program or library, and by the Ciao documenter to build the program or library reference manual.

1.7.6 PART VI - Ciao Prolog library miscellanea

This part documents several Ciao libraries which provide different useful additional functionalities. Such functionalities include performing operating system calls, gathering statistics from the Prolog engine, file and file name manipulation, error and exception handling, fast reading and writing of terms (marshalling and unmarshalling), file locking, program reporting messages, pretty-printing programs and assertions, a browser of the system libraries, additional expansion utilities, concurrent aggregates, graph visualization, etc.

1.7.7 PART VII - Ciao Prolog extensions

The libraries documented in this part extend the Ciao language in several different ways. The extensions include:

- pure Prolog programming (well, this can be viewed more as a restriction than an extension);
- feature terms or records (i.e., structures with names for each field);
- parallel programming (e.g., &-Prolog style);
- functional syntax;
- higher-order library;
- global variables;
- setarg and undo;
- delaying predicate execution;
- active modules;
• breadth-first execution;
• iterative deepening-based execution;
• constraint logic programming;
• object oriented programming.

1.7.8 PART VIII - Interfaces to other languages and systems

The following interfaces to/from Ciao Prolog are documented in this part:
• External interface (e.g., to C).
• Socket interface.
• Tcl/tk interface.
• Web interface (http, html, xml, etc.);
• Persistent predicate databases (interface between the Prolog internal database and the external file system).
• SQL-like database interface (interface between the Prolog internal database and external SQL/ODBC systems).
• Java interface.
• Calling emacs from Prolog.

1.7.9 PART IX - Abstract data types

This part includes libraries which implement some generic data structures (abstract data types) that are used frequently in programs or in the Ciao system itself.

1.7.10 PART X - Miscellaneous standalone utilities

This is the documentation for a set of miscellaneous standalone utilities contained in the etc directory of the Ciao distribution.

1.7.11 PART XI - Contributed libraries

This part includes a number of libraries which have contributed by users of the Ciao system. Over time, some of these libraries are moved to the main library directories of the system.

1.7.12 PART XII - Appendices

These appendices describe the installation of the Ciao environment on different systems and some other issues such as reporting bugs, signing up on the Ciao user's mailing list, downloading new versions, limitations, etc.

1.8 Acknowledgments

The Ciao system is a joint effort on one side of the present (Francisco Bueno, Daniel Cabeza, Manuel Carro, Manuel Hermenegildo, Pedro López, and Germán Puebla) and past (María José García de la Banda) members of the CLIP group at the School of Computer Science, Technical University of Madrid, and on the other side of several colleagues and students that have collaborated with us over the years of its development. The following is an (inevitably incomplete) list of those that have contributed to the development of Ciao:
• The Ciao engine, compiler, libraries and documentation, although completely rewritten at this point, have their origins in the &-Prolog parallel Prolog engine and parallelizing compiler, developed by Manuel Hermenegildo, Kevin Greene, Kalyan Muthukumar, and Roger Nasr at MCC and later at UPM. The &-Prolog engine and low-level (WAM) compilers in turn were derived from early versions (0.5 to 0.7) of SICStus Prolog [Car88]. SICStus is an excellent, high performance Prolog system, developed by Mats Carlsson and colleagues at the Swedish Institute of Computer Science (SICS), that every user of Prolog should check out [Swe95,AAF91]. Very special thanks are due to Seif Haridi, Mats Carlsson, and colleagues at SICS for allowing the SICStus 0.5-0.7 components in &-Prolog and its successor, Ciao, to be distributed freely. Parts of the parallel abstract machine have been developed in collaboration with Gopal Gupta and Enrico Pontelli (New Mexico State University).

• Many aspects of the analyzers in the Ciao preprocessor (ciaopp) have been developed in collaboration with Peter Stuckey (Melbourne U.), Kim Marriott (Monash U.), Maurice Bruynooghe, Gerda Janssens, Anne Mulkers, and Veroniek Dumortier (K.U. Leuven), and Saumya Debray (U. of Arizona). The assertion system has been developed in collaboration with Jan Maluszynski and Wlodek Drabent (Linkoping U.) and Pierre Deransart (INRIA). The core of type inference system derives from the system developed by John Gallagher [GdW94] (Bristol University) and later adapted to CLP(FD) by Pawel Pietrzak (Linkoping U.).

• The constraint solvers for R and Q are derived from the code developed by Christian Holzbauer (Austrian Research Institute for AI in Vienna) [Hol94,Hol90,Hol92].

• The Ciao manuals include material from the DECsystem-10 Prolog User’s Manual by D.L. Bowen (editor), L. Byrd, F.C.N. Pereira, L.M. Pereira, and D.H.D. Warren [BBP81]. They also contain material from the SICStus Prolog user manuals for SICStus versions 0.5-0.7 by Mats Carlsson and Johan Widen [Car88], as well as from the Prolog ISO standard documentation [DEDC96].

• Ciao is designed to be highly extendable in a modular way. Many of the libraries distributed with Ciao have been developed by other people all of which is impossible to mention here. Individual author names are included in the documentation of each library and appear in the indices.

• The development of the Ciao system has been supported in part by European research projects ACCLAIM, PARFORCE, DISCIPL, AMOS, and ASAP and by MICYT projects ELLA, EDIPIA, and CUBICO.

If you feel you have contributed to the development of Ciao and we have forgotten adding your name to this list or the acknowledgements given in the different chapters, please let us know and we will be glad to give proper credits.

1.9 Version/Change Log (ciao)

Version 1.9 (2002/5/16, 23:17:34 CEST)
New development version after stable 1.8p0 (MCL, DCG)

• Improvements related to supported platforms:
  • Support for Mac OS X 10.1, based on the Darwin kernel.
  • Initial support for compilation on Linux for Power PC (contributed by Paulo Moura).
  • Workaround for incorrect C compilation while using newer (> 2.95) gcc compilers.
  • .bat files generated in Windows.
• Changes in compiler behavior and user interface:
• Corrected a bug which caused wrong code generation in some cases.
• Changed execution of initialization directives. Now the initialization of a module/file never runs before the initializations of the modules from which the module/file imports (excluding circular dependencies).
• The engine is more intelligent when looking for an engine to execute bytecode; this caters for a variety of situations when setting explicitly the CIAOLIB environment variable.
• Fixed bugs in the toplevel: behaviour of module:main calls and initialization of a module (now happens after related modules are loaded).
• Layout char not needed any more to end Prolog files.
• Syntax errors now disable .itf creation, so that they show next time the code is used without change.
• Redefinition warnings now issued only when an unqualified call is seen.
• Context menu in Windows can now load a file into the toplevel.
• Updated Windows installation in order to run CGI executables under Windows: a new information item is added to the registry.
• Added new directories found in recent Linux distributions to INFOPATH.
• Emacs-based environment and debugger improved:
  • Errors located immediately after code loading.
  • Improved ciao-check-types-modes (preprocessor progress now visible).
  • Fixed loading regions repeatedly (no more predicate redefinition warnings).
  • Added entries in ciaopp menu to set verbosity of output.
  • Fixed some additional xemacs compatibility issues (related to searches).
  • Errors reported by inferior processes are now explored in forward order (i.e., the first error reported is the first one highlighted). Improved tracking of errors.
  • Specific tool bar now available, with icons for main functions (works from emacs 21.1 on). Also, other minor adaptations for working with emacs 21.1 and later.
  • Debugger faces are now locally defined (and better customization). This also improves compatibility with xemacs (which has different faces).
  • Direct access to a common use of the preprocessor (checking modes/types and locating errors) from toolbar.
  • Inferior modes for Ciao and CiaoPP improved: contextual help turned on by default.
  • Fixes to set-query. Also, previous query now appears in prompt.
  • Improved behaviour of stored query.
  • Improved behaviour of recentering, finding errors, etc.
  • Wait for prompt has better termination characteristics.
  • Added new interactive entry points (M-x): ciao, prolog, ciaopp.
  • Better tracking of last inferior buffer used.
  • Miscellaneous bugs removed; some colors changed to adapt to different Emacs versions.
  • Fixed some remaining incompatibilities with xemacs.
  • :- doc now also supported and highlighted.
• Eliminated need for calendar.el
• Added some missing library directives to fontlock list, organized this better.

• New libraries added to the system:
  • hiord: new library which needs to be loaded in order to use higher-order call/N and P(X) syntax. Improved model for predicate abstractions.
  • fuzzy: allows representing fuzzy information in the form or Prolog rules.
  • use_url: allows loading a module remotely by using a WWW address of the module source code.
  • andorra: alternative search method where goals which become deterministic at run time are executed before others.
  • iterative deepening (id): alternative search method which makes a depth-first search until a predetermined depth is reached. Complete but in general cheaper than breadth first.
  • det_hook: allows making actions when a deterministic situation is reached.
  • ProVRML: read VRML code and translate it into Prolog terms, and the other way around.
  • io_alias_redirection: change where stdin/stdout/stderr point to from within Ciao Prolog programs.
  • tcl_tk: an interface to Tcl/Tk programs.
  • tcl_tk_obj: object-based interface to Tcl/Tk graphical objects.
  • CiaoPP: options to interface with the CiaoPP Prolog preprocessor.

• Some libraries greatly improved:
  • WebDB: utilities to create WWW-based database interfaces.
  • Improved java interface implementation (this forced renaming some interface primitives).
  • User-transparent persistent predicate database revamped:
    • Implemented asserta_fact/1 (assert_fact/1).
    • Now it is never necessary to explicitly call init_persdb, a call to initialize_db is only needed after dynamically defining facts of persistent_dir/2. Thus, pcurrent_fact/1 predicate eliminated.
    • Facts of persistent predicates included in the program code are now included in the persistent database when it is created. They are ignored in successive executions.
    • Files where persistent predicates reside are now created inside a directory named as the module where the persistent predicates are defined, and are named as F_A* for predicate F/A.
    • Now there are two packages: persdb and 'persdb/ll' (for low level). In the first, the standard builtins asserta_fact/1, assertz_fact/1, and retract_fact/1 are replaced by new versions which handle persistent data predicates, behaving as usual for normal data predicates. In the second package, predicates with names starting with 'p' are defined, so that there is not overhead in calling the standard builtins.
    • Needed declarations for persistent_dir/2 are now included in the packages.
  • SQL now works with mysql.
  • system: expanded to contain more predicates which act as interface to the underlying system / operating system.
• Other libraries improved:
  • xref: creates cross-references among Prolog files.
  • concurrency: new predicates to create new concurrent predicates on-the-fly.
  • sockets: bugs corrected.
  • objects: concurrent facts now properly recognized.
  • fast read/write: bugs corrected.
  • Added ‘webbased’ protocol for active modules: publication of active module address can now be made through WWW.
  • Predicates in library(dynmods) moved to library(compiler).
  • Expansion and meta predicates improved.
  • Pretty printing.
  • Assertion processing.
  • Module-qualified function calls expansion improved.
  • Module expansion calls goal expansion even at runtime.
• Updates to builtins (there are a few more; these are the most relevant):
  • Added a prolog_flag to retrieve the version and patch.
  • current_predicate/1 in library(dynamic) now enumerates non-engine modules, prolog_sys:current_predicate/2 no longer exists.
  • exec/* bug fixed.
  • srandom/1 bug fixed.
• Updates for C interface:
  • Fixed bugs in already existing code.
  • Added support for creation and traversing of Prolog data structures from C predicates.
  • Added support for raising Prolog exceptions from C predicates.
  • Preliminary support for calling Prolog from C.
• Miscellaneous updates:
  • Installation made more robust.
  • Some pending documentation added.
  • ‘ciao’ script now adds (locally) to path the place where it has been installed, so that other programs can be located without being explicitly in the $PATH.
  • Loading programs is somewhat faster now.
  • Some improvement in printing path names in Windows.

Version 1.7 (2000/7/12, 19:1:20 CEST)
Development version following even 1.6 distribution.

Version 1.6 (2000/7/12, 18:55:50 CEST)
• Source-level debugger in emacs, breakpts.
• Emacs environment improved, added menus for Ciaopp and LPDoc.
• Debugger embeddable in executables.
• Stand-alone executables available for UNIX-like operating systems.
• Many improvements to emacs interface.
• Menu-based interface to autodocumenter.
• Threads now available in Win32.
• Many improvements to threads.
• Modular clp(R) / clp(Q).
• Libraries implementing And-fair breadth-first and iterative deepening included.
• Improved syntax for predicate abstractions.
• Library of higher-order list predicates.
• Better code expansion facilities (macros).
• New delay predicates (when/2).
• Compressed object code/executables on demand.
• The size of atoms is now unbound.
• Fast creation of new unique atoms.
• Number of clauses/predicates essentially unbound.
• Delayed goals with freeze restored.
• Faster compilation and startup.
• Much faster fast write/read.
• Improved documentation.
• Other new libraries.
• Improved installation/deinstallation on all platforms.
• Many improvements to autodocumenter.
• Many bug fixes in libraries and engine.

Development version following even 1.4 distribution.

Version 1.4 (1999/11/27, 19:0:0 MEST)
• Documentation greatly improved.
• Automatic (re)compilation of foreign files.
• Concurrency primitives revamped; restored &Prolog-like multiengine capability.
• Windows installation and overall operation greatly improved.
• New version of O’Ciao class/object library, with improved performance.
• Added support for "predicate abstractions" in call/N.
• Implemented reexportation through reexport declarations.
• Changed precedence of importations, last one is now higher.
• Modules can now implicitly export all predicates.
• Many minor bugs fixed.

Version 1.3 (1999/6/16, 17:5:58 MEST)
Development version following even 1.2 distribution.

Version 1.2 (1999/6/14, 16:54:55 MEST)
Temporary version distributed locally for extensive testing of reexportation and other 1.3 features.

Development version following even 1.0 distribution.

Version 1.0 (1999/6/4, 13:27:42 MEST)
• Added Tcl/Tk interface library to distribution.
• Added push_prolog_flag/2 and pop_prolog_flag/1 declarations/builtins.
• Filename processing in Windows improved.
• Added redefining/1 declaration to avoid redefining warnings.
- Changed syntax/1 declaration to use package/1.
- Added add_clause_trans/1 declaration.
- Changed format of .itf files such that a '+' stands for all the standard imports from engine, which are included in c_itf source internally (from engine(builtin_exports)). Further changes in itf data handling, so that once an .itf file is read in a session, the file is cached and next time it is needed no access to the file system is required.
- Many bugs fixed.

**Version 0.9 (1999/3/10, 17:3:49 CET)**
- Test version before 1.0 release. Many bugs fixed.

**Version 0.8 (1998/10/27, 13:12:36 MET)**
- Changed compiler so that only one pass is done, eliminated .dep files.
- New concurrency primitives.
- Changed assertion comment operator to #.
- Implemented higher-order with call/N.
- Integrated SQL-interface to external databases with persistent predicate concept.
- First implementation of object oriented programming package.
- Some bugs fixed.

**Version 0.7 (1998/9/15, 12:12:33 MEST)**
- Improved debugger capabilities and made easier to use.
- Simplified assertion format.
- New arithmetic functions added, which complete all ISO functions.
- Some bugs fixed.

**Version 0.6 (1998/7/16, 21:12:7 MET DST)**
- Defining other path aliases (in addition to 'library’) which can be loaded dynamically in executables is now possible.
- Added the possibility to define multifile predicates in the shell.
- Added the possibility to define dynamic predicates dynamically.
- Added addmodule meta-argument type.
- Implemented persistent data predicates.
- New version of PiLLoW WWW library (XML, templates, etc.).
- Ported active modules from “distributed Ciao” (independent development version of Ciao).
- Implemented lazy loading in executables.
- Modularized engine(builtin).
- Some bugs fixed.

**Version 0.5 (1998/3/23)**
- First Windows version.
- Integrated debugger in toplevel.
- Implemented DCG’s as (Ciao-style) expansions.
- Builtins renamed to match ISO-Prolog.
- Made ISO the default syntax/package.

**Version 0.4 (1998/2/24)**
- First version with the new Ciao emacs mode.
• Full integration of concurrent engine and compiler/library.
• Added new declaration/1 directive.
• Added modular syntax enhancements.
• Shell script interpreter separated from toplevel shell.
• Added new compilation warnings.

**Version 0.3 (1997/8/20)**
• Ciao builtins modularized.
• New prolog flags can be defined by libraries.
• Standalone command-line compiler available, with automatic "make".
• Added assertions and regular types.
• First version using the automatic documentation generator.

**Version 0.2 (1997/4/16)**
• First module system implemented.
• Implemented exceptions using catch/3 and throw/1.
• Added functional & record syntax.
• Added modular sentence, term, and goal translations.
• Implemented attributed variables.
• First CLPQ/CLPR implementation.
• Added the posibility of linking external .so files.
• Changes in syntax to allow P(X) and "string" || L.
• Changed to be more similar to ISO-Prolog.
• Implemented Prolog shell scripts.
• Implemented data predicates.

**Version 0.1 (1997/2/13)**
First fully integrated, standalone Ciao distribution. Based on integrating into an evolution of the &-Prolog engine/libraries/preprocessor [Her86,HG91] many functionalities from several previous independent development versions of Ciao [HC93,HC94,HCC95,Bue95,CLI95,HBdlBP95,HBC96,CHV96b,HBC99].
2 Getting started on Un*x-like machines

Author(s): M.Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#133 (2001/11/1, 16:34:6 CET)

This part guides you through some very basic first steps with Ciao on a Un*x-like system. It
assumes that Ciao is already installed correctly on your Un*x system. If this is not the case, then
follow the instructions in Chapter 220 [Installing Ciao from the source distribution], page 897
first.

We start with by describing the basics of using Ciao from a normal command shell such
as sh/bash, csh/tcsh, etc. We strongly recommend reading also Section 2.4 [An introduction
to the Ciao emacs environment (Un*x)], page 20 for the basics on using Ciao under emacs,
which is a much simpler and much more powerful way of developing Ciao programs, and has
the advantage of offering an almost identical environment under Un*x and Windows.

2.1 Testing your Ciao Un*x installation

It is a good idea to start by performing some tests to check that Ciao is installed correctly on
your system (these are the same tests that you are instructed to do during installation, so you
can obviously skip them if you have done them already at that time). If any of these tests do
not succeed either your environment variables are not set properly (see Section 2.2 [Un*x user
setup], page 17 for how to fix this):

- Typing ciao (or ciaosh) should start the typical Prolog top-level shell.
- In the top-level shell, Prolog library modules should load correctly. Type for example
  use_module(library(dec10_io)) —you should get back a prompt with no errors reported.
- To exit the top level shell, type halt. as usual, or (^D).
- Typing ciaoc should produce the help message from the Ciao standalone compiler.
- Typing ciao-shell should produce a message saying that no code was found. This is a
  Ciao application which can be used to write scripts written in Prolog, i.e., files which do
  not need any explicit compilation to be run.

Also, the following documentation-related actions should work:

- If the info program is installed, typing info should produce a list of manuals which should
  include Ciao manual(s) in a separate area (you may need to log out and back in so that
  your shell variables are reinitialized for this to work).
- Opening with a WWW browser (e.g., netscape) the directory or URL corresponding to the
  DOCROOT setting should show a series of Ciao-related manuals. Note that style sheets should
  be activated for correct formatting of the manual.
- Typing man ciao should produce a man page with some very basic general information on
  Ciao (and pointing to the on-line manuals).
- The DOCROOT directory should contain the manual also in the other formats such as
  postscript or pdf which specially useful for printing. See Section 2.3.7 [Printing man-
  uals (Un*x)], page 20 for instructions.

2.2 Un*x user setup

If the tests above have succeeded, the system is probably installed correctly and your envi-
ronment variables have been set already. In that case you can skip to the next section.

Otherwise, if you have not already done so, make the following modifications in your startup
scripts, so that these files are used (<LIBROOT> must be replaced with the appropriate value,
i.e., where the Ciao library is installed):
• For users a csh-compatible shell (csh, tcsh, ...), add to ~/.cshrc:

```bash
if ( -e <LIBROOT>/ciao/DOTcshrc ) then
  source <LIBROOT>/ciao/DOTcshrc
endif
```

Mac OS X users should add (or modify) the path file in the directory ~/Library/init/tcsh, adding the lines shown above. Note: while this is recognized by the terminal shell, and therefore by the text-mode Emacs which comes with Mac OS X, the Aqua native Emacs 21 does not recognize that initialization. It is thus necessary, at this moment, to set manually the Ciao shell (ciaosh) and Ciao library location by hand. This can be done from the Ciao menu within Emacs after a Ciao Prolog file has been loaded. We suppose that the reason is that Mac OS X does not actually consult the per-user initialization files on startup. It should also be possible to put the right initializations in the .emacs file using the setenv function of Emacs-lisp, as in

```emacs
(setenv "CIAOLIB" "<LIBROOT>/ciao")
```

The same can be done for the rest of the variables initialized in <LIBROOT>/ciao/DOTcshrc

• For users of an sh-compatible shell (sh, bash, ...), add to ~/.profile:

```bash
if [ -f <LIBROOT>/ciao/DOTprofile ]; then
  . <LIBROOT>/ciao/DOTprofile
fi
```

This will set up things so that the Ciao executables are found and you can access the Ciao system manuals using the info command. Note that, depending on your shell, you may have to log out and back in for the changes to take effect.

• Also, if you use emacs (highly recommended) add this line to your ~/.emacs file:

```
(load-file "<LIBROOT>/ciao/DOTemacs.el")
```

If after following these steps things do not work properly, then the installation was probably not completed properly and you may want to try reinstalling the system.

### 2.3 Using Ciao from a Un*x command shell

#### 2.3.1 Starting/exiting the top-level shell (Un*x)

The basic methods for starting/exiting the top-level shell have been discussed above. If upon typing ciao you get a “command not found” error or you get a longer message from Ciao before starting, it means that either Ciao was not installed correctly or you environment variables are not set up properly. Follow the instructions on the message printed by Ciao or refer to the installation instructions regarding user-setup for details.

#### 2.3.2 Getting help (Un*x)

The basic methods for accessing the manual on-line have also been discussed above. Use the table of contents and the indices of predicates, libraries, concepts, etc. to find what you are looking for. Context-sensitive help is available within the emacs environment (see below).

#### 2.3.3 Compiling and running programs (Un*x)

Once the shell is started, you can compile and execute Prolog modules inside the interactive top-level shell in the standard way. E.g., type `use_module(file), use_module(library(file)).` for library modules, `ensure_loaded(file).` for files which are not modules, and `use_package(file).` for library packages (these are syntactic/semantic packages that extend the Ciao
Prolog language in many different ways). Note that the use of `compile/1` and `consult/1` is discouraged in Ciao.

For example, you may want to type `use_package(iso)` to ensure Ciao has loaded all the ISO builtins (whether this is done by default or not depends on your `.ciaorc` file). Do not worry about any “module already in executable” messages – these are normal and simply mean that a certain module is already pre-loaded in the top-level shell. At this point, typing `write(hello)` should work.

Note that some predicates that may be built-ins in other Prologs are available through libraries in Ciao. This facilitates making small executables.

To change the working directory to, say, the `examples` directory in the Ciao root directory, first do:

```prolog
?- use_module(library(system)).
```

(loading the `system` library makes a number of system-related predicates such as `cd/1` accessible) and then:

```prolog
?- cd('$/examples').
```

(in Ciao the sequence `$/ at the beginning of a path name` is replaced by the path of the Ciao root directory).

For more information see Chapter 5 [The interactive top-level shell], page 37.

### 2.3.4 Generating executables (Un*x)

Executables can be generated from the top-level shell (using `make_exec/2`) or using the standalone compiler (`ciaoc`). To be able to make an executable, the file should define the predicate `main/1` (or `main/0`), which will be called upon startup (see the corresponding manual section for details). In its simplest use, given a top-level `foo.pl` file for an application, the compilation process produces an executable `foo`, automatically detecting which other files used by `foo.pl` need recompilation.

For example, within the `examples` directory, you can type:

```prolog
?- make_exec(hw, _).
```

which should produce an executable. Typing `hw` in a shell (or double-clicking on the icon from a graphical window) should execute it.

For more information see Chapter 5 [The interactive top-level shell], page 37 and Chapter 4 [The stand-alone command-line compiler], page 29.

### 2.3.5 Running Ciao scripts (Un*x)

Ciao allows writing Prolog scripts. These are files containing Prolog source but which get executed without having to explicitly compile them (in the same way as, e.g., `.bat` files or programs in scripting languages). As an example, you can run the file `hw` in the `examples` directory of the Ciao distribution and look at the source with an editor. You can try changing the `Hello world` message and running the program again (no need to recompile!).

As you can see, the file should define the predicate `main/1` (not `main/0`), which will be called upon startup. The two header lines are necessary in Un*x in. In Windows you can leave them in or you can take them out, but you need to rename the script to `hw.pls`. Leaving the lines in has the advantage that the script will also work in Un*x without any change.

For more information see Chapter 8 [The script interpreter], page 59.
2.3.6 The Ciao initialization file (Un*x)

The Ciao toplevel can be made to execute upon startup a number of commands (such as, e.g., loading certain files or setting certain Prolog flags) contained in an initialization file. This file should be called .ciaorc and placed in your home directory (e.g., ~, the same in which the .emacs file is put). You may need to set the environment variable HOME to the path of this directory for the Ciao toplevel shell to be able to locate this file on startup.

2.3.7 Printing manuals (Un*x)

As mentioned before, the manual is available in several formats in the reference directory within the doc directory in the Ciao distribution, including postscript or pdf, which are especially useful for printing. These files are also available in the DOCROOT directory specified during installation. Printing can be done using an application such as ghostview (freely available from http://www.cs.wisc.edu/~ghost/index.html) or acrobat reader (http://www.adobe.com, only pdf).

2.4 An introduction to the Ciao emacs environment (Un*x)

While it is easy to use Ciao with any editor of your choice, using it within the emacs editor/program development system is highly recommended: Ciao includes an emacs mode which provides a very complete application development environment which greatly simplifies many program development tasks. See (undefined) [Using Ciao inside GNU emacs], page (undefined) for details on the capabilities of ciao/emacs combination.

If the (freely available) emacs editor/environment is not installed in your system, we highly recommend that you also install it at this point (there are instructions for where to find emacs and how to install it in the Ciao installation instructions). After having done this you can try for example the following things:

- A few basic things:
  - Typing \((\texttt{M-\textasciitilde})\)% (or in the menus Help->Manuals->Browse Manuals with Info) should open a list of manuals in info format in which the Ciao manual(s) should appear.
  - When opening a Prolog file, i.e., a file with .pl or .pls ending, using \((\texttt{X}\texttt{F})\)filename (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and Ciao/Prolog menus should appear in the menu bar on top of the emacs window.
  - Loading the file using the Ciao/Prolog menu (or typing \((\texttt{C})\)% should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the toplevel shell and make queries from within emacs.

Note: when using emacs it is very convenient to swap the locations of the (normally not very useful) \((\texttt{Caps} \texttt{Lock})\) key and the (very useful in emacs) \((\texttt{Ctrl})\) key on the keyboard. How to do this is explained in the emacs frequently asked questions FAQs (see the emacs download instructions for their location).

(if these things do not work the system or emacs may not be installed properly).

- You can go to the location of most of the errors that may be reported during compilation by typing \((\texttt{C})\)%.
- You can also, e.g., create executables from the Ciao/Prolog menu, as well as compile individual files, or generate active modules.
- Loading a file for source-level debugging using the Ciao/Prolog menu (or typing \((\texttt{C})\)% and then issuing a query should start the source-level debugger and move a marker on the code in a window while execution is stepped through in the window running the Ciao toplevel.
You can add the lines needed in Un*x for turning any file defining main/1 into a script from the Ciao/Prolog menu or by typing `C-h h`. You can also work with the preprocessor and auto-documenter directly from emacs: see their manuals or browse through the corresponding menus that appear when editing .pl files.

We encourage you once more to read (undefined) [Using Ciao inside GNU emacs], page (undefined) to discover the many other functionalities of this environment.

### 2.5 Keeping up to date (Un*x)

You may want to read Chapter 222 [Beyond installation], page 911 for instructions on how to sign up on the Ciao user’s mailing list, receive announcements regarding new versions, download new versions, report bugs, etc.
3 Getting started on Windows machines

Author(s): M.Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#157 (2004/1/9, 15:21:39 CET)

This part guides you through some very basic first steps with Ciao on an MSWindows (“Win32”) system. It assumes that Ciao is already installed correctly on your Windows system. If this is not the case, then follow the instructions in Chapter 221 [Installing Ciao from a Win32 binary distribution], page 907 (or Chapter 220 [Installing Ciao from the source distribution], page 897) first.

We start with by describing the basics of using Ciao from the Windows explorer and/or a DOS command shell. We strongly recommend reading also Section 3.3 [An introduction to the Ciao emacs environment (Win32)], page 25 for the basics on using Ciao under emacs, which is a much simpler and much more powerful way of developing Ciao programs, and has the advantage of offering an almost identical environment under Windows and Un*x.

3.1 Testing your Ciao Win32 installation

It is a good idea to start by performing some tests to check that Ciao is installed correctly on your system (these are the same tests that you are instructed to do during installation, so you can obviously skip them if you have done them already at that time):

- Ciao-related file types (.pl source files, .cpx executables, .itf, .po, .asr interface files, .pls scripts, etc.) should have specific icons associated with them (you can look at the files in the folders in the Ciao distribution to check).
- Double-clicking on the shortcut to ciaosh(.cpx) on the desktop should start the typical Prolog top-level shell in a window. If this shortcut has not been created on the desktop, then double-clicking on the ciaosh(.cpx) icon inside the shell folder within the Ciao source folder should have the same effect.
- In the top-level shell, Prolog library modules should load correctly. Type for example use_module(library(dec10_io)). at the Ciao top-level prompt—you should get back a prompt with no errors reported.
- To exit the top level shell, type halt. as usual, or Ctrl-D.

Also, the following documentation-related actions should work:

- Double-clicking on the shortcut to ciao(.html) which appears on the desktop should show the Ciao manual in your default WWW browser. If this shortcut has not been created you can double-click on the ciao(.html) file in the doc\reference\ciao_html folder inside the Ciao source folder. Make sure you configure your browser to use style sheets for correct formatting of the manual (note, however, that some older versions of Explorer did not support style sheets well and will give better results turning them off).
- The doc\reference folder contains the manual also in the other formats present in the distribution, such as info (very convenient for users of the emacs editor/programming system) and postscript or pdf, which are specially useful for printing. See Section 3.2.7 [Printing manuals (Win32)], page 25 for instructions.

3.2 Using Ciao from the Windows explorer and command shell

3.2.1 Starting/exiting the top-level shell (Win32)

The basic methods for starting/exiting the top-level shell have been discussed above. The installation script also leaves a ciaosh(.bat) file inside the shell folder of the Ciao distribution which can be used to start the top-level shell from the command line in Windows systems.
3.2.2 Getting help (Win32)

The basic methods for accessing the manual on-line have also been discussed above. Use the table of contents and the indices of predicates, libraries, concepts, etc. to find what you are looking for. Context-sensitive help is available within the emacs environment (see below).

3.2.3 Compiling and running programs (Win32)

Once the shell is started, you can compile and execute Prolog modules inside the interactive toplevel shell in the standard way. E.g., type `use_module(file), use_module(library(file))`. for library modules, `ensure_loaded(file)`. for files which are not modules, and `use_package(file)` for library packages (these are syntactic/semantic packages that extend the Ciao Prolog language in many different ways). Note that the use of `compile/1` and `consult/1` is discouraged in Ciao.

For example, you may want to type `use_package(iso)` to ensure Ciao has loaded all the ISO builtins (whether this is done by default or not depends on your `.ciaorc` file). Do not worry about any “module already in executable” messages – these are normal and simply mean that a certain module is already pre-loaded in the toplevel shell. At this point, typing `write(hello).` should work.

Note that some predicates that may be built-ins in other Prologs are available through libraries in Ciao. This facilitates making small executables.

To change the working directory to, say, the `examples` directory in the Ciao source directory, first do:

```prolog
?- use_module(library(system)).
```

(loading the `system` library makes a number of system-related predicates such as `cd/1` accessible) and then:

```prolog
?- cd('$/examples').
```

(in Ciao the sequence `$/ at the beginning of a path name` is replaced by the path of the Ciao root directory).

For more information see Chapter 5 [The interactive top-level shell], page 37.

3.2.4 Generating executables (Win32)

Executables can be generated from the toplevel shell (using `make_exec/2`) or using the standalone compiler (`ciaoc(.cpx), located in the ciaoc folder). To be able to make an executable, the file should define the predicate `main/1` (or `main/0`), which will be called upon startup (see the corresponding manual section for details).

For example, within the `examples` directory, you can type:

```prolog
?- make_exec(hw,\_).
```

which should produce an executable. Double-clicking on this executable should execute it.

Another way of creating Ciao executables from source files is by right-clicking on `.pl` files and choosing “make executable”. This uses the standalone compiler (this has the disadvantage, however, that it is sometimes difficult to see the error messages).

For more information see Chapter 5 [The interactive top-level shell], page 37 and Chapter 4 [The stand-alone command-line compiler], page 29.
3.2.5 Running Ciao scripts (Win32)

Double-clicking on files ending in .pls, Ciao Prolog scripts, will also execute them. These are files containing Prolog source but which get executed without having to explicitly compile them (in the same way as, e.g., .bat files or programs in scripting languages). As an example, you can double-click on the file hw.pls in the examples folder and look at the source with an editor. You can try changing the Hello world message and double-clicking again (no need to recompile!).

As you can see, the file should define the predicate main/1 (not main/0), which will be called upon startup. The two header lines are only necessary in Un*x. In Windows you can leave them in or you can take them out, but leaving them in has the advantage that the script will also work in Un*x without any change.

For more information see Chapter 8 [The script interpreter], page 59.

3.2.6 The Ciao initialization file (Win32)

The Ciao toplevel can be made to execute upon startup a number of commands (such as, e.g., loading certain files or setting certain Prolog flags) contained in an initialization file. This file should be called .ciaorc and placed in your home folder (e.g., the same in which the .emacs file is put). You may need to set the environment variable HOME to the path of this folder for the Ciao toplevel shell to be able to locate this file on startup.

3.2.7 Printing manuals (Win32)

As mentioned before, the manual is available in several formats in the reference folder within Ciao’s doc folder, including postscript or pdf, which are specially useful for printing. This can be done using an application such as ghostview (freely available from http://www.cs.wisc.edu/~ghost/index.html) or acrobat reader (http://www.adobe.com, only pdf).

3.3 An introduction to the Ciao emacs environment (Win32)

While it is easy to use Ciao with any editor of your choice, using it within the emacs editor/program development system is highly recommended: Ciao includes an emacs mode which provides a very complete application development environment which greatly simplifies many program development tasks. See (undefined) [Using Ciao inside GNU emacs], page (undefined) for details on the capabilities of ciao/emacs combination.

If the (freely available) emacs editor/environment is not installed in your system, we highly recommend that you also install it at this point (there are instructions for where to find emacs and how to install it in the Ciao installation instructions). After having done this you can try for example the following things:

- A few basic things:
  - Typing (Ctrl) 0 (or in the menus Help->Manuals->Browse Manuals with Info) should open a list of manuals in info format in which the Ciao manual(s) should appear.
  - When opening a Prolog file, i.e., a file with .pl or .pls ending, using (X) (F) filename (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and Ciao/Prolog menus should appear in the menu bar on top of the emacs window.
  - Loading the file using the Ciao/Prolog menu (or typing (C) 0) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the the toplevel shell and make queries from within emacs.
Note: when using \texttt{emacs} it is \textit{very convenient} to swap the locations of the (normally not
very useful) \texttt{(Caps Lock)} key and the (very useful in \texttt{emacs}) \texttt{(Ctrl)} key on the keyboard. How to
do this is explained in the \texttt{emacs} frequently asked questions FAQs (see the \texttt{emacs} download
instructions for their location).

(if these things do not work the system or \texttt{emacs} may not be installed properly).

- You can go to the location of most of the errors that may be reported during compilation
  by typing \texttt{(C)} \texttt(\textbackslash (C)} \texttt{C}i\texttt{h}.
- You can also, e.g., create executables from the Ciao/Prolog menu, as well as compile
  individual files, or generate active modules.
- Loading a file for source-level debugging using the Ciao/Prolog menu (or typing \texttt{(C)} \texttt(\textbackslash (C)} \texttt{d})
  and then issuing a query should start the source-level debugger and move a marker on the
  code in a window while execution is stepped through in the window running the Ciao top
  level.
- You can add the lines needed in Un*x for turning any file defining \texttt{main/1}
  into a script from
  the Ciao/Prolog menu or by typing \texttt{(C)} \texttt(\textbackslash (C)} \texttt{l}i\texttt{S}i.
- You can also work with the preprocessor and auto-documenter directly from \texttt{emacs}:
  see
  their manuals or browse through the corresponding menus that appear when editing \texttt{.pl}
  files.

  We encourage you once more to read \texttt{(undefined) [Using Ciao inside GNU emacs]}, page \texttt{(undefined)}
to discover the many other functionalities of this environment.

3.4 Keeping up to date (Win32)

 You may want to read Chapter 222 [Beyond installation], page 911 for instructions on how to
 sign up on the Ciao user’s mailing list, receive announcements regarding new versions, download
 new versions, report bugs, etc.
PART I - The program development environment

Author(s): The CLIP Group.

This part documents the components of the basic Ciao program development environment. They include:

ciaoc: the standalone compiler, which creates executables without having to enter the interactive top-level.

ciaosh: (also invoked simply as ciao) is an interactive top-level shell, similar to the one found on most Prolog systems (with some enhancements).

debugger.pl: a Byrd box-type debugger, similar to the one found on most Prolog systems (also with some enhancements, such as source-level debugging). This is not a standalone application, but is rather included in ciaosh, as is done in other Prolog systems. However, it is also embeddable, in the sense that it can be included as a library in executables, and activated dynamically and conditionally while such executables are running.

ciao-shell: an interpreter/compiler for Prolog scripts (i.e., files containing Prolog code which run without needing explicit compilation).

ciao.el: a complete program development environment, based on GNU emacs, with syntax coloring, direct access to all the tools described above (as well as the preprocessor and the documenter), automatic location of errors, source-level debugging, context-sensitive access to on-line help/manuals, etc. The use of this environment is very highly recommended!

The Ciao program development environment also includes ciaopp, the preprocessor, and lpdoc, the documentation generator, which are described in separate manuals.
4 The stand-alone command-line compiler

Author(s): Daniel Cabeza, The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#98 (2003/8/27, 12:39:15 CEST)

ciaoc [CH00b] is the Ciao stand-alone command-line compiler. ciaoc can be used to create executables or to compile individual files to object code (to be later linked with other files). ciaoc is specially useful when working from the command line. Also, it can be called to compile Ciao programs from other tools such as, e.g., shell scripts, Makefiles, or project files. All the capabilities of ciaoc are also available from the interactive top-level shell, which uses the ciaoc modules as its components.

4.1 Introduction to building executables

An executable can be built from a single file or from a collection of inter-related files. In the case of only one file, this file must define the predicate main/0 or main/1. This predicate is the one which will be called when the executable is started. As an example, consider the following file, called hello.pl:

```prolog
main :-
    write('Hello world'),
    nl.
```

To compile it from the command line using the ciaoc standalone compiler it suffices to type “ciaoc hello” (in Win32 you may have to put the complete path to the ciaoc folder of the Ciao distribution, where the installation process leaves a ciaoc.bat file):

```
/herme@clip:/tmp
[60]> ciaoc hello
```

This produces an executable called hello in Un*x-like systems and hello.cpx under Win32 systems. This executable can then be run in Win32 by double-clicking on it and on Un*x systems by simply typing its name (see for Section 4.3 [Running executables from the command line], page 30 for how to run executables from the command line in Win32):

```
/herme@clip:/tmp
[61]> hello
Hello world
```

If the application is composed of several files the process is identical. Assume hello.pl is now:

```
:- use_module(aux,[p/1]).

main :-
    p(X),
    write(X),
    nl.
```

where the file aux.pl contains:

```
:- module(aux,[p/1]).

p('Hello world').
```

This can again be compiled using the ciaoc standalone compiler as before:
The invocation of `ciaoc hello` compiles the file `hello.pl` and all connected files that may need recompilation – in this case the file `aux.pl`. Also, if any library files used had not been compiled previously they would be compiled at this point (See Section 4.6 [Intermediate files in the compilation process], page 33). Also, if, say, `hello.pl` is changed and recompiled, the object code resulting from the previous compilation of `aux.pl` will be reused. This is all done without any need for `Makefile`s, and considerably accelerates the development process for large applications. This process can be observed by selecting the `-v` option when invoking `ciaoc` (which is equivalent to setting the `verbose_compilation` Prolog flag to `on` in the top-level interpreter).

If `main/1` is defined instead of `main/0` then when the executable is started the argument of `main/1` will be instantiated to a list of atoms, each one of them corresponding to a command line option. Consider the file `say.pl`:

```prolog
main(Argv) :-
    write_list(Argv), nl.
write_list([]).
write_list([Arg|Args]) :-
    write(Arg),
    write(' '),
    write_list(Args).
```

Compiling this program and running it results in the following output:

```
/herme@clip:/tmp
[91]> ciaoc say
/herme@clip:/tmp
[91]> say hello dolly
hello dolly
```

The name of the generated executable can be controlled with the `-o` option (See Section 4.7 [Usage (ciaoc)], page 34).

### 4.2 Paths used by the compiler during compilation

The compiler will look for files mentioned in commands such as `use_module/1` or `ensure_loaded/1` in the current directory. Other paths can be added by including them in a file whose name is given to `ciaoc` using the `-u` option. This file should contain facts of the predicates `file_search_path/2` and `library_directory/1` (see the documentation for these predicates and also Chapter 9 [Customizing library paths and path aliases], page 61 for details).

### 4.3 Running executables from the command line

As mentioned before, what the `ciaoc` compiler generates and how it is started varies somewhat from OS to OS. In general, the product of compiling an application with `ciaoc` is a file that contains the bytecode (the product of the compilation) and invokes the Ciao engine on it.
• Un Un*x this is a script (see the first lines of the file) which invokes the ciao engine on this file. To run the generated executable from a Un*x shell, or from the bash shell that comes with the Cygwin libraries (see Section 220.6 [Installation and compilation under Windows], page 902) it suffices to type its name at the shell command line, as in the examples above.

• In a Win32 system, the compiler produces a similar file with a .cpx ending. The Ciao installation process typically makes sure that the Windows registry contains the right entries so that this executable will run upon double-clicking on it.

In you want to run the executable from the command line an additional .bat file is typically needed. To help in doing this, the Win32 installation process creates a .bat skeleton file called bat_skel in the Win32 folder of the distribution) which allows running Ciao executables from the command line. If you want to run a Ciao executable file.cpx from the command line, you normally copy the skeleton file to the folder were the executable is and rename it to file.bat, then change its contents as explained in a comment inside the file itself.

Note that this .bat file is usually not necessary in NT, as its command shell understands file extension associations. I.e., in windows NT it is possible to run the file.cpx executable directly. Due to limitations of .bat files in Windows 95/98, in those OSs no more than 9 command line arguments can be passed to the executable (in NT there is no such restriction).

Finally, in a system in which Cygnus Win32 is installed executables can also be used directly from the bash shell command line, without any associated .bat files, by simply typing their name at the bash shell command line, in the same way as in Un*x. This only requires that the bash shell which comes with Cygnus Win32 be installed and accessible: simply, make sure that /bin/sh.exe exists.

Except for a couple of header lines, the contents of executables are almost identical under different OSs (except for self-contained ones). The bytecode they contain is architecture-independent. In fact, it is possible to create an executable under Un*x and run it on Windows or viceversa, by making only minor modifications (e.g., creating the .bat file and/or setting environment variables or editing the start of the file to point to the correct engine location).

4.4 Types of executables generated

While the default options used by ciaoc are sufficient for normal use, by selecting other options ciaoc can generate several different types of executables, which offer interesting tradeoffs among size of the generated executable, portability, and startup time [CH00b]:

Dynamic executables:

ciaoc produces by default dynamic executables. In this case the executable produced is a platform-independent file which includes in compiled form all the user-defined files. On the other hand, any system libraries used by the application are loaded dynamically at startup. More precisely, any files that appear as library(...) in use_module/1 and ensure_loaded/1 declarations will not be included explicitly in the executable and will instead be loaded dynamically. It is also possible to mark other path aliases (see the documentation for file_search_path/2) for dynamic loading by using the -d option. Files accessed through such aliases will also be loaded dynamically.

Dynamic loading allows making smaller executables. Such executables may be used directly in the same machine in which they were compiled, since suitable paths to the location of the libraries will be included as default in the executable by ciaoc during compilation.

The executable can also be used in another machine, even if the architecture and OS are different. The requirement is that the Ciao libraries (which will also include the appropriate Ciao engine for that architecture and OS) be installed in the target
machine, and that the CIAOLIB and CIAOENGINE environment variables are set appropriately for the executable to be able to find them (see Section 4.5 [Environment variables used by Ciao executables], page 33). How to do this differs slightly from OS to OS.

Static executables:
Selecting the -s option ciaoc produces a static executable. In this case the executable produced (again a platform-independent file) will include in it all the auxiliary files and any system libraries needed by the application. Thus, such an executable is almost complete, needing in order to run only the Ciao engine, which is platform-specific. Again, if the executable is run in the same machine in which it was compiled then the engine is found automatically. If the executable is moved to another machine, the executable only needs access to a suitable engine (which can be done by setting the CIAOENGINE environment variable to point to this engine).

This type of compilation produces larger executables, but has the advantage that these executables can be installed and run in a different machine, with different architecture and OS, even if Ciao is not installed on that machine. To install (or distribute) such an executable, one only needs to copy the executable file itself and the appropriate engine for the target platform (See Chapter 220 [Installing Ciao from the source distribution], page 897 or Chapter 221 [Installing Ciao from a Win32 binary distribution], page 907 and Section 220.5 [Multiarchitecture support], page 902), and to set things so that the executable can find the engine.

Dynamic executables, with lazy loading:
Selecting the -l option is very similar to the case of dynamic executables above, except that the code in the library modules is not loaded when the program is started but rather it is done during execution, the first time a predicate defined in that file is called. This is advantageous if a large application is composed of many parts but is such that typically only some of the parts are used in each invocation. The Ciao preprocessor, ciaopp, is a good example of this: it has many capabilities but typically only some of them are used in a given session. An executable with lazy load has the advantage that it starts fast, loading a minimal functionality on startup, and then loads the different modules automatically as needed.

Self-contained executables:
Self-contained executables are static executables (i.e., this option also implies static compilation) which include a Ciao engine along with the bytecode, so they do not depend on an external one for their execution. This is useful to create executables which run even if the machine where the program is to be executed does not have a Ciao engine installed and/or libraries. The disadvantage is that such executables are platform-dependent (as well as larger than those that simply use an external library). This type of compilation is selected with the -S option. Cross-compilation is also possible with the -SS option, so you can specify the target OS and architecture (e.g.

---

1. Currently there is an exception to this related to libraries which are written in languages other than Prolog, as, e.g., C. C files are currently always compiled to dynamically loadable object files (.so files), and they thus need to be included manually in a distribution of an application. This will be automated in upcoming versions of the Ciao system.

2. It is also possible to produce real standalone executables, i.e., executables that do not need to have an engine around. However, this is not automated yet, although it is planned for an upcoming version of the compiler. In particular, the compiler can generate a .c file for each .pl file. Then all the .c files can be compiled together into a real executable (the engine is added one more element during link time) producing a complete executable for a given architecture. The downside of course is that such an executable will not be portable to other architectures without recompilation.
LINUXx86). To be able to use the latter option, it is necessary to have installed a ciaoengine for the target machine in the Ciao library (this requires compiling the engine in that OS/architecture and installing it, so that it is available in the library).

Compressed executables:
In compressed executables the bytecode is compressed. This allows producing smaller executables, at the cost of a slightly slower startup time. This is selected with the -z option. You can also produce compressed libraries if you use -z1 along with the -c option. If you select -z1 while generating an executable, any library which is compiled to accomplish this will be also compressed.

Active modules:
The compiler can also compile (via the -a option) a given file into an active module (see Chapter 99 [Active modules (high-level distributed execution)], page 421 for a description of this).

4.5 Environment variables used by Ciao executables

The executables generated by the Ciao compiler (including the ciao development tools themselves) locate automatically where the Ciao engine and libraries have been installed, since those paths are stored as defaults in the engine and compiler at installation time. Thus, there is no need for setting any environment variables in order to run Ciao executables (on a single architecture – see Section 220.5 [Multiarchitecture support], page 902 for running on multiple architectures).

However, the default paths can be overridden by using the environment variables CIAOENGINE and CIAOLIB. The first one will tell the Ciao executables where to look for an engine, and the second will tell them where to look for the libraries. Thus, it is possible to actually use the Ciao system without installing it by setting these variables to the following values:

- CIAOENGINE: $(SRC)/bin/$(CIAOARCH)/ciaoengine
- CIAOLIB: $(SRC)

where $(CIAOARCH) is the string echoed by the command SRC/etc/ciao_get_arch (or BINROOT/ciao_get_arch, after installation).

This allows using alternate engines or libraries, which can be very useful for system development and experimentation.

4.6 Intermediate files in the compilation process

Compiling an individual source (i.e., .pl) file produces a .itf file and a .po file. The .itf file contains information of the modular interface of the file, such as information on exported and imported predicates and on the other modules used by this module. This information is used to know if a given file should be recompiled at a given point in time and also to be able to detect more errors statically including undefined predicates, mismatches on predicate characteristics across modules, etc. The .po file contains the platform-independent object code for a file, ready for linking (statically or dynamically).

It is also possible to use ciaoc to explicitly generate the .po file for one or more .pl files by using the -c option.

4.7 Usage (ciaoc)

The following provides details on the different command line options available when invoking ciaoc:
ciaoc <MiscOpts> <ExecOpts> [-o <execname>] <file> ...

Make an executable from the listed files. If there is more than one file, they must be non-module, and the first one must include the main predicate. The -o option allows generating an arbitrary executable name.

ciaoc <MiscOpts> <ExecOpts> -a <publishmod> <module>

Make an active module executable from <module> with address publish module <publishmod>.

ciaoc <MiscOpts> -c <file> ...

Compile listed files (make .po objects).

<MiscOpts> can be: [-v] [-ri] [-u <file>]

-v verbose mode
-ri generate human readable .itf files
-u use <file> for compilation

<ExecOpts> can be: [-s|-S|-SS <target>|-z|-zl|-e|-l|(-ll <module>)*] (-d <alias>)* [-x]

-s make a static executable (otherwise dynamic files are not included)
-S make standalone executable for the current OS and architecture
-SS make standalone executable for <target> OS and architecture
valid <target> values may be: LINUXi86, SolarisSparc...

(both -S and -SS imply -s)

-z generate executables with compressed bytecode
-zl generate libraries with compressed bytecode - any library (re)compiled as consequence of normal executable compilation will also be affected

-e make executable with eager load of dynamic files at startup (default)
-l idem with lazy load of dynamic files (except insecure cases)

-ll force <module> to be loaded lazily, implies -l

-d files using this path alias are dynamic (default: library)

-x Extended recompilation: only useful for Ciao standard library developers
default extension for files is '.pl'
5 The interactive top-level shell

Author(s): Daniel Cabeza and the CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.7#129 (2001/10/28, 15:38:52 CET)

ciaoosh is the Ciao interactive top-level shell. It provides the user with an interactive programming environment with tools for incrementally building programs, debugging programs by following their executions, and modifying parts of programs without having to start again from scratch. If available, it is strongly recommended to use it with the emacs interface provided, as it greatly simplifies the operation. This chapter documents general operation in the shell itself. Other chapters document the

5.1 Shell invocation and startup

When invoked, the shell responds with a message of identification and the prompt `?-` as soon as it is ready to accept input, thus:


When the shell is initialized it looks for a file `.ciaorc` in the HOME directory and makes an include of it, if it exists. This file is useful for including `use_module/1` declarations for the modules one wants to be loaded by default, changing prolog flags, etc. (Note that the `.ciaorc` file can only contain directives, not actual code; to load some code at startup put it in a separate file and load it using e.g. a `use_module/1` declaration.) If the initialization file does not exist, the default package `default` is included, to provide more or less what other prologs define by default. Thus, if you want to have available all builtins you had before adding the initialization file, you have to include `:- use_package(default)` in it. Two command-line options control the loading of the initialization file:

- `-f` Fast start, do not load any initialization file.
- `-l File` Look for initialization file File instead of `~/ciaorc`. If it does not exist, include the default package.

5.2 Shell interaction

After the shell outputs the prompt, it is expecting either an internal command (see the following sections) or a query (a goal or sequence of goals). When typing in the input, which must be a valid prolog term, if the term does not end in the first line, subsequent lines are indented. For example:

    ?- X =
        f(a,
          b).

    X = f(a,b) ?

    yes

    ?-

The queries are executed by the shell as if they appeared in the user module. Thus, in addition to builtin predicates, predicates available to be executed directly are all predicates defined by loaded user files (files with no module declaration), and imported predicates from modules by the use of `use_module`.

The possible answers of the shell, after executing an internal command or query, are:
If the execution failed (or produced an error), the answer is **no**.

If the execution was successful, and no answer variable (see below) was bound (or constraints where imposed on such variables), the answer is simply **yes**. This behavior can be changed by doing `set_prolog_flag(prompt_alternatives_no_bindings, on)`, so that in any case the user will be consulted as explained in the next point (useful if the solutions produce side effects).

If the execution was successful and bindings where made (or constraints where imposed) on answer variables, then the shell outputs the values of answer variables, as a sequence of bindings (or constraints), and then prints a `?` as a prompt. At this point it is expecting an input line from the user. By entering a carriage-return (RET) or any line starting with `y`, the query terminates and the shell answer **yes**. Entering a `;`, the shell enters a recursive level (see below). Finally, any other answer forces the system to backtrack and look for the next solution (answering as with the first solution).

To allow using connection variables in queries without having to report their results, variables whose name starts with `_` are not considered in answers, the rest being the **answer variables**.

This example illustrates the previous points:

```prolog
?- member(a, [b, c]).
no
?- member(a, [a, b]).
yes
?- member(X, [a|L]).
X = a ? ;
L = [X|_] ?
yes
?- atom_codes(ciao, _C), member(L, _C).
L = 99 ? ;
L = 105 ? ;
L = 97 ? ;
L = 111 ? ;
no
?- 
```

### 5.3 Entering recursive (conjunctive) shell levels

As stated before, when the user answers with `;` after a solution is presented, the shell enters a **recursive level**, changing its prompt to `N ?-` (where `N` is the recursion level) and keeping the bindings or constraints of the solution (this is inspired by the LogIn language developed by H. Ait-Kaci, P. Lincoln and Roger Nasr [AKNL86]). Thus, the following queries will be executed within that context, and all variables in the lower level solutions will be reported in subsequent solutions at this level. To exit a recursive level, input an `EOF` character or the command `up`. The last solution after entering the level is repeated, to allow asking for more solutions. Use command `top` to exit all recursive levels and return to the top level. Example interaction:
?- directory_files('.', _Fs), member(F, _Fs).

F = 'file_utils.po' ,

1 ?- file_property(F, mod_time(T)).

F = 'file_utils.po',
T = 923497679 ?

yes
1 ?- up.

F = 'file_utils.po' ;

F = 'file_utils.pl' ;

F = 'file_utils.itf' ,

1 ?- file_property(F, mod_time(T)).

F = 'file_utils.itf',
T = 923497679 ?

yes
1 ?- ^D

F = 'file_utils.itf' ?

yes

?-
5.4 Usage and interface (ciaosh)

- **Library usage:**
  The following predicates can be used at the top-level shell natively (but see also the commands available in Chapter 6 [The interactive debugger], page 45 which are also available within the top-level shell).

- **Exports:**
  - *Predicates:*
    - use_module/1, use_module/2, ensure_loaded/1, make_exec/2, include/1, use_package/1, consult/1, compile/1, ./2, make_po/1, unload/1, set_debug_mode/1, set_nodebug_mode/1, make_actmod/2, force_lazy/1, undo_force_lazy/1, dynamic_search_path/1, multifile/1.

- **Other modules used:**
  - *Application modules:*
    - library(ciaosh).
  - *System library modules:*
    - libpaths, compiler/compiler, compiler/exemaker, compiler/c_itf, debugger/debugger.

5.5 Documentation on exports (ciaosh)

**use_module/1:**

**PREDICATE**

*Usage:* use_module(Module)

- *Description:* Load into the top-level the module defined in Module, importing all the predicates it exports.

- *The following properties should hold at call time:*
  - Module is a source name. (streams_basic:sourcename/1)

**use_module/2:**

**PREDICATE**

*Usage:* use_module(Module, Imports)

- *Description:* Load into the top-level the module defined in Module, importing the predicates in Imports.

- *The following properties should hold at call time:*
  - Module is a source name. (streams_basic:sourcename/1)
  - Imports is a list of prednames. (basic_props:list/2)

**ensure_loaded/1:**

**PREDICATE**

*Usage:* ensure_loaded(File)

- *Description:* Load into the top-level the code residing in file (or files) File, which is user (i.e. non-module) code.

- *The following properties should hold at call time:*
  - File is a source name or a list of source names. (ciaosh_doc:sourcenames/1)
PREDICATE

**make_exec/2:**

**Usage:** make_exec(File, ExecName)

- **Description:** Make a Ciao executable from file (or files) File, giving it name ExecName. If ExecName is a variable, the compiler will choose a default name for the executable and will bind the variable ExecName to that name. The name is chosen as follows: if the main prolog file has no .pl extension or we are in Windows, the executable will have extension .cpx; else the executable will be named as the main prolog file without extension.

- **The following properties should hold at call time:**
  - File is a source name or a list of source names. (ciaosh_doc:sourcenames/1)
- **The following properties hold upon exit:**
  - ExecName is an atom. (basic_props:atm/1)

PREDICATE

**include/1:**

**Usage:** include(File)

- **Description:** The contents of the file File are included in the top-level shell. For the moment, it only works with some directives, which are interpreted by the shell, or with normal clauses (which are asserted), if library(dynamic) is loaded beforehand.

- **The following properties should hold at call time:**
  - File is a source name. (streams_basic:sourcename/1)

PREDICATE

**use_package/1:**

**Usage:** use_package(Package)

- **Description:** Equivalent to issuing an include(library(Package)) for each listed file. By now some package contents cannot be handled.

- **The following properties should hold at call time:**
  - Package is a source name or a list of source names. (ciaosh_doc:sourcenames/1)

PREDICATE

**consult/1:**

**Usage:** consult(File)

- **Description:** Provided for backward compatibility. Similar to ensure_loaded/1, but ensuring each listed file is loaded in consult mode (see Chapter 6 [The interactive debugger], page 45).

- **The following properties should hold at call time:**
  - File is a source name or a list of source names. (ciaosh_doc:sourcenames/1)

PREDICATE

**compile/1:**

**Usage:** compile(File)

- **Description:** Provided for backward compatibility. Similar to ensure_loaded/1, but ensuring each listed file is loaded in compile mode (see Chapter 6 [The interactive debugger], page 45).

- **The following properties should hold at call time:**
  - File is a source name or a list of source names. (ciaosh_doc:sourcenames/1)
The Ciao Prolog System

./2:
Usage: .((File, Files)

- Description: Provided for backward compatibility, obsoleted by ensure_loaded/1.
- The following properties should hold at call time:
  File is a source name. (streams_basic:sourcename/1)
  Files is a list of sourcenames. (basic_props:list/2)

make_po/1:
Usage: make_po(Files)

- Description: Make object (.po) files from Files. Equivalent to executing "ciaoc -c" on the files.
- The following properties should hold at call time:
  Files is a source name or a list of source names. (ciaosh_doc:sourcenames/1)

unload/1:
Usage: unload(File)

- Description: Unloads dynamically loaded file File.
- The following properties should hold at call time:
  File is a source name. (streams_basic:sourcename/1)

set_debug_mode/1:
Usage: set_debug_mode(File)

- Description: Set the loading mode of File to consult. See Chapter 6 [The interactive debugger], page 45.
- The following properties should hold at call time:
  File is a source name. (streams_basic:sourcename/1)

set_nodebug_mode/1:
Usage: set_nodebug_mode(File)

- Description: Set the loading mode of File to compile. See Chapter 6 [The interactive debugger], page 45.
- The following properties should hold at call time:
  File is a source name. (streams_basic:sourcename/1)

make_actmod/2:
Usage: make_actmod(ModuleFile, PublishMod)

- Description: Make an active module executable from the module residing in ModuleFile, using address publish module of name PublishMod (which needs to be in the library paths).
- The following properties should hold at call time:
  ModuleFile is a source name. (streams_basic:sourcename/1)
  PublishMod is an atom. (basic_props:atm/1)
force_lazy/1:  
Usage: force_lazy(Module)

- Description: Force module of name Module to be loaded lazily in the subsequent created executables.
- The following properties should hold at call time:
  Module is an atom. (basic_props:atm/1)

undo_force_lazy/1:  
Usage: undo_force_lazy(Module)

- Description: Disable a previous force_lazy/1 on module Module (or, if it is uninstantiated, all previous force_lazy/1).
- Calls should, and exit will be compatible with:
  Module is an atom. (basic_props:atm/1)

dynamic_search_path/1:  
Usage: dynamic_search_path(Name)

- Description: Asserting a fact to this data predicate, files using path alias Name will be treated as dynamic in the subsequent created executables.
- The following properties should hold at call time:
  Name is an atom. (basic_props:atm/1)

multifile/1:  
Usage: multifile Pred

- Description: Dynamically declare predicate Pred as multifile. This is useful at the top-level shell to be able to call multifile predicates of loaded files.
- The following properties should hold at call time:
  Pred is a Name/Arity structure denoting a predicate name:
  predname(P/A) :-
     atm(P),
     nnegint(A).
  (basic_props:predname/1)

5.6 Documentation on internals (ciaosh)
sourcenames/1:  
Is defined as follows:

sourcenames(File) :-
    sourcename(File).
sourcenames(Files) :-
    list(Files,sourcename).

See sourcename/1 in Chapter 21 [Basic file/stream handling], page 111
Usage: sourcenames(Files)

- Description: Files is a source name or a list of source names.
Chapter 6: The interactive debugger


Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#67 (2003/12/19, 16:11:38 CET)

The Ciao program development environment includes a number of advanced debugging tools, such as a source-level debugger, the ciaopp preprocessor, and some execution visualizers. Herein we discuss the interactive debugger available in the standard top-level, which allows tracing the control flow of programs, in a similar way to other popular Prolog systems. This is a classical Byrd box-type debugger [Byr80,BBP81], with some enhancements, most notably being able to track the execution on the source program. We also discuss the embedded debugger, which is a version of the debugger which can be embedded into executables so that an interactive debugging session can be triggered at any time while running that executable without needing the top-level shell.

Byrd’s Procedure Box model of debugging execution provides a simple way of visualising control flow, including backtracking. Control flow is in principle viewed at the predicate level, rather than at the level of individual clauses. The Ciao debugger has the ability to mark selected modules and/or files for debugging (traditional and source debugging), rather than having to exhaustively trace your program. It also allows to selectively set spy-points and breakpoints. Spy-points allow the programmer to nominate interesting predicates at which program execution is to pause so that the programmer can interact with the debugger. Breakpoints are similar to spy-points, but allow pausing at a specific line in the code, corresponding to a particular literal. There is a wide choice of control and information options available during debugging interaction.

Note: While the debugger described herein can be used in a standalone way (i.e., from a operating system shell or terminal window) in the same way as other Prolog debuggers, the most convenient way of debugging Ciao programs is by using the emacs mode (see [Using Ciao inside GNU emacs], page (undefined)), i.e., debugging from within the emacs editor / programming environment.

6.1 Marking modules and files for debugging in the top-level debugger

Usually, when a program is not working properly, the programmer has a feeling of which are the modules where the fault may be. Since full-fledged debugging is only available on interpreted (called interpreted mode in traditional Prolog systems) modules, which are executed much slower than compiled modules, there is the possibility of telling the top level which particular modules are to be loaded in interpreted mode, with the aim of debugging them. The simplest way of achieving this is by executing in the Ciao shell prompt, for each suspicious module Module in the program, a command like this:

?- debug_module(Module).

An alternative way of loading a module in interpreted mode exists which will instruct the debugger to keep track of the line numbers in the source file and to report them during debugging. This feature can be selected for a suspicious module Module in the program by executing a command such as:

?- debug_module_source(Module).

This is most useful when running the top-level inside the emacs editor since in that case the Ciao emacs mode allows performing full source-level debugging in each module marked as above, i.e., the source lines being executed will be highlighted dynamically during debugging in a window showing the source code of the module.
Note that all files with no module declaration belong to the pseudo-module user, so the command to be issued for debugging a user file, say foo.pl, would be debug_module(user) or debug_module_source(user), and not debug_module(foo).

The two ways of performing source-level debugging are fully compatible between them, i.e., Ciao allows having some modules loaded with debug_module/1 and others with debug_module_source/1. To change from one interpreted mode to the other mode it suffices to select the module with the new interpreted mode (debugger mode), using the appropriate command, and reload the module.

The commands above perform in fact two related actions: first, they let the compiler know that if a file containing a module with this name is loaded, it should be loaded in interpreted mode (source or traditional). In addition, they instruct the debugger to actually prepare for debugging the code belonging to that module. After that, the modules which are to be debugged have to be (re)loaded so that they are compiled or loaded for interpretation in the appropriate way. The nice thing is that, due to the modular behaviour of the compiler/top-level, if the modules are part of a bigger application, it suffices to load the main module of the application, since this will automatically force the dependent modules which have changed to be loaded in the appropriate way, including those whose loading mode has changed (i.e., changing the loading mode has the effect of forcing the required re-loading of the module at the appropriate time).

Later in the debugging process, as the bug location is isolated, typically one will want to restrict more and more the modules where debugging takes place. To this end, and without the need for reloading, one can tell the debugger to not consider a module for debugging issuing a nodebug_module/1 command, which counteracts a debug_module/1 or debug_module_source/1 command with the same module name, and reloading it (or the main file).

There are also two top-level commands set_debug_mode/1 and set_nodebug_mode/1, which accept as argument a file spec (i.e., library(foo) or foo, even if it is a user file) to be able to load a file in interpreted mode without changing the set of modules that the debugger will try to spy.

### 6.2 The debugging process

Once modules or user files are marked for debugging and reloaded, the traditional debugging shell commands can be used (the documentation of the debugger library following this chapter contains all the commands and their description), with the same meaning as in other classical Prolog systems. The differences in their behavior are:

- Debugging takes place only in the modules in which it was activated,
- nospy/1 and spy/1 accept sequences of predicate specs, and they will search for those predicates only in the modules marked for debugging (traditional or source-level debugging).
- breakpt/6 and nobreakpt/6 allow setting breakpoints at selected clause literals and will search for those literals only in the modules marked for source-level debugging (modules marked with debug_module_source/1).

In particular, the system is initially in nodebug mode, in which no tracing is performed. The system can be put in debug mode by a call to debug/0 in which execution of queries will proceed until the first spy-point or breakpoint. Alternatively, the system can be put in trace mode by a call to trace/0 in which all predicates will be trace.

### 6.3 Marking modules and files for debugging with the embedded debugger

The embedded debugger, as the interpreted debugger, has three different modes of operation: debug, trace or nodebug. These debuggers modes can be set by adding a package declaration in the module, as follows:
Chapter 6: The interactive debugger

:- use_package(debug).
:- use_package(trace).
:- use_package(nodebug).

and recompiling the application.

In order to debug, or trace, correctly the complete code these declarations must appear the last ones of all use_package declarations used. Also it is possible, as usual, to add the debugging package(s) in the module declaration using the predicate module/3 (and they should also be the last ones).

The embedded debugger has limitations over the interpreted debugger. The most important is that the “retry” option is not available. But it is possible to add, and remove, spy-points and breakpoints using the predicates spy/1, nospy/1, breakpt/6 and nobreakpt/6, etc. These can be used in a clause declaration or as declarations. Also it is possible to add in the code predicates for issuing the debugger (i.e., use debug mode, and in a clause add the predicate trace/1).

The nodebug mode allows keeping the spy-points and breakpoints in the code instead of removing them from the code.

Note that there is a particularly interesting way of using the embedded debugger: if an application is run in a shell buffer which has been set with Ciao inferior mode (M-x ciao-inferior-mode) and this application starts emitting output from the embedded debugger (i.e., which contains the embedded debugger and is debugging its code) then the Ciao emacs mode will be able to follow these messages, for example tracking execution in the source level code. This also works if the application is written in a combination of languages, provided the parts written in Ciao are compiled with the embedded debugger package and is thus a convenient way of debugging multi-language applications. The only thing needed is to make sure that the output messages appear in a shell buffer that is in Ciao inferior mode.

See the following as a general example of use of the embedded debugger:

```prolog
:- module( foo, [main/1], [assertions, debug]).

:- entry main/1.

main(X) :-
    display(X),
    spy(foo),
    foo(X),
    notrace,
    nl.

foo([]).
foo([X|T]) :-
    trace,
    bar(X),
    foo(T).

bar(X) :-
    display(X).
```

6.4 The procedure box control flow model

During debugging the interpreter prints out a sequence of goals in various states of instantiation in order to show the state that the program has reached in its execution. However, in
order to understand what is occurring it is necessary to understand when and why the interpreter prints out goals. As in other programming languages, key points of interest are procedure entry and return, but in Prolog there is the additional complexity of backtracking. One of the major confusions that novice Prolog programmers have to face is the question of what actually happens when a goal fails and the system suddenly starts backtracking. The Procedure Box model of Prolog execution views program control flow in terms of movement about the program text. This model provides a basis for the debugging mechanism in the interpreter, and enables the user to view the behaviour of the program in a consistent way. It also provides the basis for the visualization performed on the source level program when source level program when source-level debugging is activated within `emacs`.

Let us look at an example Prolog procedure:

```
descendant(Z,Y).
descendant(X,Y):- offspring(X,Z),
descendant(X,Y):- offspring(X,Y).
```

The first clause states that $Y$ is a descendant of $X$ if $Y$ is an offspring of $X$, and the second clause states that $Y$ is a descendant of $X$ if $Z$ is an offspring of $X$ and $Y$ is a descendant of $Z$. In the diagram a box has been drawn around the whole procedure and labelled arrows indicate the control flow in and out of this box. There are four such arrows which we shall look at in turn.

- **Call**
  This arrow represents initial invocation of the procedure. When a goal of the form `descendant(X,Y)` is required to be satisfied, control passes through the Call port of the descendant box with the intention of matching a component clause and then satisfying any subgoals in the body of that clause. Note that this is independent of whether such a match is possible; i.e. first the box is called, and then the attempt to match takes place. Textually we can imagine moving to the code for descendant when meeting a call to descendant in some other part of the code.

- **Exit**
  This arrow represents a successful return from the procedure. This occurs when the initial goal has been unified with one of the component clauses and any subgoals have been satisfied. Control now passes out of the Exit port of the descendant box. Textually we stop following the code for descendant and go back to the place we came from.

- **Redo**
  This arrow indicates that a subsequent goal has failed and that the system is backtracking in an attempt to find alternatives to previous solutions. Control passes through the Redo port of the descendant box. An attempt will now be made to resatisfy one of the component subgoals in the body of the clause that last succeeded; or, if that fails, to completely rematch the original goal with an alternative clause and then try to satisfy any subgoals in the body of this new clause. Textually we follow the code backwards up the way we came looking for new ways of succeeding, possibly dropping down on to another clause and following that if necessary.

- **Fail**
  This arrow represents a failure of the initial goal, which might occur if no clause is matched, or if subgoals are never satisfied, or if any solution produced is always rejected by later processing. Control now passes out of the Fail port of the descendant box and the system continues to backtrack. Textually we move back to the code which called this procedure and keep moving backwards up the code looking for choice points.

In terms of this model, the information we get about the procedure box is only the control flow through these four ports. This means that at this level we are not concerned with which
clause matches, and how any subgoals are satisfied, but rather we only wish to know the initial goal and the final outcome. However, it can be seen that whenever we are trying to satisfy subgoals, what we are actually doing is passing through the ports of their respective boxes. If we were following this (e.g., activating source-level debugging), then we would have complete information about the control flow inside the procedure box.

Note that the box we have drawn around the procedure should really be seen as an invocation box. That is, there will be a different box for each different invocation of the procedure. Obviously, with something like a recursive procedure, there will be many different Calls and Exits in the control flow, but these will be for different invocations. Since this might get confusing each invocation box is given a unique integer identifier in the messages, as described below.

Note that not all procedure calls are traced; there are a few basic predicates which have been made invisible since it is more convenient not to trace them. These include debugging directives, basic control structures, and some builtins. This means that messages will never be printed for these predicates during debugging.

6.5 Format of debugging messages

This section explains the two formats of the message output by the debugger at a port. All trace messages are output to the terminal regardless of where the current output stream is directed (which allows tracing programs while they are performing file I/O). The basic format, which will be shown in traditional debug and in source-level debugging within Ciao emacs mode, is as follows:

```
S 13 7 Call: T user:descendant(dani,_123) ?
```

S is a spy-point or breakpoint indicator. It is printed as '+', indicating that there is a spy-point on descendant/2 in module user, as 'B' denoting a breakpoint, or as ' ', denoting no spy-point or breakpoint. If there is a spy-point and a breakpoint in the same predicate the spy-point indicator takes preference over breakpoint indicator.

T is a subterm trace. This is used in conjunction with the ^ command (set subterm), described below. If a subterm has been selected, T is printed as the sequence of commands used to select the subterm. Normally, however, T is printed as ' ', indicating that no subterm has been selected.

The first number is the unique invocation identifier. It is always nondecreasing (provided that the debugger is switched on) regardless of whether or not the invocations are being actually seen. This number can be used to cross correlate the trace messages for the various ports, since it is unique for every invocation. It will also give an indication of the number of procedure calls made since the start of the execution. The invocation counter starts again for every fresh execution of a command, and it is also reset when retries (see later) are performed.

The number following this is the current depth; i.e., the number of direct ancestors this goal has. The next word specifies the particular port (Call, Exit, Redo or Fail). The goal is then printed so that its current instantiation state can be inspected. The final ? is the prompt indicating that the debugger is waiting for user interaction. One of the option codes allowed (see below) can be input at this point.

The second format, quite similar to the format explained above, is shown when using source-level debugging outside the Ciao emacs mode, and it is as follows:

```
In /home/mcarlos/ciao/foo.pl (5-9) descendant-1
S 13 7 Call: T user:descendant(dani,_123) ?
```

This format is identical to the format above except for the first line, which contains the information for location of the point in the source program text where execution is currently at. The first line contains the name of the source file, the start and end lines where the literal can be found, the subtring to search for between those lines and the number of subtrings to locate. This information for locating the point on the source file is not shown when executing the source-level debugger from the Ciao emacs mode.
Ports can be “unleashed” by calling the `leash/1` predicate omitting that port in the argument. This means that the debugger will stop but user interaction is not possible for an unleashed port. Obviously, the `?` prompt will not be shown in such messages, since the user has specified that no interaction is desired at this point.

6.6 Options available during debugging

This section describes the particular options that are available when the debugger prompts after printing out a debugging message. All the options are one letter mnemonics, some of which can be optionally followed by a decimal integer. They are read from the terminal with any blanks being completely ignored up to the next terminator (carriage-return, line-feed, or escape). Some options only actually require the terminator; e.g., the `creep` option, only requires `RET`.

The only option which really needs to be remembered is `h` (followed by `RET`). This provides help in the form of the following list of available options.

- `c (creep)`
  causes the debugger to single-step to the very next port and print a message. Then if the port is leashed the user is prompted for further interaction. Otherwise it continues creeping. If leashin is off, creep is the same as `leap` (see below) except that a complete trace is printed on the terminal.

- `l (leap)`
  causes the interpreter to resume running the program, only stopping when a spy-point or breakpoint is reached (or when the program terminates). Leaping can thus be used to follow the execution at a higher level than exhaustive tracing. All that is needed to do is to set spy-points and breakpoints on an evenly spread set of pertinent predicates or lines, and then follow the control flow through these by leaping from one to the other.

- `s (skip)`
  is only valid for Call and Redo ports, if it is issued in Exit or Fail ports it is equivalent to `creep`. It skips over the entire execution of the predicate. That is, no message will be seen until control comes back to this predicate (at either the Exit port or the Fail port). Skip is particularly useful while creeping since it guarantees that control will be returned after the (possibly complex) execution within the box. With skip then no message at all will appear until control returns to the Exit port or Fail port corresponding to this Call port or Redo port. This includes calls to predicates with spy-points and breakpoints set: they will be masked out during the skip. There is a way of overriding this: the `t` option after a `CTRL` interrupt will disable the masking. Normally, however, this masking is just what is required!
r (retry)
can be used at any of the four ports (although at the Call port it has no effect). It transfers control back to the Call port of the box. This allows restarting an invocation when, for example, it has left the programmer with some weird result. The state of execution is exactly the same as in the original call (unless the invocation has performed side effects, which will not be undone). When a retry is performed the invocation counter is reset so that counting will continue from the current invocation number regardless of what happened before the retry. This is in accord with the fact that execution has, in operational terms, returned to the state before anything else was called.

If an integer is supplied after the retry command, then this is taken as specifying an invocation number and the system tries to get to the Call port, not of the current box, but of the invocation box specified. It does this by continuously failing until it reaches the right place. Unfortunately this process cannot be guaranteed: it may be the case that the invocation the programmer is looking for has been cut out of the search space by cuts in the program. In this case the system fails to the latest surviving Call port before the correct one.

f (fail)
can be used at any of the four ports (although at the Fail port it has no effect). It transfers control to the Fail port of the box, forcing the invocation to fail prematurely. If an integer is supplied after the command, then this is taken as specifying an invocation number and the system tries to get to the Fail port of the invocation box specified. It does this by continuously failing until it reaches the right place. Unfortunately, as before, this process cannot be guaranteed.

d (display)
displays the current goal using display/1. See w below.

p (print)
re-prints the current goal using print/1. Nested structures will be printed to the specified printdepth (see below).

w (write)
writes the current goal on the terminal using write/1.

g (ancestors)
provides a list of ancestors to the current goal, i.e., all goals that are hierarchically above the current goal in the calling sequence. It is always possible to jump to any goal in the ancestor list (by using retry, etc.). If an integer n is supplied, then only n ancestors will be printed. That is to say, the last n ancestors will be printed counting back from the current goal. Each entry in the list is preceded by the invocation number followed by the depth number (as would be given in a trace message).

n (nodebug)
switches the debugger off. Note that this is the correct way to switch debugging off at a trace point. The @ option cannot be used because it always returns to the debugger.

= (debugging)
outputs information concerning the status of the current debugging session.

+ spy
sets a spy-point on the current goal.

- (nospy)
removes the spy-point from the current goal.

a (abort)
causes an abort of the current execution. All the execution states built so far are destroyed and the system is put right back at the top-level of the interpreter. (This is the same as the built-in predicate abort/0.)
allows calling arbitrary goals. The initial message \( ?- \) will be output on the terminal, and a command is then read from the terminal and executed as if it was at top-level.

- \( u \) (\textit{unify}( ))
  
  is available at the Call port and gives the option of providing a solution to the goal from the terminal rather than executing the goal. This is convenient, e.g., for providing a “stub” for a predicate that has not yet been written. A prompt \( 1: \) will be output on the terminal, and the solution is then read from the terminal and unified with the goal.

- \( < \) (\textit{printdepth})
  
  sets a limit for the subterm nesting level that is printed in messages. While in the debugger, a printdepth is in effect for limiting the subterm nesting level when printing the current goal. When displaying or writing the current goal, all nesting levels are shown. The limit is initially 10. This command, without arguments, resets the limit to 10. With an argument of \( n \) the limit is set to \( n \).

- \( ^{\sim} \) (\textit{subterm})
  
  sets the subterm to be printed in messages. While at a particular port, a current subterm of the current goal is maintained. It is the current subterm which is displayed, printed, or written when prompting for a debugger command. Used in combination with the printdepth, this provides a means for navigating in the current goal for focusing on the part which is of interest. The current subterm is set to the current goal when arriving at a new port. This command, without arguments, resets the current subterm to the current goal. With an argument of \( n \) (greater than 0 and less or equal to the number of subterms of the current subterm), the current subterm is replaced by its \( n \)th subterm. With an argument of 0, the current subterm is replaced by its parent term.

- \( ? \) or \( h \) (\textit{help})
  
  displays the table of options given above.

### 6.7 Calling predicates that are not exported by a module

The Ciao module system does not allow calling predicates which are not exported during debugging. However, as an aid during debugging, this is allowed (only from the top-level and for modules which are in debug mode or source-level debug mode) using the \texttt{call\_in\_module/2} predicate.

Note that this does not affect analysis or optimization issues, since it only works on modules which are loaded in debug mode or source-level debug mode, i.e. unoptimized.

### 6.8 Acknowledgements

Originally written by Andrzej Ciepielewski. Minor modifications by Mats Carlsson. Later modifications (17 Dec 87) by Takashi Chikayama (making tracer to use \texttt{print/1} rather than \texttt{write/1}, temporarily switching debugging flag off while writing trace message and within “break” level). Additional modifications by Kish Shen (May 88): subterm navigation, handle unbound args in \texttt{spy/1} and \texttt{nospy/1}, trapping arithmetics errors in debug mode. Adapted then to &-Prolog and Ciao by D. Cabeza and included in the Ciao version control system. Extended for source-level debugging by Manuel C. Rodríguez. (See changelog if included in the document for more detailed documentation of the later changes.)
7 Predicates controlling the interactive debugger

Author(s): A. Ciepielewski, M. Carlsson, T. Chikayama, K. Shen, D. Cabeza, M. Rodriguez.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#59 (2003/11/26, 3:11:51 CET)

This library implements predicates which are normally used in the interactive top-level shell to debug programs. A subset of them are available in the embeddable debugger.

7.1 Usage and interface (debugger)

- Library usage:
  ```prolog
  :- use_module(library(debugger)).
  ```
- Exports:
  - Predicates:
    ```prolog
    debug_module/1, nodebug_module/1, debug_module_source/1, debug/0, nodebug/0,
    trace/0, notrace/0, spy/1, nospy/1, nospyall/0, breackpt/6, nobreakpt/6,
    nobreakall/0, list_breakpt/0, debugging/0, leash/1, maxdepth/1, call_in_module/2,
    adjust_debugger_state/2.
    ```
- Other modules used:
  - System library modules:
    ```prolog
    debugger/debugger_lib, format, ttyout, read, system, write, aggregates, sort.
    ```

7.2 Documentation on exports (debugger)

**debug_module/1:**

**PREDICATE**

Usage: `debug_module(Module)`

- Description: The debugger will take into account module `Module` (assuming it is loaded in interpreted mode). When issuing this command at the top-level shell, the compiler is instructed also to set to `interpret` the loading mode of files defining that module and also to mark it as 'modified' so that (re)loading this file or a main file that uses this module will force it to be reloaded for source-level debugging.
- The following properties should hold at call time:
  ```prolog
  Module is an atom.  (basic_props:atm/1)
  ```

**nodebug_module/1:**

**PREDICATE**

Usage: `nodebug_module(Module)`

- Description: The debugger will not take into account module `Module`. When issuing this command at the top-level shell, the compiler is instructed also to set to `compile` the loading mode of files defining that module.
- The following properties should hold at call time:
  ```prolog
  Module is an atom.  (basic_props:atm/1)
  ```
debug_module_source/1:

**Usage:** debug_module_source(Module)

- **Description:** The debugger will take into account module Module (assuming it is is loaded in source-level debug mode). When issuing this command at the toplevel shell, the compiler is instructed also to set to interpret the loading mode of files defining that module and also to mark it as 'modified' so that (re)loading this file or a main file that uses this module will force it to be reloaded for source-level debugging.
- The following properties should hold at call time:
  
  Module is an atom. (basic_props:atm/1)

debug/0:

**Usage:**

- **Description:** Switches the debugger on. The interpreter will stop at all ports of procedure boxes of spied predicates.

nodebug/0:

**Usage:**

- **Description:** Switches the debugger off. If there are any spy-points set then they will be kept but disabled.

trace/0:

**Usage:**

- **Description:** Start tracing, switching the debugger on if needed. The interpreter will stop at all leashed ports of procedure boxes of predicates either belonging to debugged modules or called from clauses of debugged modules. A message is printed at each stop point, expecting input from the user (write h to see the available options).

notrace/0:

**Usage:**

- **Description:** Equivalent to nodebug/0.

spy/1:

**Usage:** spy(PredSpec)

- **Description:** Set spy-points on predicates belonging to debugged modules and which match PredSpec, switching the debugger on if needed. This predicate is defined as a prefix operator by the toplevel.
- The following properties should hold at call time:
  
  PredSpec is a sequence of multpredspecs. (basic_props:sequence/2)
nospy/1: PREDICATE
Usage: nospy(PredSpec)
Description: Remove spy-points on predicates belonging to debugged modules which match PredSpec. This predicate is defined as a prefix operator by the toplevel.
The following properties should hold at call time:
PredSpec is a sequence of multipredspecs.

nospyall/0: PREDICATE
Usage:
Description: Remove all spy-points.

breakpt/6: PREDICATE
Usage: breakpt(Pred, Src, Ln0, Ln1, Number, RealLine)
Description: Set a breakpoint in file Src between lines Ln0 and Ln1 at the literal corresponding to the Number'th occurrence of (predicate) name Pred. The pair Ln0-Ln1 uniquely identifies a program clause and must correspond to the start and end line numbers for the clause. The rest of the arguments provide enough information to be able to locate the exact literal that the RealLine line refers to. This is normally not issued by users but rather by the emacs mode, which automatically computes the different argument after selecting a point in the source file.
The following properties should hold at call time:
Pred is an atom.
Src is a source name.
Ln0 is an integer.
Ln1 is an integer.
Number is an integer.
RealLine is an integer.

nobreakpt/6: PREDICATE
Usage: nobreakpt(Pred, Src, Ln0, Ln1, Number, RealLine)
Description: Remove a breakpoint in file Src between lines Ln0 and Ln1 at the Number'th occurrence of (predicate) name Pred (see breakpt/6). Also normally used from the emacs mode.
The following properties should hold at call time:
Pred is an atom.
Src is a source name.
Ln0 is an integer.
Ln1 is an integer.
Number is an integer.
RealLine is an integer.
nobreakall/0:
  Usage:
  - Description: Remove all breakpoints.

list_breakpt/0:
  Usage:
  - Description: Prints out the location of all breakpoints. The location of the breakpoints is showed usual by referring to the source file, the lines between which the predicate can be found, the predicate name and the number of occurrence of the predicate name of the literal.

debugging/0:
  Usage:
  - Description: Display debugger state.

leash/1:
  Usage: leash(Ports)
  - Description: Leash on ports Ports, some of call, exit, redo, fail. By default, all ports are on leash.
  - The following properties should hold at call time:
    Ports is a list of ports. (basic_props:list/2)

maxdepth/1:
  Usage: maxdepth(MaxDepth)
  - Description: Set maximum invocation depth in debugging to MaxDepth. Calls to compiled predicates are not included in the computation of the depth.
  - The following properties should hold at call time:
    MaxDepth is an integer. (basic_props:int/1)

call_in_module/2:
  Usage: call_in_module(Module, Predicate)
  - Description: Calls predicate Predicate belonging to module Module, even if that module does not export the predicate. This only works for modules which are in debug (interpreted) mode (i.e., they are not optimized).
  - The following properties should hold at call time:
    Module is an atom. (basic_props:atm/1)
    Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

adjust_debugger_state/2:
  No further documentation available for this predicate.
7.3 Documentation on internals (debugger)

**multpredspec/1:**

A property, defined as follows:

```
multpredspec(Mod:Spec) :-
    atm(Mod),
    multpredspec(Spec).
multpredspec(Name/Low-High) :-
    atm(Name),
    int(Low),
    int(High).
multpredspec(Name/(Low-High)) :-
    atm(Name),
    int(Low),
    int(High).
multpredspec(Name/Arity) :-
    atm(Name),
    int(Arity).
multpredspec(Name) :-
    atm(Name).
```

7.4 Known bugs and planned improvements (debugger)

- Add an option to the emacs menu to automatically select all modules in a project.
- Consider the possibility to show debugging messages directly in the source code emacs buffer.
8 The script interpreter

Author(s): Daniel Cabeza, Manuel Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.5#130 (2000/5/3, 20:19:4 CEST)

ciao-shell is the Ciao script interpreter. It can be used to write Prolog shell scripts (see [Her96,CHV96b]), that is, executable files containing source code, which are compiled on demand.

Writing Prolog scripts can sometimes be advantageous with respect to creating binary executables for small- to medium-sized programs that are modified often and perform relatively simple tasks. The advantage is that no explicit compilation is necessary, and thus changes and updates to the program imply only editing the source file. The disadvantage is that startup of the script (the first time after it is modified) is slower than for an application that has been compiled previously.

An area of application is, for example, writing CGI executables: the slow speed of the network connection in comparison with that of executing a program makes program execution speed less important and has made scripting languages very popular for writing these applications. Logic languages are, a priori, excellent candidates to be used as scripting languages. For example, the built-in grammars and databases can sometimes greatly simplify many typical script-based applications.

8.1 How it works

Essentially, ciao-shell is a smaller version of the Ciao top-level, which starts by loading the file given to it as the first argument and then starts execution at main/1 (the argument is instantiated to a list containing the command line options, in the usual way). Note that the Prolog script cannot have a module declaration for this to work. While loading the file, ciao-shell changes the prolog flag quiet so that no informational or warning messages are printed (error messages will be reported to user_error, however). The operation of ciao-shell in Unix-like systems is based in a special compiler feature: when the first character of a file is '#', the compiler skips the first lines until an empty line is found. In Windows, its use is as easy as naming the file with a .pls extension, which will launch ciao-shell appropriately.

For example, in a Linux/Unix system, assume a file called hello contains the following program:

```
#!/bin/sh
exec ciao-shell $0 "$@
# --** mode: ciao; **-
main(_)
write('Hello world'), nl.
```

Then, the file hello can be run by simply making it executable and invoking it from the command line:

```
/herme@clip:/tmp
[86]> chmod +x hello

/herme@clip:/tmp
[87]> hello
Hello world
```

The line:

```
#!/bin/sh
```

invokes the /bin/sh shell which will interpret the following line:
exec ciao-shell $0 "$0" # -*- mode: ciao; -*-
and invoke ciao-shell, instructing it to read this same file ($0), passing it the rest of the arguments to hello as arguments to the prolog program. The second part of the line # -*- mode: ciao; -*- is simply a comment which is seen by emacs and instructs it to edit this file in Ciao mode (this is needed because these script files typically do not have a .pl ending). When ciao-shell starts, if it is the first time, it compiles the program (skipping the first lines, as explained above), or else at successive runs loads the .po object file, and then calls main/1.

Note that the process of creating Prolog scripts is made very simple by the Ciao emacs mode, which automatically inserts the header and makes the file executable (See (undefined) [Using Ciao inside GNU emacs], page (undefined)).

8.2 Command line arguments in scripts

The following example illustrates the use of command-line arguments in scripts. Assume that a file called say contains the following lines:

```bash
#!/bin/sh
exec ciao-shell $0 "$0" # -*- mode: ciao; -*-
main(Argv) :-
    write_list(Argv), nl.
write_list([]).
write_list([Arg|Args]) :-
    write(Arg),
    write(’,’),
    write_list(Args).
```

An example of use is:

```
/herme@clip:/tmp
[91]> say hello dolly
hello dolly
```
9 Customizing library paths and path aliases

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#66 (2003/12/19, 16:9:45 CET)

This library provides means for customizing, from environment variables, the libraries and path aliases known by an executable. Many applications of Ciao, including ciaoc, ciaosh, and ciao-shell make use of this library. Note that if an executable is created dynamic, it will try to load its components at startup, before the procedures of this module can be invoked, so in this case all the components should be in standard locations.

9.1 Usage and interface (libpaths)

- Library usage:
  ```prolog```
  :- use_module(library(libpaths)).
  ```prolog```

- Exports:
  - Predicates:
    - get_alias_path/0.
  - Multifiles:
    - file_search_path/2, library_directory/1.

- Other modules used:
  - System library modules:
    - system, lists.

9.2 Documentation on exports (libpaths)

get_alias_path/0: PREDICATE

get_alias_path(get_alias_path
Consult the environment variable 'CIAOALIASPATH' and add facts to predicates library_directory/1 and file_search_path/2 to define new library paths and path aliases. The format of 'CIAOALIASPATH' is a sequence of paths or alias assignments separated by colons, an alias assignment is the name of the alias, an '=' and the path represented by that alias (no blanks allowed). For example, given

CIAOALIASPATH=/home/bardo/ciao:contrib=/usr/local/lib/ciao

the predicate will define /home/bardo/ciao as a library path and /usr/local/lib/ciao as the path represented by 'contrib'.

9.3 Documentation on multifiles (libpaths)

file_search_path/2: PREDICATE

See Chapter 21 [Basic file/stream handling], page 111.
The predicate is multifile.
The predicate is of type dynamic.
library_directory/1:
See Chapter 21 [Basic file/stream handling], page 111.
The predicate is m ult i file.
The predicate is of type d y n a m i c.
10 CiaoMode (library)

10.1 Usage and interface (CiaoMode)

- Library usage:
  
  ```prolog
  :- use_package(CiaoMode).
  or
  :- module(...,...,[CiaoMode]).
  ```
Author(s): The Clip Group.

This part documents the Ciao basic builtins. These predefined predicates and declarations are available in every program, unless the pure package is used (by using a :- module(_,_,[pure]). declaration or :- use_package(pure).). These predicates are contained in the engine directory within the lib library. The rest of the library predicates, including the packages that provide most of the ISO-Prolog builtins, are documented in subsequent parts.
Chapter 11: The module system

Author(s): Daniel Cabeza and the CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#28 (2002/11/20, 14:3:5 CET)

Modularity is a basic notion in a modern computer language. Modules allow dividing programs in several parts, which have their own independent name spaces. The module system in Ciao [CH00a], as in many other Prolog implementations, is procedure based. This means that predicate names are local to a module, but functor/atom names in data are shared.

The predicates visible in a module are the predicates defined in that module, plus the predicates imported from other modules. Only predicates exported by a module can be imported from other modules. The default module of a given predicate name is the local one if the predicate is defined locally, else the last module from which the predicate is imported, having explicit imports priority (that is, a predicate imported by an use_module/2 declaration is always preferred above a predicate imported by an use_module/1 declaration). To refer to a predicate from a module which is not the default for that predicate the name has to be module qualified. A module qualified predicate name has the form Module:Predicate as in the call debugger:debug_module(M). Note that this does not allow having access to predicates not imported, nor defining clauses of other modules.

All predicates defined in files with no module declaration belong to a special module called user, and all are implicitly exported. This allows dividing programs in several files without being aware of the module system at all. Note that this feature is only supported for compatibility reasons, being its use discouraged. Many attractive compilation features of Ciao cannot be performed in user modules.

The case of multile predicates (defined with the declaration multifile/1) is also special. Multile predicates can be defined by clauses distributed in several modules, and all modules which define a predicate as multile can use that predicate. The name space of multile predicates is independent, as if they belonged to special module multifile.

Every user or module file imports implicitly a number of modules called builtin modules. They are imported before all other importations of the module, allowing thus redefining any of their predicates (with the exception of true/0) by defining local versions or importing them from other modules. Importing explicitly from a builtin module, however, disables the implicit importation of the rest (this feature is used by package library(pure) to define pure prolog code).

11.1 Usage and interface (modules)

- Library usage:
  Modules are an intrinsic feature of Ciao, so nothing special has to be done to use them.

11.2 Documentation on internals (modules)

module/3: DECLARATION
Usage: :- module(Name, Exports, Packages).
Description: Declares a module of name Name which exports the predicates in Exports, and uses the packages in Packages. Name must match with the name of the file where the module resides, without extension. For each source in Packages,
a package file is included, as if by an include/1 declaration. If the source is specified with a path alias, this is the file included, if it is an atom, the library paths are searched. Package files provide functionalities by declaring imports from other modules, defining operators, new declarations, translations of code, etc.

This directive must appear the first in the file.

Also, if the compiler finds an unknown declaration as the first term in a file, the name of the declaration is regarded as a package library to be included, and the arguments of the declaration (if present) are interpreted like the arguments of module/3.

- The following properties hold at call time:
  Name is a module name (an atom). (modules:modulename/1)
  Exports is a list of prednames. (basic.props:list/2)
  Packages is a list of sourcenames. (basic.props:list/2)

**module/2:**

Usage: :- module(Name, Exports).

- Description: Same as directive module/3, with an implicit package default.
- The following properties hold at call time:
  Name is a module name (an atom). (modules:modulename/1)
  Exports is a list of prednames. (basic.props:list/2)

**export/1:**

Usage 1: :- export(Pred).

- Description: Adds Pred to the set of exported predicates.
- The following properties hold at call time:
  Pred is a Name/Arity structure denoting a predicate name:
  predname(P/A) :-
  atm(P), nnegint(A). (basic.props:predname/1)

Usage 2: :- export(Exports).

- Description: Adds Exports to the set of exported predicates.
- The following properties hold at call time:
  Exports is a list of prednames. (basic.props:list/2)

**use_module/2:**

Usage: :- use_module(Module, Imports).

- Description: Specifies that this code imports from the module defined in Module the predicates in Imports. The imported predicates must be exported by the other module.
- The following properties hold at call time:
  Module is a source name. (streams_basic:sourcename/1)
  Imports is a list of prednames. (basic.props:list/2)
use_module/1:  
Usage: :- use_module(Module).
- Description: Specifies that this code imports from the module defined in Module all the predicates exported by it. The previous version with the explicit import list is preferred to this as it minimizes the chances to have to recompile this code if the other module changes.
- The following properties hold at call time:
  Module is a source name. (streams_basic:sourcename/1)

import/2:  
Usage: :- import(Module, Imports).
- Description: Declares that this code imports from the module with name Module the predicates in Imports.
  Important note: this declaration is intended to be used when the current module or the imported module is going to be dynamically loaded, and so the compiler does not include the code of the imported module in the current executable (if only because the compiler cannot know the location of the module file at the time of compilation). For the same reason the predicates imported are not checked to be exported by Module. Its use in other cases is strongly discouraged, as it disallows many compiler optimizations.
- The following properties hold at call time:
  Module is a module name (an atom). (modules:modulename/1)
  Imports is a list of prednames. (basic_props:list/2)

reexport/2:  
Usage: :- reexport(Module, Preds).
- Description: Specifies that this code reexports from the module defined in Module the predicates in Preds. This implies that this module imports from the module defined in Module all the predicates exported by it, an also that this module exports the predicates in Preds.
- The following properties hold at call time:
  Module is a source name. (streams_basic:sourcename/1)
  Preds is a list of prednames. (basic_props:list/2)

reexport/1:  
Usage: :- reexport(Module).
- Description: Specifies that this code reexports from the module defined in Module all the predicates exported by it. This implies that this module imports from the module defined in Module all the predicates exported by it, an also that this module exports all such predicates.
- The following properties hold at call time:
  Module is a source name. (streams_basic:sourcename/1)
meta_predicate/1: DECLARATION
Usage: :- meta_predicate MetaSpecs.
- Description: Specifies that the predicates in MetaSpecs have arguments which represent predicates and thus have to be module expanded. The directive is only mandatory for exported predicates (in modules). This directive is defined as a prefix operator in the compiler.
- The following properties hold at call time:
  MetaSpecs is a sequence of metaspecs. (basic_prop:sequence/2)

modulename/1: REGTYPE
A module name is an atom, not containing characters ':' or '$'. Also, user and multifile are reserved, as well as the module names of all builtin modules (because in an executable all modules must have distinct names).
Usage: modulename(M)
- Description: M is a module name (an atom).

metaspec/1: REGTYPE
A meta-predicate specification for a predicate is the functor of that predicate applied to atoms which represent the kind of module expansion that should be done with the arguments. Possible contents are represented as:

  goal  This argument will be a term denoting a goal (either a simple or complex one) which will be called. For compatibility reasons it can be named as : as well.
  clause This argument will be a term denoting a clause.
  fact  This argument should be instantiated to a term denoting a fact (head-only clause).
  spec  This argument should be instantiated to a predicate name, as Functor/Arity.
  pred(N) This argument should be instantiated to a predicate construct to be called by means of a call/N predicate call (see call/2).
  addmodule This is in fact not a real meta-data specification. It specifies that in an argument added after this one will be passed the calling module, to allow handling more involved meta-data (e.g., lists of goals) by using conversion builtins.
  ?,+,-,_ These other values denote that this argument is not module expanded.
Usage: metaspec(M)
- Description: M is a meta-predicate specification.
12 Directives for using code in other files

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#30 (2002/11/20, 14:15:12 CET)

Documentation for the directives used to load code into Ciao Prolog (both from the toplevel shell and by other modules).

12.1 Usage and interface (loading_code)

- Library usage:
  These directives are builtin in Ciao, so nothing special has to be done to use them.

12.2 Documentation on internals (loading_code)

ensure_loaded/1:

Usage: :- ensure_loaded(File).

- Description: Specifies that the code present in File will be included in the executable being prepared, in the user module. The file File cannot have a module declaration. This directive is intended to be used by programs not divided in modules. Dividing programs into modules is however strongly encouraged, since most of the attractive features of Ciao (such as static debugging and global optimization) are only partially available for user modules.
- The following properties should hold at call time:
  File is a source name.  (streams_basic:sourcename/1)

include/1:

Usage: :- include(File).

- Description: The contents of the file File are included in the current program text exactly as if they had been written in place of this directive.
- The following properties should hold at call time:
  File is a source name.  (streams_basic:sourcename/1)

use_package/1:

:- use_package(Package).

Specifies the use in this file of the packages defined in Package. See the description of the third argument of module/3 for an explanation of package files.

This directive must appear the first in the file, or just after a module/3 declaration. A file with no module declaration, in the absence of this directive, uses an implicit package default (see Chapter 30 [Other predicates and features defined by default], page 159).

Usage 1: :- use_package(Package).
The Ciao Prolog System

- The following properties should hold at call time:
  Package is a source name. (streams_basic:sourcename/1)

Usage 2: :- use_package(Package).
- The following properties should hold at call time:
  Package is a list of sourcenames. (basic_props:list/2)
13 Control constructs/predicates

Author(s): Daniel Cabeza, Manuel Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#204 (2004/2/27, 19:38:32 CET)

This module contains the set of basic control predicates, except the predicates dealing with exceptions, which are in Chapter 23 [Exception handling], page 129.

13.1 Usage and interface (basiccontrol)

- **Library usage:**
  These predicates/constructs are built-in in Ciao, so nothing special has to be done to use them. In fact, as they are hardwired in some parts of the system, most of them cannot be redefined.

- **Exports:**
  - **Predicates:**
    - ,/2, ;/2, ->/2, !/0, \+/1, if/3, true/0, fail/0, repeat/0, false/0, otherwise/0, $metachoice/1, $metacut/1, interpret_goal/2, interpret_compiled_goal/2, undefined_goal/1, debug_goal/1.

- **Other modules used:**
  - **System library modules:**
    - debugger/debugger.

13.2 Documentation on exports (basiccontrol)

,/2:

\[ P , Q \]

Conjunction (P and Q).

Meta-predicate with arguments: goal,goal.

Usage:

- The following properties should hold at call time:
  - P is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  - Q is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

;/2:

\[ P ; Q \]

Disjunction (P or Q).

Meta-predicate with arguments: goal;goal.

Usage:

- ISO
- The following properties should hold at call time:
  P is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  Q is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

\( \rightarrow/2:\)  
\( P \rightarrow Q \)
If \( P \) then \( Q \) else fail, using first solution of \( P \) only. Also, \( (P \rightarrow Q ; R) \), if \( P \) then \( Q \) else \( R \), using first solution of \( P \) only. No cuts are allowed in \( P \).

Meta-predicate with arguments: goal->goal.

Usage:
- The following properties should hold at call time:
  P is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  Q is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

\!/0:\)  
Usage:
- Description: Commit to any choices taken in the current predicate.

\!/1:\)  
\(+\ P\)
Goal \( P \) is not provable (negation by failure). Fails if \( P \) has a solution, and succeeds otherwise. No cuts are allowed in \( P \).

Meta-predicate with arguments: \(+\)goal.

Usage:
- The following properties should hold at call time:
  P is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP as \( \text{not}(X) \). (basic_props:native/2)

if/3:\)  
if(\( P, Q, R \))
If \( P \) then \( Q \) else \( R \), exploring all solutions of \( P \). No cuts are allowed in \( P \).

Meta-predicate with arguments: if(goal,goal,goal).

Usage:
The following properties should hold at call time:

- P is a term which represents a goal, i.e., an atom or a structure.  
  (basic_props:callable/1)
- Q is a term which represents a goal, i.e., an atom or a structure.  
  (basic_props:callable/1)
- R is a term which represents a goal, i.e., an atom or a structure.  
  (basic_props:callable/1)

true/0:

Usage:

- Description: Succeed (noop).
- The following properties hold globally:
  This predicate is understood natively by CiaoPP.  
  (basic_props:native/1)

fail/0:

Usage:

- Description: Fail, backtrack immediately.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP.  
  (basic_props:native/1)

repeat/0:

Usage:

- Description: Generates an infinite sequence of backtracking choices.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP.  
  (basic_props:native/1)

false/0:

No further documentation available for this predicate.

otherwise/0:

No further documentation available for this predicate.

$metachoice/1:

Usage:

- The following properties hold upon exit:
  Arg1 is an integer.  
  (basic_props:int/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP.  
  (basic_props:native/1)
$metacut/1:  
Usage:  
- The following properties should hold at call time:  
  Arg1 is an integer.  
- The following properties hold globally:  
  This predicate is understood natively by CiaoPP.  
  \( \text{(basic_props:int/1)} \)

interpret_goal/2:  
No further documentation available for this predicate.

interpret_compiled_goal/2:  
No further documentation available for this predicate.

undefined_goal/1:  
No further documentation available for this predicate.

debug_goal/1:  
No further documentation available for this predicate.

13.3 Documentation on internals (basiccontrol)

\( /2: \)  
An alias for disjunction (when appearing outside a list). The alias is performed when terms are read in.
Chapter 14: Basic builtin directives

Author(s): Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#29 (2002/11/20, 14:4:17 CET)

This chapter documents the basic builtin directives in Ciao, additional to the documented in other chapters. These directives are natively interpreted by the Ciao compiler (ciaoc).

Unlike in other Prolog systems, directives in Ciao are not goals to be executed by the compiler or top level. Instead, they are read and acted upon by these programs. The advantage of this is that the effect of the directives is consistent for executables, code loaded in the top level, code analyzed by the preprocessor, etc.

As a result, by default only the builtin directives or declarations defined in this manual can be used in user programs. However, it is possible to define new declarations using the new_declaration/1 and new_declaration/2 directives (or using packages including them). Also, packages may define new directives via code translations.

14.1 Usage and interface (builtin_directives)

- Library usage:
  These directives are builtin in Ciao, so nothing special has to be done to use them.

14.2 Documentation on internals (builtin_directives)

multifile/1:

Usage: :- multifile Predicates.

- Description: Specifies that each predicate in Predicates may have clauses in more than one file. Each file that contains clauses for a multifile predicate must contain a directive multifile for the predicate. The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

- The following properties should hold at call time:
  Predicates is a sequence or list of prednames. (basic_props:sequence_or_list/2)

discontiguous/1:

Usage: :- discontiguous Predicates.

- Description: Specifies that each predicate in Predicates may be defined in this file by clauses which are not in consecutive order. Otherwise, a warning is signaled by the compiler when clauses of a predicate are not consecutive (this behavior is controllable by the prolog flag discontiguous_warnings). The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

- The following properties should hold at call time:
  Predicates is a sequence or list of prednames. (basic_props:sequence_or_list/2)
impl_defined/1:
Usage: :- impl_defined(Predicates).

- Description: Specifies that each predicate in Predicates is implicitly defined in the current prolog source, either because it is a builtin predicate or because it is defined in a C file. Otherwise, a warning is signaled by the compiler when an exported predicate is not defined in the module or imported from other module.
- The following properties should hold at call time:
  Predicates is a sequence or list of prednames. (basic_props:sequence_or_list/2)

redefining/1:
Usage: :- redefining(Predicate).

- Description: Specifies that this module redefines predicate Predicate, also imported from other module, or imports it from more than one module. This prevents the compiler giving warnings about redefinitions of that predicate. Predicate can be partially (or totally) uninstantiated, to allow disabling those warnings for several (or all) predicates at once.
- The following properties should hold at call time:
  Predicate is compatible with predname (basic_props:compat/2)

initialization/1:
Usage: :- initialization(Goal).

- Description: Goal will be executed at the start of the execution of any program containing the current code. The initialization of a module/file never runs before the initializations of the modules from which the module/file imports (excluding circular dependences).
- The following properties should hold at call time:
  Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

on_abort/1:
Usage: :- on_abort(Goal).

- Description: Goal will be executed after an abort of the execution of any program containing the current code.
- The following properties should hold at call time:
  Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
15 Basic data types and properties

Author(s): Daniel Cabeza, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#166 (2004/2/3, 21:12:16 CET)

This library contains the set of basic properties used by the builtin predicates, and which constitute the basic data types and properties of the language. They can be used both as type testing builtins within programs (by calling them explicitly) and as properties in assertions.

15.1 Usage and interface (basic_props)

- Library usage:

These predicates are builtin in Ciao, so nothing special has to be done to use them.

- Exports:

- Properties:
  - member/2, compat/2, iso/1, not_further_inst/2, sideff/2, regtype/1, native/1, native/2, eval/1.
  - Regular Types:
    - term/1, int/1, nnegint/1, flt/1, num/1, atm/1, struct/1, gnd/1, constant/1, callable/1, operator_specifier/1, list/1, list/2, sequence/2, sequence_or_list/2, character_code/1, string/1, predname/1, atm_or_atm_list/1.

15.2 Documentation on exports (basic_props)

term/1: REGTYPE

The most general type (includes all possible terms).

General properties: term(X)
- The following properties hold globally:
  term(X) is side-effect free. (basic_props:sideff/2)

Usage: term(X)
- Description: X is any term.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

int/1: REGTYPE

The type of integers. The range of integers is \([-2^{2147483616}, 2^{2147483616})\). Thus for all practical purposes, the range of integers can be considered infinite.

General properties: int(T)
- The following properties hold globally:
  int(T) is side-effect free. (basic_props:sideff/2)

Usage: int(T)
- Description: T is an integer.
The following properties hold globally:
This predicate is understood natively by CiaoPP. (basic_props:native/1)

\texttt{nnegint/1:} \hspace{1cm} \text{REGTYPE}

The type of non-negative integers, i.e., natural numbers.

\textbf{General properties:} \texttt{nnegint(T)}
- The following properties hold globally:
  \texttt{nnegint(T)} is side-effect free. (basic_props:sideff/2)

\textbf{Usage:} \texttt{nnegint(T)}
- Description: \( T \) is a non-negative integer.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

\texttt{flt/1:} \hspace{1cm} \text{REGTYPE}

The type of floating-point numbers. The range of floats is the one provided by the C \texttt{double} type, typically \([4.9e-324, 1.8e+308]\) (plus or minus). There are also three special values: Infinity, either positive or negative, represented as \(1.0e1000\) and \(-1.0e1000\); and Not-a-number, which arises as the result of indeterminate operations, represented as \(0.\text{NaN}\).

\textbf{General properties:} \texttt{flt(T)}
- The following properties hold globally:
  \texttt{flt(T)} is side-effect free. (basic_props:sideff/2)

\textbf{Usage:} \texttt{flt(T)}
- Description: \( T \) is a float.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

\texttt{num/1:} \hspace{1cm} \text{REGTYPE}

The type of numbers, that is, integer or floating-point.

\textbf{General properties:} \texttt{num(T)}
- The following properties hold globally:
  \texttt{num(T)} is side-effect free. (basic_props:sideff/2)

\textbf{Usage:} \texttt{num(T)}
- Description: \( T \) is a number.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

\texttt{atm/1:} \hspace{1cm} \text{REGTYPE}

The type of atoms, or non-numeric constants. The size of atoms is unbound.

\textbf{General properties:} \texttt{atm(T)}
The following properties hold globally:
\[ \text{atm}(T) \text{ is side-effect free.} \]

**Usage:** \text{atm}(T)

- **Description:** \(T\) is an atom.

The following properties hold globally:

**Usage:** \text{atm}(T)

- **Description:** \(T\) is an atom.

The following properties hold globally:

**Usage:** \text{atm}(T)

- **Description:** \(T\) is an atom.

The following properties hold globally:

This predicate is understood natively by CiaoPP.

**Usage:** \text{struct}(T)

- **Description:** \(T\) is a compound term.

This predicate is understood natively by CiaoPP.

**Usage:** \text{struct}(T)

- **Description:** \(T\) is a compound term.

This predicate is understood natively by CiaoPP.

**Usage:** \text{gnd}(T)

- **Description:** \(T\) is ground.

This predicate is understood natively by CiaoPP.

**Usage:** \text{gnd}(T)

- **Description:** \(T\) is an atomic term (an atom or a number).

This predicate is understood natively by CiaoPP.

**Usage:** \text{constant}(T)

- **Description:** \(T\) is a term which represents a goal, i.e., an atom or a structure.
operator_specifier/1:

The type and associativity of an operator is described by the following mnemonic atoms:

- **xfx** Infix, non-associative: it is a requirement that both of the two subexpressions which are the arguments of the operator must be of lower precedence than the operator itself.
- **xfy** Infix, right-associative: only the first (left-hand) subexpression must be of lower precedence; the right-hand subexpression can be of the same precedence as the main operator.
- **yfx** Infix, left-associative: same as above, but the other way around.
- **fx** Prefix, non-associative: the subexpression must be of lower precedence than the operator.
- **fy** Prefix, associative: the subexpression can be of the same precedence as the operator.
- **xf** Postfix, non-associative: the subexpression must be of lower precedence than the operator.
- **yf** Postfix, associative: the subexpression can be of the same precedence as the operator.

**General properties:** operator_specifier(X)

- The following properties hold globally:
  - operator_specifier(X) is side-effect free. (basic_props:sideff/2)

**Usage:** operator_specifier(X)

- Description: X specifies the type and associativity of an operator.

---

list/1:

A list is formed with successive applications of the functor `'.'/2, and its end is the atom `[]`. Defined as

```prolog
list([|L|]).
list([L]) :-
    list(L).
```

**General properties:** list(L)

- The following properties hold globally:
  - list(L) is side-effect free. (basic_props:sideff/2)

**Usage:** list(L)

- Description: L is a list.

---

list/2:

list(L, T)

L is a list, and for all its elements, T holds.

**Meta-predicate with arguments:** list(? , pred(1)).

**General properties:** list(L, T)

- The following properties hold globally:
  - list(L,T) is side-effect free. (basic_props:sideff/2)

**Usage:** list(L, T)

- Description: L is a list of Ts.
Chapter 15: Basic data types and properties

**member/2:**

**PROPERTY**

**General properties:** member(X, L)

- The following properties hold globally:
  
  member(X,L) is side-effect free.  
  
  (basic_props:sideff/2)

**Usage:** member(X, L)

- Description: X is an element of L.

**sequence/2:**

**REGTYPE**

A sequence is formed with zero, one or more occurrences of the operator ',', '/2. For example, a, b, c is a sequence of three atoms, a is a sequence of one atom.

**Meta-predicate** with arguments: sequence(? ,pred(1)).

**General properties:** sequence(S, T)

- The following properties hold globally:
  
  sequence(S,T) is side-effect free.  
  
  (basic_props:sideff/2)

**Usage:** sequence(S, T)

- Description: S is a sequence of Ts.

**sequence_or_list/2:**

**REGTYPE**

**Meta-predicate** with arguments: sequence_or_list(? ,pred(1)).

**General properties:** sequence_or_list(S, T)

- The following properties hold globally:
  
  sequence_or_list(S,T) is side-effect free.  
  
  (basic_props:sideff/2)

**Usage:** sequence_or_list(S, T)

- Description: S is a sequence or list of Ts.

**character_code/1:**

**REGTYPE**

**General properties:** character_code(T)

- The following properties hold upon exit:
  
  T is an integer.  
  
  (basic_props:int/1)

**Usage:** character_code(T)

- Description: T is an integer which is a character code.

**string/1:**

**REGTYPE**

A string is a list of character codes. The usual syntax for strings "string" is allowed, which is equivalent to [0’s,0’t,0’r,0’i,0’n,0’g] or [115,116,114,105,110,103]. There is also a special Ciao syntax when the list is not complete: "st"|R is equivalent to [0’s,0’t|R].

**General properties:** string(T)
The following properties hold upon exit:

\( T \) is a list of character_codes. \( \text{(basic\_props:list/2)} \)

\( \text{string(T)} \)

The following properties hold globally:

\( \text{string(T)} \) is side-effect free. \( \text{(basic\_props:sideff/2)} \)

**Usage:** \( \text{string(T)} \)

- **Description:** \( T \) is a string (a list of character codes).

---

**predname/1:**

**General properties:** \( \text{predname}(P) \)

The following properties hold globally:

\( \text{predname}(P) \) is side-effect free. \( \text{(basic\_props:sideff/2)} \)

**Usage:** \( \text{predname}(P) \)

- **Description:** \( P \) is a Name/Arity structure denoting a predicate name:

\[
\text{predname}(P/A) :- \\
\text{atm}(P), \\
\text{nnegint}(A).
\]

---

**atm\_or\_atm\_list/1:**

**General properties:** \( \text{atm\_or\_atm\_list}(T) \)

The following properties hold globally:

\( \text{atm\_or\_atm\_list}(T) \) is side-effect free. \( \text{(basic\_props:sideff/2)} \)

**Usage:** \( \text{atm\_or\_atm\_list}(T) \)

- **Description:** \( T \) is an atom or a list of atoms.

---

**compat/2:**

This property captures the notion of type or property compatibility. The instantiation or constraint state of the term is compatible with the given property, in the sense that assuming that imposing that property on the term does not render the store inconsistent. For example, terms \( X \) (i.e., a free variable), \( [Y|Z] \), and \( [Y,Z] \) are all compatible with the regular type \( \text{list/1} \), whereas the terms \( f(a) \) and \( [1|2] \) are not.

**Meta-predicate** with arguments: \( \text{compat(?\_pred(1))} \).

**General properties:** \( \text{compat}(Term, Prop) \)

The following properties hold globally:

\( \text{compat}(Term,Prop) \) is side-effect free. \( \text{(basic\_props:sideff/2)} \)

**Usage:** \( \text{compat}(Term, Prop) \)

- **Description:** \( Term \) is compatible with \( Prop \).
iso/1:  PROPERTY
   General properties: iso(G)
   - The following properties hold globally:
     iso(G) is side-effect free.
     (basic_props:sideff/2)
   Usage: iso(G)
   - Description: Complies with the ISO-Prolog standard.

not_further_instant/2:  PROPERTY
   General properties: not_further_instant(G, V)
   - The following properties hold globally:
     not_further_instant(G, V) is side-effect free.
     (basic_props:sideff/2)
   Usage: not_further_instant(G, V)
   - Description: V is not further instantiated.

sideff/2:  PROPERTY
sideff(G, X)
Declares that G is side-effect free (if its execution has no observable result other than its success, its failure, or its abortion), soft (if its execution may have other observable results which, however, do not affect subsequent execution, e.g., input/output), or hard (e.g., assert/retract).

   Meta-predicate with arguments: sideff(goal,?).
   General properties: sideff(G, X)
   - The following properties hold globally:
     This predicate is understood natively by CiaoPP.
     sideff(G, X) is side-effect free.
     (basic_props:sideff/2)
   Usage: sideff(G, X)
   - Description: G is side-effect X.
   - If the following properties should hold at call time:
     X is an element of [free, soft, hard].
     (basic_props:member/2)

regtype/1:  PROPERTY
   Meta-predicate with arguments: regtype(goal).
   General properties: regtype(G)
   - The following properties hold globally:
     regtype(G) is side-effect free.
     (basic_props:sideff/2)
   Usage: regtype(G)
   - Description: Defines a regular type.
native/1:  
*Meta-predicate* with arguments: native(goal).

**General properties:** native(P)
- *The following properties hold globally:*
  
  native(P) is side-effect free.  
  
  
  (basic_props:sideff/2)

**Usage:** native(Pred)
- *Description:* This predicate is understood natively by CiaoPP.

native/2:  
*Meta-predicate* with arguments: native(goal,?).

**General properties:** native(P, K)
- *The following properties hold globally:*
  
  native(P,K) is side-effect free.  
  
  (basic_props:sideff/2)

**Usage:** native(Pred, Key)
- *Description:* This predicate is understood natively by CiaoPP as Key.

eval/1:  
A property, defined as follows:
16 Extra-logical properties for typing

Author(s): Daniel Cabeza, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#167 (2004/2/3, 21:15:56 CET)

This library contains traditional Prolog predicates for testing types. They depend on the state of instantiation of their arguments, thus being of extra-logical nature.

16.1 Usage and interface (term_typing)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Properties:
    - var/1, nonvar/1, atom/1, integer/1, float/1, number/1, atomic/1, ground/1, type/2.

16.2 Documentation on exports (term_typing)

var/1: PROPERTY

General properties: var(X)

- The following properties hold globally:
  - X is not further instantiated. (basic_props:not_further_inst/2)
  - var(X) is side-effect free. (basic_props:sideff/2)

Usage: var(X)

- Description: X is a free variable.

nonvar/1: PROPERTY

General properties: nonvar(X)

- The following properties hold globally:
  - X is not further instantiated. (basic_props:not_further_inst/2)
  - nonvar(X) is side-effect free. (basic_props:sideff/2)

Usage: nonvar(X)

- Description: X is currently a term which is not a free variable.
atom/1:

General properties: atom(X)

- The following properties hold upon exit:
  X is an atom. (basic_props:atm/1)

atom(X)

- The following properties hold globally:
  X is not further instantiated. (basic_props:not_further_inst/2)
  atom(X) is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage: atom(X)

- Description: X is currently instantiated to an atom.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

integer/1:

General properties: integer(X)

- The following properties hold upon exit:
  X is an integer. (basic_props:int/1)

integer(X)

- The following properties hold globally:
  X is not further instantiated. (basic_props:not_further_inst/2)
  integer(X) is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage: integer(X)

- Description: X is currently instantiated to an integer.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

float/1:

General properties: float(X)

- The following properties hold upon exit:
  X is a float. (basic_props:flt/1)

float(X)

- The following properties hold globally:
  X is not further instantiated. (basic_props:not_further_inst/2)
  float(X) is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage: float(X)

- Description: X is currently instantiated to a float.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)
PROPERTY \textbf{number/1:}

\textbf{General properties:} \texttt{number(X)}

- \textit{The following properties hold upon exit:}
  \begin{itemize}
  \item $X$ is a number. (\texttt{basic_props:num/1})
  \end{itemize}

\texttt{number(X)}

- \textit{The following properties hold globally:}
  \begin{itemize}
  \item $X$ is not further instantiated. (\texttt{basic_props:not\_further\_inst/2})
  \item \texttt{number(X)} is side-effect free. (\texttt{basic_props:sideeff/2})
  \item This predicate is understood natively by CiaoPP. (\texttt{basic_props:native/1})
  \end{itemize}

\textbf{Usage:} \texttt{number(X)}

- \textit{Description:} $X$ is currently instantiated to a number.
- \textit{The following properties hold globally:}
  \begin{itemize}
  \item This predicate is understood natively by CiaoPP. (\texttt{basic_props:native/1})
  \end{itemize}

PROPERTY \textbf{atomic/1:}

\textbf{General properties:} \texttt{atomic(X)}

- \textit{The following properties hold globally:}
  \begin{itemize}
  \item $X$ is not further instantiated. (\texttt{basic_props:not\_further\_inst/2})
  \item \texttt{atomic(X)} is side-effect free. (\texttt{basic_props:sideeff/2})
  \item This predicate is understood natively by CiaoPP. (\texttt{basic_props:native/1})
  \end{itemize}

\textbf{Usage:} \texttt{atomic(X)}

- \textit{Description:} $X$ is currently instantiated to an atom or a number.
- \textit{The following properties hold globally:}
  \begin{itemize}
  \item This predicate is understood natively by CiaoPP. (\texttt{basic_props:native/1})
  \end{itemize}

PROPERTY \textbf{ground/1:}

\textbf{General properties:} \texttt{ground(X)}

- \textit{The following properties hold upon exit:}
  \begin{itemize}
  \item $X$ is ground. (\texttt{basic_props:gnd/1})
  \end{itemize}

\texttt{ground(X)}

- \textit{The following properties hold globally:}
  \begin{itemize}
  \item $X$ is not further instantiated. (\texttt{basic_props:not\_further\_inst/2})
  \item \texttt{ground(X)} is side-effect free. (\texttt{basic_props:sideeff/2})
  \item This predicate is understood natively by CiaoPP. (\texttt{basic_props:native/1})
  \end{itemize}

\textbf{Usage:} \texttt{ground(X)}

- \textit{Description:} $X$ is currently ground (it contains no variables).
- \textit{The following properties hold globally:}
  \begin{itemize}
  \item This predicate is understood natively by CiaoPP. (\texttt{basic_props:native/1})
  \end{itemize}
type/2:

General properties: type(X, Y)

- The following properties hold upon exit:
  - Y is an atom. (basic_props:atm/1)

- The following properties hold globally:
  - type(Arg1,Arg2) is side-effect free. (basic_props:sideff/2)
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage: type(X, Y)

- Description: X is internally of type Y (var, attv, float, integer, structure, atom or list).

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)
17 Basic term manipulation

Author(s): Daniel Cabeza, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#170 (2004/2/3, 21:41:49 CET)
This module provides basic term manipulation.

17.1 Usage and interface (term_basic)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    =/2, arg/3, functor/3, =../2, copy_term/2, C/3.

17.2 Documentation on exports (term_basic)

=:/2:
Usage: X = Y
  - Description: X and Y unify.
  - The following properties hold globally:
    X = Y is side-effect free.
    This predicate is understood natively by CiaoPP.
    basic_props:eval(X=Y)

arg/3:
Usage: arg(ArgNo, Term, Arg)
  - Description: Argument ArgNo of the term Term is Arg.
  - The following properties should hold at call time:
    ArgNo is currently a term which is not a free variable.
    Term is currently a term which is not a free variable.
    ArgNo is an integer.
  - The following properties hold globally:
    arg(ArgNo,Term,Arg) is side-effect free.
    This predicate is understood natively by CiaoPP.
    basic_props:eval(arg(ArgNo,Term,Arg))

functor/3:
Usage 1: functor(Term, Name, Arity)
  - Description: The principal functor of the term Term has name Name and arity Arity.
The following properties hold upon exit:
- Name is an atom. (basic_props:atm/1)
- Arity is a number. (basic_props:num/1)

The following properties hold globally:
- functor(Term,Name,Arity) is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 2: functor(Term, Name, Arity)
- The following properties should hold at call time:
  Term is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold globally:
  basic_props:eval(functor(Term,Name,Arity)) (basic_props:eval/1)

Usage 3: functor(Term, Name, Arity)
- The following properties should hold at call time:
  Name is currently a term which is not a free variable. (term_typing:nonvar/1)
  Arity is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold globally:
  basic_props:eval(functor(Term,Name,Arity)) (basic_props:eval/1)

========
Usage 1: Term =.. List
- Description: The functor and arguments of the term Term comprise the list List.
- The following properties hold upon exit:
  List is a list. (basic_props:list/1)
- The following properties hold globally:
  Term=..List is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 2: Term =.. List
- The following properties should hold at call time:
  Term is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold globally:
  basic_props:eval(Term=..List) (basic_props:eval/1)

Usage 3: Arg1 =.. List
- The following properties should hold at call time:
  List is a list. (basic_props:list/1)
  term_basic:atom_head(List) (term_basic:atom_head/1)
- The following properties hold globally:
  basic_props:eval(Arg1=..List) (basic_props:eval/1)

copy_term/2:
Usage: copy_term(Term, Copy)
Description: Copy is a renaming of Term, such that brand new variables have been substituted for all variables in Term. If any of the variables of Term have attributes, the copied variables will have copies of the attributes as well. It behaves as if defined by:

```prolog
:- data 'copy of'/1.

copy_term(X, Y) :-
    asserta_fact('copy of'(X)),
    retract_fact('copy of'(Y)).
```

The following properties hold globally:

- `copy_term(Term, Copy)` is side-effect free. (basic_props:sideff/2)
- This predicate is understood natively by CiaoPP. (basic_props:native/1)

C/3:

Usage: `C(S1, Terminal, S2)`

Description: S1 is connected by the terminal Terminal to S2. Internally used in DCG grammar rules. Defined as if by the single clause: `'C'(X|S], X, S).

The following properties hold globally:

- `C(S1, Terminal, S2)` is side-effect free. (basic_props:sideff/2)
- This predicate is understood natively by CiaoPP. (basic_props:native/1)
Chapter 18: Comparing terms

Author(s): Daniel Cabeza, Manuel Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#171 (2004/2/3, 21:45:8 CET)

These built-in predicates are extra-logical. They treat uninstantiated variables as objects with values which may be compared, and they never instantiate those variables. They should not be used when what you really want is arithmetic comparison or unification.

The predicates make reference to a standard total ordering of terms, which is as follows:

- Variables, by age (roughly, oldest first – the order is not related to the names of variables).
- Floats, in numeric order (e.g. -1.0 is put before 1.0).
- Integers, in numeric order (e.g. -1 is put before 1).
- Atoms, in alphabetical (i.e. character code) order.
- Compound terms, ordered first by arity, then by the name of the principal functor, then by the arguments in left-to-right order. Recall that lists are equivalent to compound terms with principal functor `/2.`

For example, here is a list of terms in standard order:

```
[ X, -1.0, -9, 1, bar, foo, [1], X = Y, foo(0,2), bar(1,1,1) ]
```

18.1 Usage and interface (term_compare)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - Predicates: 
    \=/2, \</2, \=@</2, \>=/2, \>/2, compare/3.
  - Properties: 
    ==/2.

18.2 Documentation on exports (term_compare)

==/2:

**PROPERTY**

General properties: Term1 == Term2

- The following properties hold globally:
  
  Term1 is not further instantiated. (basic_props:not_further_inst/2)
  Term2 is not further instantiated. (basic_props:not_further_inst/2)
  Term1==Term2 is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage: Term1 == Term2

- Description: The terms Term1 and Term2 are strictly identical.
\texttt{\textbackslash ==/2:} \hspace{1cm} \textbf{PREDICATE}

\textbf{Usage:} \texttt{T}erm1 \texttt{\textbackslash ==} \texttt{T}erm2

\textbf{Description:} The terms \texttt{T}erm1 and \texttt{T}erm2 are not strictly identical.

\textbf{The following properties hold globally:}

- \texttt{T}erm1 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{T}erm2 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{T}erm1\texttt{\textbackslash ==}\texttt{T}erm2 is side-effect free. \hspace{1cm} (\texttt{basic\_props:sideff/2})
- This predicate is understood natively by CiaoPP. \hspace{1cm} (\texttt{basic\_props:native/1})

\texttt{\textbackslash @</2:} \hspace{1cm} \textbf{PREDICATE}

\textbf{Usage:} \texttt{\textbackslash @<}(\texttt{T}erm1, \texttt{T}erm2)

\textbf{Description:} The term \texttt{T}erm1 precedes the term \texttt{T}erm2 in the standard order.

\textbf{The following properties hold globally:}

- \texttt{T}erm1 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{T}erm2 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{\textbackslash @<}(\texttt{T}erm1,\texttt{T}erm2) is side-effect free. \hspace{1cm} (\texttt{basic\_props:sideff/2})
- This predicate is understood natively by CiaoPP. \hspace{1cm} (\texttt{basic\_props:native/1})

\texttt{\textbackslash @=\textbackslash </2:} \hspace{1cm} \textbf{PREDICATE}

\textbf{Usage:} \texttt{\textbackslash @=\textbackslash <}(\texttt{T}erm1, \texttt{T}erm2)

\textbf{Description:} The term \texttt{T}erm1 precedes or is identical to the term \texttt{T}erm2 in the standard order.

\textbf{The following properties hold globally:}

- \texttt{T}erm1 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{T}erm2 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{\textbackslash @=\textbackslash <}(\texttt{T}erm1,\texttt{T}erm2) is side-effect free. \hspace{1cm} (\texttt{basic\_props:sideff/2})
- This predicate is understood natively by CiaoPP. \hspace{1cm} (\texttt{basic\_props:native/1})

\texttt{\textbackslash @>/2:} \hspace{1cm} \textbf{PREDICATE}

\textbf{Usage:} \texttt{\textbackslash @>}(\texttt{T}erm1, \texttt{T}erm2)

\textbf{Description:} The term \texttt{T}erm1 follows the term \texttt{T}erm2 in the standard order.

\textbf{The following properties hold globally:}

- \texttt{T}erm1 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{T}erm2 is not further instantiated. \hspace{1cm} (\texttt{basic\_props:not\_further\_inst/2})
- \texttt{\textbackslash @>}(\texttt{T}erm1,\texttt{T}erm2) is side-effect free. \hspace{1cm} (\texttt{basic\_props:sideff/2})
- This predicate is understood natively by CiaoPP. \hspace{1cm} (\texttt{basic\_props:native/1})

\texttt{\textbackslash @>=/2:} \hspace{1cm} \textbf{PREDICATE}

\textbf{Usage:} \texttt{\textbackslash @>=}(\texttt{T}erm1, \texttt{T}erm2)

\textbf{Description:} The term \texttt{T}erm1 follows or is identical to the term \texttt{T}erm2 in the standard order.
The following properties hold globally:

- **Term1** is not further instantiated. (basic_props:not_further_inst/2)
- **Term2** is not further instantiated. (basic_props:not_further_inst/2)
- `@>=(Term1,Term2)` is side-effect free. (basic_props:sideff/2)
- This predicate is understood natively by CiaoPP. (basic_props:native/1)

**compare/3:**

```
compare(Op, Term1, Term2)
```

Op is the result of comparing the terms **Term1** and **Term2**.

**Usage:**

- *Calls should, and exit will be compatible with:*
  - Op is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*
  - **Term1** is any term. (basic_props:term/1)
  - **Term2** is any term. (basic_props:term/1)
- *The following properties hold upon exit:*
  - Op is an atom. (basic_props:atm/1)
  - **Term1** is any term. (basic_props:term/1)
  - **Term2** is any term. (basic_props:term/1)
  - Op is an element of `=[,>,<]`. (basic_props:member/2)
  - **Term1** is any term. (basic_props:term/1)
  - **Term2** is any term. (basic_props:term/1)
- *The following properties hold globally:*
  - **Term1** is not further instantiated. (basic_props:not_further_inst/2)
  - **Term2** is not further instantiated. (basic_props:not_further_inst/2)
  - `compare(Op,Term1,Term2)` is side-effect free. (basic_props:sideff/2)
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)
Chapter 19: Basic predicates handling names of constants

19 Basic predicates handling names of constants

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#169 (2004/2/3, 21:36:40 CET)

The Ciao system provides built-in predicates which allow dealing with names of constants (atoms or numbers). Note that sometimes strings (character code lists) are more suitable to handle sequences of characters.

19.1 Usage and interface (atomic_basic)

- Library usage:
  These predicates are built-in in Ciao, so nothing special has to be done to use them.

- Exports:
  - Predicates:
    name/2, atom_codes/2, number_codes/2, number_codes/3, atom_number/2, atom_length/2, atom_concat/3, sub_atom/4.

19.2 Documentation on exports (atomic_basic)

name/2: 

name(Const, String)

String is the list of the ASCII codes of the characters comprising the name of Const. Note that if Const is an atom whose name can be interpreted as a number (e.g. '96'), the predicate is not reversible, as that atom will not be constructed when Const is un-instantiated. Thus it is recommended that new programs use the ISO-compliant predicates atom_codes/2 or number_codes/2, as these predicates do not have this inconsistency.

General properties:
- The following properties hold globally:
  name(Const, String) is side-effect free. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 1:
- Calls should, and exit will be compatible with:
  String is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold at call time:
  Const is an atomic term (an atom or a number). (basic_props:constant/1)
- The following properties hold upon exit:
  String is a string (a list of character codes). (basic_props:string/1)
- The following properties hold globally:
  basic_props:eval(name(Const, String)) (basic_props:eval/1)

Usage 2:
- Description: If String can be interpreted as a number, Const is unified with that number, otherwise with the atom whose name is String.
The following properties should hold at call time:

- Const is a free variable. (term_typing:var/1)
- String is a string (a list of character codes). (basic_props:string/1)

The following properties hold upon exit:

- Const is an atomic term (an atom or a number). (basic_props:constant/1)
- The following properties hold globally:
  - basic_props:eval(name(Const,String)) (basic_props:eval/1)

atom_codes/2:

atom_codes(Atom, String)

String is the list of the ASCII codes of the characters comprising the name of Atom.

General properties:

- The following properties hold globally:
  - atom_codes(Atom,String) is side-effect free. (basic_props:sideff/2)
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 1:

- Calls should, and exit will be compatible with:
  - String is a string (a list of character codes). (basic_props:string/1)
  - The following properties should hold at call time:
    - Atom is an atom. (basic_props:atm/1)
  - The following properties hold upon exit:
    - String is a string (a list of character codes). (basic_props:string/1)
  - The following properties hold globally:
    - basic_props:eval(atom_codes(Atom,String)) (basic_props:eval/1)

Usage 2:

- The following properties should hold at call time:
  - Atom is a free variable. (term_typing:var/1)
  - String is a string (a list of character codes). (basic_props:string/1)
  - The following properties hold upon exit:
    - Atom is an atom. (basic_props:atm/1)
  - The following properties hold globally:
    - basic_props:eval(atom_codes(Atom,String)) (basic_props:eval/1)

number_codes/2:

number_codes(Number, String)

String is the list of the ASCII codes of the characters comprising a representation of Number.

General properties:

- The following properties hold globally:
  - number_codes(Number,String) is side-effect free. (basic_props:sideff/2)
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)
Usage 1:
- Calls should, and exit will be compatible with:
  - String is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold at call time:
  - Number is a number. (basic_props:num/1)
- The following properties hold upon exit:
  - String is a string (a list of character codes). (basic_props:string/1)
- The following properties hold globally:
  - basic_props:eval(number_codes(Number,String)) (basic_props:eval/1)

Usage 2:
- The following properties should hold at call time:
  - Number is a free variable. (term_typing:var/1)
  - String is a string (a list of character codes). (basic_props:string/1)
- The following properties hold upon exit:
  - Number is a number. (basic_props:num/1)
- The following properties hold globally:
  - basic_props:eval(number_codes(Number,String)) (basic_props:eval/1)

**number_codes/3:**

`number_codes(Number, String, Base)`

String is the list of the ASCII codes of the characters comprising a representation of Number in base Base.

General properties:
- The following properties hold globally:
  - `number_codes(Number, String, Base)` is side-effect free. (basic_props:sideff/2)
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 1:
- Calls should, and exit will be compatible with:
  - Base is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold at call time:
  - Number is a number. (basic_props:num/1)
  - String is an integer. (basic_props:int/1)
- The following properties hold upon exit:
  - Base is a string (a list of character codes). (basic_props:string/1)
- The following properties hold globally:
  - basic_props:eval(number_codes(Number,String,Base)) (basic_props:eval/1)

Usage 2:
- The following properties should hold at call time:
  - Number is a free variable. (term_typing:var/1)
  - String is an integer. (basic_props:int/1)
  - Base is a string (a list of character codes). (basic_props:string/1)
- The following properties hold upon exit:
  - Number is a number. (basic_props:num/1)
– The following properties hold globally:
  basic_props:eval(number_codes(Number,String,Base)) (basic_props:eval/1)

atom_number/2:
  atom_number(Atom, Number)

  Atom can be read as a representation of Number.

  General properties:
  – The following properties hold globally:
    atom_number(Atom,Number) is side-effect free. (basic_props:sideff/2)
    This predicate is understood natively by CiaoPP. (basic_props:native/1)

  Usage 1:
  – Calls should, and exit will be compatible with:
    Number is a number. (basic_props:num/1)
  – The following properties should hold at call time:
    Atom is an atom. (basic_props:atm/1)
  – The following properties hold upon exit:
    Number is a number. (basic_props:num/1)
  – The following properties hold globally:
    basic_props:eval(atom_number(Atom,Number)) (basic_props:eval/1)

Usage 2:
  – The following properties should hold at call time:
    Atom is a free variable. (term_typing:var/1)
    Number is a number. (basic_props:num/1)
  – The following properties hold upon exit:
    Atom is an atom. (basic_props:atm/1)
  – The following properties hold globally:
    basic_props:eval(atom_number(Atom,Number)) (basic_props:eval/1)

atom_length/2:
  atom_length(Atom, Length)

  Length is the number of characters forming the name of Atom.

  General properties:
  – The following properties hold globally:
    atom_length(Atom,Length) is side-effect free. (basic_props:sideff/2)
    This predicate is understood natively by CiaoPP. (basic_props:native/1)

  Usage:
  – Calls should, and exit will be compatible with:
    Length is an integer. (basic_props:int/1)
  – The following properties should hold at call time:
    Atom is an atom. (basic_props:atm/1)
  – The following properties hold upon exit:
    Length is an integer. (basic_props:int/1)
The following properties hold globally:
```
basic_props:eval(atom_length(Atom,Length))
```

**atom_concat/3:**
```
atom_concat(Atom_1, Atom_2, Atom_12)
```

Atom_12 is the result of concatenating Atom_1 followed by Atom_2.

**Usage 1:**
- **Description:** Concatenate two atoms.
- **Calls should, and exit will be compatible with:**
  - Atom_12 is currently instantiated to an atom.
  - Atom_1 is currently instantiated to an atom.
  - Atom_2 is currently instantiated to an atom.
- **The following properties hold globally:**
  - basic_props:eval(atom_concat(Atom_1, Atom_2, Atom_12))

**Usage 2:**
- **Description:** Non-deterministically split an atom.
- **The following properties should hold at call time:**
  - Atom_1 is a free variable.
  - Atom_2 is a free variable.
  - Atom_12 is currently instantiated to an atom.
- **The following properties hold globally:**
  - basic_props:eval(atom_concat(Atom_1, Atom_2, Atom_12))

**Usage 3:**
- **Description:** Take out of an atom a certain suffix (or fail if it cannot be done).
- **The following properties should hold at call time:**
  - Atom_1 is a free variable.
  - Atom_2 is currently instantiated to an atom.
  - Atom_12 is currently instantiated to an atom.
- **The following properties hold globally:**
  - Atom_1 is currently instantiated to an atom.
The following properties hold globally:
\[
\text{basic_props:eval(\text{atom_concat(Atom}_1,\text{Atom}_2,\text{Atom}_{12}))} \quad \text{(basic_props:eval/1)}
\]

Usage 4:
- **Description:** Take out of an atom a certain prefix (or fail if it cannot be done).
- **The following properties should hold at call time:**
  \[
  \text{Atom}_1 \text{ is currently instantiated to an atom.} \quad \text{(term_typing:atom/1)}
  \]
  \[
  \text{Atom}_2 \text{ is a free variable.} \quad \text{(term_typing:var/1)}
  \]
  \[
  \text{Atom}_{12} \text{ is currently instantiated to an atom.} \quad \text{(term_typing:atom/1)}
  \]
- **The following properties hold upon exit:**
  \[
  \text{Atom}_2 \text{ is currently instantiated to an atom.} \quad \text{(term_typing:atom/1)}
  \]
- **The following properties hold globally:**
  \[
  \text{basic_props:eval(\text{atom_concat(Atom}_1,\text{Atom}_2,\text{Atom}_{12}))} \quad \text{(basic_props:eval/1)}
  \]

**sub_atom/4:**

\[
\text{sub_atom(Atom, Before, Length, Sub_atom)}
\]

*Sub_atom* is formed with *Length* consecutive characters of *Atom* after the *Before* character. For example, the goal *sub_atom(summer,1,4,umme)* succeeds.

**General properties:**
- **The following properties hold globally:**
  \[
  \text{sub_atom(Atom,Before,Length,Sub_atom) is side-effect free.} \quad \text{(basic_props:sideff/2)}
  \]
  This predicate is understood natively by CiaoPP. \text{(basic_props:native/1)}

**Usage:**
- **Calls should, and exit will be compatible with:**
  \[
  \text{Sub_atom is an atom.} \quad \text{(basic_props:atm/1)}
  \]
- **The following properties should hold at call time:**
  \[
  \text{Atom is an atom.} \quad \text{(basic_props:atm/1)}
  \]
  \[
  \text{Before is an integer.} \quad \text{(basic_props:int/1)}
  \]
  \[
  \text{Length is an integer.} \quad \text{(basic_props:int/1)}
  \]
- **The following properties hold upon exit:**
  \[
  \text{Sub_atom is an atom.} \quad \text{(basic_props:atm/1)}
  \]
- **The following properties hold globally:**
  \[
  \text{basic_props:eval(sub_atom(Atom,Before,Length,Sub_atom))} \quad \text{(basic_props:eval/1)}
  \]
20 Arithmetic

Author(s): Daniel Cabeza, Manuel Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#168 (2004/2/3, 21:28:52 CET)

Arithmetic is performed by built-in predicates which take as arguments arithmetic expressions (see arithexpression/1) and evaluate them. Terms representing arithmetic expressions can be created dynamically, but at the time of evaluation, each variable in an arithmetic expression must be bound to a non-variable expression (the term must be ground). For example, given the code in the first line a possible shell interaction follows:

\[
\text{evaluate}(\text{Expression}, \text{Answer}) :\text{ Answer is Expression.}
\]

?- _X=24*9, evaluate(_X+6, Ans).

\[
\text{Ans} = 222 \ ?
\]

yes

20.1 Usage and interface (arithmetic)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    - is/2, \\
    - =/2, =:=/2, =
  - Regular Types:
    - arithexpression/1.

20.2 Documentation on exports (arithmetic)

is/2:

\[
\text{Val is Exp}
\]

The arithmetic expression \( \text{Exp} \) is evaluated and the result is unified with \( \text{Val} \).

General properties:

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)
  - Val is Exp is side-effect free. (basic_props:sideff/2)

Usage: X is Exp

- The following properties should hold at call time:
  - Exp is an arithmetic expression. (arithmetic:arithexpression/1)
- The following properties hold upon exit:
  - X is a number. (basic_props:num/1)
- The following properties hold globally:
  - basic_props:eval(X is Exp) (basic_props:eval/1)
\(</2\):

\(\text{Exp1} < \text{Exp2}\)

The numeric value of \(\text{Exp1}\) is less than the numeric value of \(\text{Exp2}\) when both are evaluated as arithmetic expressions.

**General properties:**

- *The following properties hold globally:*
  
  This predicate is understood natively by CiaoPP.  
  \(\text{Exp1}<\text{Exp2}\) is side-effect *free*. 

**Usage:**

- *The following properties should hold at call time:*
  
  \(\text{Exp1}\) is an arithmetic expression. 
  \(\text{Exp2}\) is an arithmetic expression. 

- *The following properties hold globally:*
  
  \(\text{basic_props:eval(Exp1<Exp2)}\)

\(\leq/2\):

\(\text{Exp1} \leq \text{Exp2}\)

The numeric value of \(\text{Exp1}\) is less than or equal to the numeric value of \(\text{Exp2}\) when both are evaluated as arithmetic expressions.

**General properties:**

- *The following properties hold globally:*
  
  This predicate is understood natively by CiaoPP.  
  \(\text{Exp1}\leq\text{Exp2}\) is side-effect *free*. 

**Usage:**

- *The following properties should hold at call time:*
  
  \(\text{Exp1}\) is an arithmetic expression. 
  \(\text{Exp2}\) is an arithmetic expression. 

- *The following properties hold globally:*
  
  \(\text{basic_props:eval(Exp1<Exp2)}\)

\(>/2\):

\(\text{Exp1} > \text{Exp2}\)

The numeric value of \(\text{Exp1}\) is greater than the numeric value of \(\text{Exp2}\) when both are evaluated as arithmetic expressions.

**General properties:**

- *The following properties hold globally:*
  
  This predicate is understood natively by CiaoPP.  
  \(\text{Exp1}>\text{Exp2}\) is side-effect *free*. 

**Usage:**

- *The following properties should hold at call time:*
  
  \(\text{Exp1}\) is an arithmetic expression. 
  \(\text{Exp2}\) is an arithmetic expression. 

- *The following properties hold globally:*
  
  \(\text{basic_props:eval(Exp1<Exp2)}\)
Chapter 20: Arithmetic

\[
\geq/2:
\]

\begin{align*}
\text{Exp1} &\geq \text{Exp2} \\
\end{align*}

The numeric value of \text{Exp1} is greater than or equal to the numeric value of \text{Exp2} when both are evaluated as arithmetic expressions.

**General properties:**
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. \((\text{basic\_props:native/1})\)
  - \text{Exp1} \geq \text{Exp2} is side-effect free. \((\text{basic\_props:sideff/2})\)

**Usage:**
- *The following properties should hold at call time:*
  - \text{Exp1} is an arithmetic expression. \((\text{arithmetic:arithexpression/1})\)
  - \text{Exp2} is an arithmetic expression. \((\text{arithmetic:arithexpression/1})\)

  *The following properties hold globally:*
  - \text{basic\_props:eval(Exp1}\geq\text{Exp2)} \((\text{basic\_props:eval/1})\)

\[
=:=/2:
\]

\begin{align*}
\text{Exp1} &\=: \text{Exp2} \\
\end{align*}

The numeric values of \text{Exp1} and \text{Exp2} are equal when both are evaluated as arithmetic expressions.

**General properties:**
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. \((\text{basic\_props:native/1})\)
  - \text{Exp1} = \text{Exp2} is side-effect free. \((\text{basic\_props:sideff/2})\)

**Usage:**
- *The following properties should hold at call time:*
  - \text{Exp1} is an arithmetic expression. \((\text{arithmetic:arithexpression/1})\)
  - \text{Exp2} is an arithmetic expression. \((\text{arithmetic:arithexpression/1})\)

  *The following properties hold globally:*
  - \text{basic\_props:eval(Exp1}=:\text{Exp2)} \((\text{basic\_props:eval/1})\)

\[
=\neq/2:
\]

\begin{align*}
\text{Exp1} &\neq \text{Exp2} \\
\end{align*}

The numeric values of \text{Exp1} and \text{Exp2} are not equal when both are evaluated as arithmetic expressions.

**General properties:**
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. \((\text{basic\_props:native/1})\)
  - \text{Exp1} \neq \text{Exp2} is side-effect free. \((\text{basic\_props:sideff/2})\)

**Usage:**
- *The following properties should hold at call time:*
  - \text{Exp1} is an arithmetic expression. \((\text{arithmetic:arithexpression/1})\)
  - \text{Exp2} is an arithmetic expression. \((\text{arithmetic:arithexpression/1})\)

  *The following properties hold globally:*
  - \text{basic\_props:eval(Exp1}\neq\text{Exp2)} \((\text{basic\_props:eval/1})\)
An arithmetic expression is a term built from numbers and evaluable functors that represent arithmetic functions. An arithmetic expression evaluates to a number, which may be an integer (\texttt{int/1}) or a float (\texttt{flt/1}). The evaluable functors allowed in an arithmetic expression are listed below, together with an indication of the functions they represent.

All evaluable functors defined in ISO-Prolog are implemented, as well as some other useful or traditional. Unless stated otherwise, an expression evaluates to a float if any of its arguments is a float, otherwise to an integer.

- \texttt{-/1}: sign reversal.
- \texttt{+/1}: identity.
- \texttt{--/1}: decrement by one.
- \texttt{++/1}: increment by one.
- \texttt{+/2}: addition.
- \texttt{-/2}: subtraction.
- \texttt{*/2}: multiplication.
- \texttt{///2}: integer division. Float arguments are truncated to integers, result always integer.
- \texttt{%/2}: division. Result always float.
- \texttt{rem/2}: integer remainder. The result is always an integer, its sign is the sign of the first argument.
- \texttt{mod/2}: modulo. The result is always a positive integer.
- \texttt{abs/1}: absolute value.
- \texttt{sign/1}: sign of.
- \texttt{float_integer_part/1}: float integer part. Result always float.
- \texttt{float_fractional_part/1}: float fractional part. Result always float.
- \texttt{truncate/1}: The result is the integer equal to the integer part of the argument.
- \texttt{integer/1}: same as \texttt{truncate/1}.
- \texttt{float/1}: conversion to float.
- \texttt{floor/1}: largest integer not greater than.
- \texttt{round/1}: integer nearest to.
- \texttt{ceiling/1}: smallest integer not smaller than.
- \texttt{**/2}: exponentiation. Result always float.
- \texttt{>>/2}: integer bitwise right shift.
- \texttt{<</2}: integer bitwise left shift.
- \texttt{\!/2}: integer bitwise and.
- \texttt{\!/2}: integer bitwise or.
- \texttt{\!/1}: integer bitwise complement.
- \texttt{#/2}: integer bitwise exclusive or (xor).
- \texttt{exp/1}: exponential (\(e\) to the power of). Result always float.
- \texttt{log/1}: natural logarithm (base \(e\)). Result always float.
- \texttt{sqrt/1}: square root. Result always float.
- \texttt{sin/1}: sine. Result always float.
- \texttt{cos/1}: cosine. Result always float.
- \texttt{atan/1}: arc tangent. Result always float.
• \texttt{gcd/2}: Greatest common divisor. Arguments must evaluate to integers, result always integer.

In addition to these functors, a list of just a number evaluates to this number. Since a quoted string is just a list of integers, this allows a quoted character to be used in place of its ASCII code; e.g. "A" behaves within arithmetic expressions as the integer 65. Note that this is not ISO-compliant, and that can be achieved by using the ISO notation \texttt{0'A}.

Arithmetic expressions, as described above, are just data structures. If you want one evaluated you must pass it as an argument to one of the arithmetic predicates defined in this library.

**General properties:**

- The following properties hold globally:
  \[
  \text{arithexpression}(\text{Arg1}) \text{ is side-effect free.} \\
  \text{(basic_props:sideff/2)}
  \]

**Usage:** \texttt{arithexpression(E)}

- \textit{Description:} \(E\) is an arithmetic expression.
21 Basic file/stream handling

Author(s): Daniel Cabeza, Mats Carlsson.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#125 (2003/12/29, 18:52:30 CET)

This module provides basic predicates for handling files and streams, in order to make input/output on them.

21.1 Usage and interface (streams_basic)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - **Predicates:**
    - open/3, open/4, close/1, set_input/1, current_input/1, set_output/1, current_output/1, character_count/2, line_count/2, line_position/2, flush_output/1, flush_output/0, clearerr/1, current_stream/3, stream_code/2, absolute_file_name/2, absolute_file_name/7.
  - **Regular Types:**
    - open_option_list/1, sourcename/1, stream/1, stream_alias/1, io_mode/1.
  - **Multifiles:**
    - file_search_path/2, library_directory/1.

21.2 Documentation on exports (streams_basic)

**open/3:**

```prolog
open(File, Mode, Stream)
```

Open **File** with mode **Mode** and return in **Stream** the stream associated with the file. No extension is implicit in **File**.

**General properties:**

- The following properties hold globally:
  This predicate is understood natively by CiaoPP.

**Usage 1:**

- **Description:** Normal use.
- **Calls should, and exit will be compatible with:**
  - **Stream** is an open stream.
  - **File** is a source name.

**Usage 2:**

- **The following properties should hold at call time:**
  - **Mode** is an opening mode (‘read’, ‘write’ or ‘append’).
  - **Stream** is an open stream.
- **Description:** In the special case that `File` is an integer, it is assumed to be a file descriptor passed to Prolog from a foreign function call. The file descriptor is connected to a Prolog stream (invoking the UNIX function `fopen`) which is unified with `Stream`.

- **Calls should, and exit will be compatible with:**

  - `Stream` is an open stream. (streams_basic:stream/1)

- **The following properties should hold at call time:**

  - `File` is an integer. (basic_props:int/1)
  - `Mode` is an opening mode (‘read’, ‘write’ or ’append’). (streams_basic:io_mode/1)

- **The following properties hold upon exit:**

  - `Stream` is an open stream. (streams_basic:stream/1)

---

**open/4:**

 Predicate

```prolog
open(File, Mode, Stream, Options)
```

Same as `open(File, Mode, Stream)` with options `Options`. See the definition of `open_option_list/1` for details.

**Usage:**

- **Calls should, and exit will be compatible with:**

  - `Stream` is an open stream. (streams_basic:stream/1)

- **The following properties should hold at call time:**

  - `File` is a source name. (streams_basic:sourcename/1)
  - `Mode` is an opening mode (‘read’, ‘write’ or ’append’). (streams_basic:io_mode/1)
  - `Options` is a list of options for `open/4`. (streams_basic:open_option_list/1)

- **The following properties hold upon exit:**

  - `Stream` is an open stream. (streams_basic:stream/1)

- **The following properties hold globally:**

  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

---

**open_option_list/1:**

 REGTYPE

A list of options for `open/4`, currently the meaningful options are:

- `lock` Try to set an advisory lock for the file. If the open mode is `read`, the lock is a read (shared) lock, else it is a write (exclusive) lock. If the lock cannot be acquired, the call waits until it is released (but can fail in exceptional cases).

- `lock_nb` Same as `lock`, but the call immediately fails if the lock cannot be acquired.

- `lock(Lock_Mode)` Same as `lock`, but specifying in `Lock_Mode` whether the lock is `read` (also `shared`) or `write` (also exclusive). This option has been included for compatibility with the SWI-Prolog locking options, because in general the type of lock should match the open mode as in the `lock` option.

- `lock_nb(Lock_Mode)` Same as the previous option but with the `lock_nb` behavior.

All file locking is implemented via the POSIX function `fentl()`. Please refer to its manual page for details.

**Usage:** `open_option_list(L)`

- **Description:** `L` is a list of options for `open/4`. 
Chapter 21: Basic file/stream handling

**close/1:**

`close(Stream)`

Close the stream `Stream`.

**Usage:**

- The following properties should hold at call time:
  Stream is an open stream. (streams_basic:stream/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

**set_input/1:**

`set_input(Stream)`

Set the current input stream to `Stream`. A notion of current input stream is maintained by the system, so that input predicates with no explicit stream operate on the current input stream. Initially it is set to `user_input`.

**Usage:**

- The following properties should hold at call time:
  Stream is an open stream. (streams_basic:stream/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

**current_input/1:**

`current_input(Stream)`

Unify `Stream` with the current input stream. In addition to the ISO behavior, stream aliases are allowed. This is useful for most applications checking whether a stream is the standard input or output.

**Usage:**

- Calls should, and exit will be compatible with:
  Stream is an open stream. (streams_basic:stream/1)
- The following properties hold upon exit:
  Stream is an open stream. (streams_basic:stream/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

**set_output/1:**

`set_output(Stream)`

Set the current output stream to `Stream`. A notion of current output stream is maintained by the system, so that output predicates with no explicit stream operate on the current output stream. Initially it is set to `user_output`.

**Usage:**

- The following properties should hold at call time:
  Stream is an open stream. (streams_basic:stream/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)
**current_output/1:**

`current_output(Stream)`

Unify `Stream` with the current output stream. The same comment as for `current_input/1` applies.

Usage:

- *Calls should, and exit will be compatible with:*
  - `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties hold upon exit:*
  - `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

**character_count/2:**

`character_count(Stream, Count)`

Count characters have been read from or written to `Stream`.

Usage:

- *Calls should, and exit will be compatible with:*
  - `Count` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  - `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties hold upon exit:*
  - `Count` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

**line_count/2:**

`line_count(Stream, Count)`

Count lines have been read from or written to `Stream`.

Usage:

- *Calls should, and exit will be compatible with:*
  - `Count` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  - `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties hold upon exit:*
  - `Count` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

**line_position/2:**

`line_position(Stream, Count)`

Count characters have been read from or written to the current line of `Stream`.

Usage:
– Calls should, and exit will be compatible with:
  Count is an integer. (basic_props:int/1)
– The following properties should hold at call time:
  Stream is an open stream. (streams_basic:stream/1)
– The following properties hold upon exit:
  Count is an integer. (basic_props:int/1)
– The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

flush_output/1:
flush_output(Stream)
Flush any buffered data to output stream Stream.
Usage:
  – The following properties should hold at call time:
    Stream is an open stream. (streams_basic:stream/1)
  – The following properties hold globally:
    This predicate is understood natively by CiaoPP. (basic_props:native/1)

flush_output/0:
flush_output(flush_output
Behaves like current_output(S), flush_output(S)

clearerr/1:
clearerr(Stream)
Clear the end-of-file and error indicators for input stream Stream.
Usage:
  – The following properties should hold at call time:
    Stream is an open stream. (streams_basic:stream/1)

current_stream/3:
current_stream(Filename, Mode, Stream)
Stream is a stream which was opened in mode Mode and which is connected to the absolute file name Filename (an atom) or to the file descriptor Filename (an integer). This predicate can be used for enumerating all currently open streams through backtracking.
General properties:
  – The following properties hold globally:
    This predicate is understood natively by CiaoPP. (basic_props:native/1)
Usage 1:
  – Calls should, and exit will be compatible with:
    Filename is an atom. (basic_props:atm/1)
    Mode is an opening mode (‘read’, ‘write’ or ‘append’). (streams_basic:io_mode/1)
    Stream is an open stream. (streams_basic:stream/1)
- The following properties hold upon exit:
  
  - **Filename** is an atom.  
    (basic_props:atm/1)
  
  - **Mode** is an opening mode ('read', 'write' or 'append').  
    (streams_basic:io_mode/1)
  
  - **Stream** is an open stream.  
    (streams_basic:stream/1)

Usage 2:

- Calls should, and exit will be compatible with:
  
  - **Filename** is an integer.  
    (basic_props:int/1)
  
  - **Mode** is an opening mode ('read', 'write' or 'append').  
    (streams_basic:io_mode/1)
  
  - **Stream** is an open stream.  
    (streams_basic:stream/1)

- The following properties hold upon exit:
  
  - **Filename** is an integer.  
    (basic_props:int/1)
  
  - **Mode** is an opening mode ('read', 'write' or 'append').  
    (streams_basic:io_mode/1)
  
  - **Stream** is an open stream.  
    (streams_basic:stream/1)

**stream_code/2:**

**PREDICATE**

```
stream_code(Stream, StreamCode)
```

**StreamCode** is the file descriptor (an integer) corresponding to the Prolog stream **Stream**.

Usage 1:

- Calls should, and exit will be compatible with:
  
  - **StreamCode** is an integer.  
    (basic_props:int/1)
  
- The following properties should hold at call time:
  
  - **Stream** is an open stream.  
    (streams_basic:stream/1)

- The following properties hold upon exit:
  
  - **StreamCode** is an integer.  
    (basic_props:int/1)

Usage 2:

- The following properties should hold at call time:
  
  - **Stream** is a free variable.  
    (term_typing:var/1)
  
  - **StreamCode** is an integer.  
    (basic_props:int/1)

- The following properties hold upon exit:
  
  - **Stream** is an open stream.  
    (streams_basic:stream/1)

**absolute_file_name/2:**

**PREDICATE**

```
absolute_file_name(RelFileSpec, AbsFileSpec)
```

If **RelFileSpec** is an absolute pathname then do an absolute lookup. If **RelFileSpec** is a relative pathname then prefix the name with the name of the current directory and do an absolute lookup. If **RelFileSpec** is a path alias, perform the lookup following the path alias rules (see **sourcename/1**). In all cases: if a matching file with suffix .pl exists, then **AbsFileSpec** will be unified with this file. Failure to open a file normally causes an exception. The behaviour can be controlled by the **fileerrors** prolog flag.

Usage: absolute_file_name(RelFileSpec, AbsFileSpec)

- **Description**: **AbsFileSpec** is the absolute name (with full path) of **RelFileSpec**.
– Calls should, and exit will be compatible with:
  
  \( \text{RelFileSpec} \) is a source name. \((\text{streams\_basic:sourcename/1})\)
  
  \( \text{AbsFileSpec} \) is an atom. \((\text{basic\_props:atm/1})\)
  
 – The following properties should hold at call time:
  
  \( \text{RelFileSpec} \) is currently a term which is not a free variable. \((\text{term\_typing:nonvar/1})\)
  
  \( \text{AbsFileSpec} \) is a free variable. \((\text{term\_typing:var/1})\)
  
 – The following properties hold globally:
  
  This predicate is understood natively by CiaoPP. \((\text{basic\_props:native/1})\)

\[\text{absolute\_file\_name/7:}\]

\[\text{PREDICATE}\]

\[\text{absolute\_file\_name}(\text{Spec, Opt, Suffix, CurrDir, AbsFile, AbsBase, AbsDir})\]

AbsFile is the absolute name (with full path) of Spec, which has an optional first suffix Opt and an optional second suffix Suffix, when the current directory is CurrDir. AbsBase is the same as AbsFile, but without the second suffix, and AbsDir is the absolute path of the directory where AbsFile is. The Ciao compiler invokes this predicate with \(\text{Opt} = \_\_\text{opt}\) and \(\text{Suffix} = \text{.pl}\) when searching source files.

Usage:

– The following properties should hold at call time:

  \( \text{Spec} \) is a source name. \((\text{streams\_basic:sourcename/1})\)
  
  \( \text{Opt} \) is an atom. \((\text{basic\_props:atm/1})\)
  
  \( \text{Suffix} \) is an atom. \((\text{basic\_props:atm/1})\)
  
  \( \text{CurrDir} \) is an atom. \((\text{basic\_props:atm/1})\)
  
  \( \text{AbsFile} \) is a free variable. \((\text{term\_typing:var/1})\)
  
  \( \text{AbsBase} \) is a free variable. \((\text{term\_typing:var/1})\)
  
  \( \text{AbsDir} \) is a free variable. \((\text{term\_typing:var/1})\)

– The following properties hold upon exit:

  \( \text{AbsFile} \) is an atom. \((\text{basic\_props:atm/1})\)
  
  \( \text{AbsBase} \) is an atom. \((\text{basic\_props:atm/1})\)
  
  \( \text{AbsDir} \) is an atom. \((\text{basic\_props:atm/1})\)

– The following properties hold globally:

  This predicate is understood natively by CiaoPP. \((\text{basic\_props:native/1})\)

\[\text{sourcename/1:}\]

\[\text{REGTYPE}\]

A source name is a flexible way of referring to a concrete file. A source name is either a relative or absolute filename given as:

– an atom, or
  
  – a unary functor (which represents a path alias, see below) applied to a relative path, the latter being given as an atom.

In all cases certain filename extensions (e.g., .pl) can be implicit. In the first form above, file names can be relative to the current directory. Also, file names beginning with ~ or $ are treated specially. For example,
The Ciao Prolog System

'~/ciao/sample.pl'
is equivalent to '/home/staff/herme/ciao/sample.pl', if/home/staff/herme is the user's home directory. (This is also equivalent to '$HOME/ciao/sample.pl' as explained below.)

'bardo/prolog/sample.pl'
is equivalent to '/home/bardo/prolog/sample.pl', if /home/bardo is bardo's home directory.

'$UTIL/sample.pl'
is equivalent to '/usr/local/src/utilities/sample.pl', if /usr/local/src/utilities is the value of the environment variable UTIL.

The second form allows using path aliases. Such aliases allow referring to files not with absolute file system paths but with paths which are relative to predefined (or user-defined) abstract names. For example, given the path alias myutils which has been defined to refer to path '/home/bardo/utilities', if that directory contains the file stuff.pl then the term myutils(stuff) in a use_module/1 declaration would refer to the file '/home/bardo/utilities/stuff.pl' (the .pl extension is implicit in the use_module/1 declaration). As a special case, if that directory contains a subdirectory named stuff which in turn contains the file stuff.pl, the same term would refer to the file '/home/bardo/utilities/stuff/stuff.pl'. If a path alias is related to several paths, all paths are scanned in sequence until a match is found. For information on predefined path aliases or how to define new path aliases, see file_search_path/2.

Usage: sourcename(F)

— Description: F is a source name.

stream/1: REGTYPE

Streams correspond to the file pointers used at the operating system level, and usually represent opened files. There are four special streams which correspond with the operating system standard streams:

user_input
The standard input stream, i.e. the terminal, usually.

user_output
The standard output stream, i.e. the terminal, usually.

user_error
The standard error stream.

user
The standard input or output stream, depending on context.

Usage: stream(S)

— Description: S is an open stream.

stream_alias/1: REGTYPE

Usage: stream_alias(S)

— Description: S is the alias of an open stream, i.e., an atom which represents a stream at Prolog level.
**io_mode/1:**

Can have the following values:

- **read**: Open the file for input.
- **write**: Open the file for output. The file is created if it does not already exist, the file will otherwise be truncated.
- **append**: Open the file for output. The file is created if it does not already exist, the file will otherwise be appended to.

**Usage:** `io_mode(M)`

- **Description:** `M` is an opening mode (‘read’, ‘write’ or ‘append’).

### 21.3 Documentation on multfiles (streams_basic)

**file_search_path/2:**

- `file_search_path(Alias, Path)`

  The path alias `Alias` is linked to path `Path`. Both arguments must be atoms. New facts (or clauses) of this predicate can be asserted to define new path aliases. Predefined path aliases in Ciao are:

  - **library**: Initially points to all Ciao library paths. See `library_directory/1`.
  - **engine**: The path of the Ciao engine builtins.
  - **.**: The current path (`’.’`).

  The predicate is `multifile`.

  The predicate is of type `dynamic`.

**library_directory/1:**

- `library_directory(Path)`

  `Path` is a library path (a path represented by the path alias `library`). Predefined library paths in Ciao are `’$CIAOLIB/lib’`, `’$CIAOLIB/library’`, and `’$CIAOLIB/contrib’`, given that `$CIAOLIB` is the path of the root ciao library directory. More library paths can be defined by asserting new facts (or clauses) of this predicate.

  The predicate is `multifile`.

  The predicate is of type `dynamic`.
22 Basic input/output

Author(s): Daniel Cabeza, Mats Carlsson.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#70 (2003/12/19, 16:49:37 CET)

This module provides predicates for character input/output and for canonical term output. From the ISO-Prolog predicates for character input/output, only the _code versions are provided, the rest are given by library(iso_byte_char), using these. Most predicates are provided in two versions: one that specifies the input or output stream as the first argument and a second which omits this argument and uses the current input or output stream.

22.1 Usage and interface (io_basic)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    - get_code/2, get_code/1, get1_code/2, get1_code/1, peek_code/2, peek_code/1, skip_code/2, skip_code/1, skip_line/1, skip_line/0, put_code/2, put_code/1, nl/1, nl/0, tab/2, tab/1, code_class/2, getct/2, getct1/2, display/2, display/1, displayq/2, displayq/1.

22.2 Documentation on exports (io_basic)

get_code/2:  PREDICATE
get_code(Stream, Code)
Reads from Stream the next character and unifies Code with its character code. At end of stream, unifies Code with the integer -1.
Usage:
- Calls should, and exit will be compatible with:
  Code is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Stream is an open stream. (streams_basic:stream/1)
- The following properties hold upon exit:
  Code is an integer. (basic_props:int/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

get_code/1:  PREDICATE
get_code(Code)
Behaves like current_input(S), get_code(S,Code).
Usage:  ⚫ ISO ⚫
get1_code/2:  
\texttt{get1\_code(Stream, Code)}

Reads from \texttt{Stream} the next non-layout character (see \texttt{code\_class/2}) and unifies \texttt{Code} with its character code. At end of stream, unifies \texttt{Code} with the integer -1.

Usage:
- \textit{Calls should, and exit will be compatible with:}
  \texttt{Code} is an integer.  \hspace{1cm} \texttt{(basic\_props:int/1)}
- \textit{The following properties should hold at call time:}
  \texttt{Stream} is an open stream.  \hspace{1cm} \texttt{(streams\_basic:stream/1)}
- \textit{The following properties hold upon exit:}
  \texttt{Code} is an integer.  \hspace{1cm} \texttt{(basic\_props:int/1)}
- \textit{The following properties hold globally:}
  This predicate is understood natively by CiaoPP.  \hspace{1cm} \texttt{(basic\_props:native/1)}

get1_code/1:
\texttt{get1\_code(Code)}

Behaves like \texttt{current\_input(S)}, \texttt{get1\_code(S,Code)}.

Usage:
- \textit{Calls should, and exit will be compatible with:}
  \texttt{Code} is an integer.  \hspace{1cm} \texttt{(basic\_props:int/1)}
- \textit{The following properties hold upon exit:}
  \texttt{Code} is an integer.  \hspace{1cm} \texttt{(basic\_props:int/1)}
- \textit{The following properties hold globally:}
  This predicate is understood natively by CiaoPP.  \hspace{1cm} \texttt{(basic\_props:native/1)}

peek_code/2:
\texttt{peek\_code(Stream, Code)}

Unifies \texttt{Code} with the character code of the next character of \texttt{Stream}, leaving the stream position unaltered. At end of stream, unifies \texttt{Code} with the integer -1.

Usage:
- \textit{Calls should, and exit will be compatible with:}
  \texttt{Code} is an integer.  \hspace{1cm} \texttt{(basic\_props:int/1)}
- \textit{The following properties should hold at call time:}
  \texttt{Stream} is an open stream.  \hspace{1cm} \texttt{(streams\_basic:stream/1)}
- \textit{The following properties hold upon exit:}
  \texttt{Code} is an integer.  \hspace{1cm} \texttt{(basic\_props:int/1)}
**peek_code/1:**

\[
\text{peek\_code}(\text{Code})
\]

Behaves like \text{current\_input}(S), \text{peek\_code}(S,\text{Code}).

**Usage:**

- **Calls should, and exit will be compatible with:**
  - \text{Code} is an integer.

- **The following properties hold upon exit:**
  - \text{Code} is an integer.

**skip_code/2:**

\[
\text{skip\_code}(\text{Stream, Code})
\]

Skips just past the next character code \text{Code} from \text{Stream}.

**Usage:**

- **The following properties should hold at call time:**
  - \text{Stream} is an open stream.
  - \text{Code} is an integer.

**skip_code/1:**

\[
\text{skip\_code}(\text{Code})
\]

Behaves like \text{current\_input}(S), \text{skip\_code}(S,\text{Code}).

**Usage:**

- **The following properties should hold at call time:**
  - \text{Code} is an integer.

**skip_line/1:**

\[
\text{skip\_line}(\text{Stream})
\]

Skips from \text{Stream} the remaining input characters on the current line. If the end of the stream is reached, the stream will stay at its end. Portable among different operating systems.

**Usage:**

- **The following properties should hold at call time:**
  - \text{Stream} is an open stream.

**skip_line/0:**

\[
\text{skip\_line}()
\]

Behaves like \text{current\_input}(S), \text{skip\_line}(S).
**put_code/2:**

```prolog
put_code(Stream, Code)
```

Outputs to Stream the character corresponding to character code Code.

Usage:

- *The following properties should hold at call time:*
  - Stream is an open stream. *(streams_basic:stream/1)*
  - Code is an integer. *(basic_props:int/1)*

- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. *(basic_props:native/1)*

**put_code/1:**

```prolog
put_code(Code)
```

Behaves like `current_output(S), put_code(S,Code)`.

Usage:

- *The following properties should hold at call time:*
  - Code is an integer. *(basic_props:int/1)*

- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. *(basic_props:native/1)*

**nl/1:**

```prolog
nl(Stream)
```

Outputs a newline character to Stream. Equivalent to `put_code(Stream, 0'
).

Usage:

- *The following properties should hold at call time:*
  - Stream is an open stream. *(streams_basic:stream/1)*

- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. *(basic_props:native/1)*

**nl/0:**

```prolog
nl(nl)
```

Behaves like `current_output(S), nl(S)`.

Usage:

- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. *(basic_props:native/1)*

**tab/2:**

```prolog
tab(Stream, Num)
```

Outputs Num spaces to Stream.

Usage:

- *The following properties should hold at call time:*
  - Stream is an open stream. *(streams_basic:stream/1)*

- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. *(basic_props:native/1)*
The following properties should hold at call time:
Stream is an open stream. (streams_basic:stream/1)
Num is an integer. (basic_props:int/1)

The following properties hold globally:
This predicate is understood natively by CiaoPP. (basic_props:native/1)

### tab/1:

`tab(Num)`

Behaves like `current_output(S), tab(S,Num)`.

**Usage:**

The following properties should hold at call time:
Num is an integer. (basic_props:int/1)

The following properties hold globally:
This predicate is understood natively by CiaoPP. (basic_props:native/1)

### code_class/2:

`code_class(Code, Class)`

Unifies Class with an integer corresponding to the lexical class of the character whose code is Code, with the following correspondence:

- 0 - layout (includes space, newline, tab)
- 1 - small letter
- 2 - capital letter (including '_')
- 3 - digit
- 4 - graphic (includes #$&*+-./:<=>?@^`~ )
- 5 - punctuation (includes !;"%'(),\{\} )

Note that in ISO-Prolog the back quote ` ` is a punctuation character, whereas in Ciao it is a graphic character. Thus, if compatibility with ISO-Prolog is desired, the programmer should not use this character in unquoted names.

**Usage:**

- Calls should, and exit will be compatible with:
  Class is an integer. (basic_props:int/1)

- The following properties should hold at call time:
  Code is an integer. (basic_props:int/1)

- The following properties hold upon exit:
  Class is an integer. (basic_props:int/1)

### getct/2:

`getct(Code, Type)`

Reads from the current input stream the next character, unifying Code with its character code, and Type with its lexical class. At end of stream, unifies both Code and Type with the integer -1. Equivalent to
get(Code), (Code = -1 -> Type = -1 ; code_class(Code, Type))

Usage:
- *Calls should, and exit will be compatible with:*
  - Code is an integer. (basic_props:int/1)
  - Type is an integer. (basic_props:int/1)
- *The following properties hold upon exit:*
  - Code is an integer. (basic_props:int/1)
  - Type is an integer. (basic_props:int/1)

getct1/2:  

getct1(Code, Type)  
Reads from the current input stream the next non-layout character, unifying Code with its character code, and Type with its lexical class (which will be nonzero). At end of stream, unifies both Code and Type with the integer -1. Equivalent to

get1(Code), (Code = -1 -> Type = -1 ; code_class(Code, Type))

Usage:
- *Calls should, and exit will be compatible with:*
  - Code is an integer. (basic_props:int/1)
  - Type is an integer. (basic_props:int/1)
- *The following properties hold upon exit:*
  - Code is an integer. (basic_props:int/1)
  - Type is an integer. (basic_props:int/1)

display/2:  

display(Stream, Term)  
Displays Term onto Stream. Lists are output using list notation, the other compound terms are output in functional notation. Similar to write_term(Stream, Term, [ignore_ops(ops)]), except that curly bracketed notation is not used with \{}/1, and the write_strings flag is not honored.

Usage:
- *The following properties should hold at call time:*
  - Stream is an open stream. (streams_basic:stream/1)
  - Term is any term. (basic_props:term/1)
- *The following properties hold upon exit:*
  - Term is any term. (basic_props:term/1)
- *The following properties hold globally:*
  - Term is not further instantiated. (basic_props:not_further_inst/2)
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)
display/1: PREDICATE

display(Term)

Behaves like current_output(S), display(S,Term).

Usage:
- *The following properties should hold at call time:*
  Term is any term. (basic_props:term/1)
- *The following properties hold upon exit:*
  Term is any term. (basic_props:term/1)
- *The following properties hold globally:*
  Term is not further instantiated. (basic_props:not_further_inst/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

displayq/2: PREDICATE

displayq(Stream, Term)

Similar to display(Stream, Term), but atoms and functors that can't be read back by read_term/3 are quoted. Thus, similar to write_term(Stream, Term, [quoted(true), ignore_ops(ops)]), with the same exceptions as display/2.

Usage:
- *The following properties should hold at call time:*
  Stream is an open stream. (streams_basic:stream/1)
  Term is any term. (basic_props:term/1)
- *The following properties hold upon exit:*
  Term is any term. (basic_props:term/1)
- *The following properties hold globally:*
  Term is not further instantiated. (basic_props:not_further_inst/2)

displayq/1: PREDICATE

displayq(Term)

Behaves like current_output(S), displayq(S,Term).

Usage:
- *The following properties should hold at call time:*
  Term is any term. (basic_props:term/1)
- *The following properties hold upon exit:*
  Term is any term. (basic_props:term/1)
- *The following properties hold globally:*
  Term is not further instantiated. (basic_props:not_further_inst/2)
Chapter 23: Exception handling

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
This module includes predicates related to exceptions, which alter the normal flow of Prolog.

23.1 Usage and interface (exceptions)

- **Library usage:**
  These predicates are built-in in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - **Predicates:**
    catch/3, intercept/3, throw/1, halt/0, halt/1, abort/0.

23.2 Documentation on exports (exceptions)

catch/3:

```
catch(Goal, Error, Handler)
```

Executes Goal. If an exception is raised during its execution, Error is unified with the exception, and if the unification succeeds, the entire execution derived from Goal is aborted, and Handler is executed. The execution resumes with the continuation of the catch/3 call. For example, given the code

```
p(X) :- throw(error), display('---').
p(X) :- display(X).
```

the execution of "catch(p(0), E, display(E)), display(.), fail." results in the output "error."

*Meta-predicate with arguments:* catch(goal,?,goal).

**Usage:**

- *Calls should, and exit will be compatible with:*
  - Error is any term. (basic_props:term/1)
- *The following properties should hold at call time:*
  - Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  - Handler is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties hold upon exit:*
  - Error is any term. (basic_props:term/1)
- *The following properties hold globally:*
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)
intercept/3:
intercept(Goal, Error, Handler)
Executes Goal. If an exception is raised during its execution, Error is unified with the exception, and if the unification succeeds, Handler is executed and then the execution resumes after the predicate which produced the exception. Note the difference with builtin catch/3, given the same code defined there, the execution of "intercept(p(0), E, display(E)), display(.), fail." results in the output "error---.0.".

Meta-predicate with arguments: intercept(goal,?,goal).
Usage:
- Calls should, and exit will be compatible with:
  Error is any term. (basic_props:term/1)
- The following properties should hold at call time:
  Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  Handler is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- The following properties hold upon exit:
  Error is any term. (basic_props:term/1)

throw/1:
throw(Ball)
Raises an error, throwing the exception Ball, to be caught by an ancestor catch/3 or intercept/3. The closest matching ancestor is chosen. Exceptions are also thrown by other builtins in case of error.

halt/0:
halt(halt
Halt the system, exiting to the invoking shell. Usage:
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

halt/1:
halt(Code)
Halt the system, exiting to the invoking shell, returning exit code Code. Usage:
- The following properties should hold at call time:
  Code is an integer. (basic_props:int/1)

abort/0:
abort(abort
Abort the current execution.
24 Changing system behaviour and various flags

Author(s): Daniel Cabeza, Mats Carlsson.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#213 (2002/5/14, 18:11:29 CEST)

Flags define some parameters of the system and control the behavior of system or library predicates. Each flag has a name and an associated predefined value, and except some system flags which are fixed in general their associated value is changeable. Predefined flags in the system are:

**version**  
The Ciao version, as a term `ciao(Version,Patch)`. `Version` is a floating point number, `Patch` is an integer. Unchangeable.

**argv**  
Its value is a list of atoms representing the program arguments supplied when the current executable was invoked. This is the value to which is instantiated the argument of the `main/1` predicate at executable startup. Unchangeable.

**bounded**  
It is `false`, to denote that the range of integers can be considered infinite (but see `int/1`). Unchangeable.

**fileerrors**  
If `on`, predicates handling files give errors (throw exceptions) when a file is inexistent or an operation is not allowed. If `off`, fail in that conditions. Initially `on`.

**gc**  
Controls whether garbage collection is done. May be `on` (default) or `off`.

**gc_margin**  
An integer `Margin`. If less than `Margin` kilobytes are reclaimed in a garbage collection then the size of the garbage collected area should be increased. Also, no garbage collection is attempted unless the garbage collected area has at least `Margin` kilobytes. Initially 500.

**gc_trace**  

**integer_rounding_function**  
It is `toward_zero`, so that `-1 =:= -3//2` succeeds. Unchangeable.

**max arity**  
It is 255, so that no compound term (or predicate) can have more than this number of arguments. Unchangeable.

**quiet**  
Controls which messages issued using `io_aux` are actually written. As the system uses that library to report its messages, this flag controls the *verbosity* of the system. Possible states of the flag are:

- `on`  
  No messages are reported.
- `error`  
  Only error messages are reported.
- `warning`  
  Only error and warning messages are reported.
- `off`  
  All messages are reported, except debug messages. This is the default state.
- `debug`  
  All messages, including debug messages, are reported. This is only intended for the system implementators.

**unknown**  
Controls action on calls to undefined predicates. The possible states of the flag are:

- `error`  
  An error is thrown with the error term `existence_error(procedure, F/A)`. 
fail The call simply fails.
warning A warning is written and the call fails.
The state is initially error.

24.1 Usage and interface (prolog_flags)

- Library usage:
  These predicates are built-in in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    set_prolog_flag/2, current_prolog_flag/2, prolog_flag/3,
    push_prolog_flag/2, pop_prolog_flag/1, prompt/2, gc/0, nogc/0, fileerrors/0,
    nogc/0, fileerrors/0.
  - Multifiles:
    define_flag/3.

24.2 Documentation on exports (prolog_flags)

set_prolog_flag/2:
  set_prolog_flag(FlagName, Value)
  Set existing flag FlagName to Value.
  Usage: (ISO)
  - The following properties should hold at call time:
    FlagName is an atom. (basic_props:atm/1)
    Value is any term. (basic_props:term/1)

current_prolog_flag/2:
  current_prolog_flag(FlagName, Value)
  FlagName is an existing flag and Value is the value currently associated with it.
  Usage: (ISO)
  - Calls should, and exit will be compatible with:
    FlagName is an atom. (basic_props:atm/1)
    Value is any term. (basic_props:term/1)
  - The following properties hold upon exit:
    FlagName is an atom. (basic_props:atm/1)
    Value is any term. (basic_props:term/1)

prolog_flag/3:
  prolog_flag(FlagName, OldValue, NewValue)
  FlagName is an existing flag, unify OldValue with the value associated with it, and set it
to new value NewValue.
  Usage 1:
Calls should, and exit will be compatible with:

FlagName is an atom. (basic_props:atm/1)
OldValue is any term. (basic_props:term/1)

The following properties should hold at call time:
NewValue is any term. (basic_props:term/1)

The following properties hold upon exit:
FlagName is an atom. (basic_props:atm/1)
OldValue is any term. (basic_props:term/1)

Usage 2: prolog_flag(FlagName, OldValue, NewValue)

- Description: Same as current_prolog_flag(FlagName, OldValue)
- The following properties should hold at call time:
  OldValue is a free variable. (term_typing:var/1)
  NewValue is a free variable. (term_typing:var/1)
  FlagName is an atom. (basic_props:atm/1)
  The terms OldValue and NewValue are strictly identical. (term_compare:== /2)

push_prolog_flag/2: PREDICATE
push_prolog_flag(Flag, NewValue)
Same as set_prolog_flag/2, but storing current value of Flag to restore it with pop_prolog_flag/1.
Usage:
- The following properties should hold at call time:
  Flag is an atom. (basic_props:atm/1)
  NewValue is any term. (basic_props:term/1)

pop_prolog_flag/1: PREDICATE
pop_prolog_flag(Flag)
Restore the value of Flag previous to the last non-canceled push_prolog_flag/2 on it.
Usage:
- The following properties should hold at call time:
  Flag is an atom. (basic_props:atm/1)

prompt/2: PREDICATE
prompt(Old, New)
Unify Old with the current prompt for reading, change it to New.
Usage 1:
- Calls should, and exit will be compatible with:
  Old is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  New is an atom. (basic_props:atm/1)
The following properties hold upon exit:
Old is an atom. 

Usage 2: prompt(Old, New)
Description: Unify Old with the current prompt for reading without changing it.

The following properties should hold at call time:
Old is a free variable.
New is a free variable.
The terms Old and New are strictly identical.

The following properties hold upon exit:
Old is an atom.
New is an atom.

PREDICATE
gc/0:
Usage:
Description: Enable garbage collection. Equivalent to set_prolog_flag(gc, on)

PREDICATE
nogc/0:
Usage:
Description: Disable garbage collection. Equivalent to set_prolog_flag(gc, off)

PREDICATE
fileerrors/0:
Usage:
Description: Enable reporting of file errors. Equivalent to set_prolog_flag(fileerrors, on)

PREDICATE
nofileerrors/0:
Usage:
Description: Disable reporting of file errors. Equivalent to set_prolog_flag(fileerrors, off)

24.3 Documentation on multilfiles (prolog_flags)

define_flag/3:
Usage 1: define_flag(Flag, ==(atom), Default)
Description: Possible values for the flag are atoms.
Example:
:- multifile define_flag/3.
    define_flag(tmpdir, atom, '/tmp').

- The following properties should hold at call time:
  
  Flag is a free variable. (term_typing:var/1)
  Default is a free variable. (term_typing:var/1)

- The following properties should hold upon exit:
  
  Flag is an atom. (basic_props:atm/1)
  Default is an atom. (basic_props:atm/1)

Usage 2: define_flag(Flag, ==(integer), Default)

- Description: Possible values for the flag are integers.
  
  Example:
  
  :- multifile define_flag/3.
      define_flag(max_connections, integer, 10).

  The following properties should hold at call time:
  
  Flag is a free variable. (term_typing:var/1)
  Default is a free variable. (term_typing:var/1)

  The following properties should hold upon exit:
  
  Flag is an atom. (basic_props:atm/1)
  Default is an integer. (basic_props:int/1)

Usage 3: define_flag(atm, list(Values), Default)

- Description: Possible values for the flag are the elements of Values.
  
  Example:
  
  :- multifile define_flag/3.
      define_flag(debug, [on, debug, trace, off], off).

  The following properties should hold upon exit:
  
  Default is an element of Values. (basic_props:member/2)
25 Fast/concurrent update of facts

Author(s): Daniel Cabeza, Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#3 (2003/4/7, 13:48:19 CEST)

Prolog implementations traditionally implement the concept of dynamic predicates: predicates which can be inspected or modified at run-time, adding or deleting individual clauses. The power of this feature comes at a cost: as new clause bodies can be arbitrarily added to the program, new predicate calls can arise which are not 'visible' at compile-time, thus complicating global analysis and optimization of the code. But it is the case that most of the time what the programmer wants is simply to store data, with the purpose of sharing it between search branches, predicates, or even execution threads. In Ciao the concept of data predicate serves this purpose: a data predicate is a predicate composed exclusively by facts, which can be inspected, and dynamically added or deleted, at run-time. Using data predicates instead of normal dynamic predicates brings benefits in terms of speed, but above all makes the code much easier to analyze automatically and thus allows better optimization.

Also, a special kind of data predicates exists, concurrent predicates, which can be used to communicate/synchronize among different execution threads (see Chapter 79 [Low-level concurrency/multithreading primitives], page 355).

Data predicates must be declared through a data/1 declaration. Concurrent data predicates must be declared through a concurrent/1 declaration.

25.1 Usage and interface (data_facts)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    - asserta_fact/1, asserta_fact/2, assertz_fact/1, assertz_fact/2, current_fact/1, current_fact/2, retract_fact/1, retractall_fact/1, current_fact_nb/1, retract_fact_nb/1, close_predicate/1, open_predicate/1, set_fact/1, erase/1.

25.2 Documentation on exports (data_facts)

asserta_fact/1:

asserta_fact(Fact)

Fact is added to the corresponding data predicate. The fact becomes the first clause of the predicate concerned.

Meta-predicate with arguments: asserta_fact(fact).

Usage:
  - The following properties should hold at call time:
    Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props: callable/1)
asserta_fact/2:  
asserta_fact(Fact, Ref)  
Same as asserta_fact/1, instantiating Ref to a unique identifier of the asserted fact.  
Meta-predicate with arguments: asserta_fact(fact,?).  
Usage:  
- The following properties should hold at call time:  
  Fact is a term which represents a goal, i.e., an atom or a structure.  
  Ref is a free variable.  
- The following properties hold upon exit:  
  Ref is a reference of a dynamic or data clause.  

assertz_fact/1:  
assertz_fact(Fact)  
Fact is added to the corresponding data predicate. The fact becomes the last clause of the predicate concerned.  
Meta-predicate with arguments: assertz_fact(fact).  
Usage:  
- The following properties should hold at call time:  
  Fact is a term which represents a goal, i.e., an atom or a structure.  

assertz_fact/2:  
assertz_fact(Fact, Ref)  
Same as assertz_fact/1, instantiating Ref to a unique identifier of the asserted fact.  
Meta-predicate with arguments: assertz_fact(fact,?).  
Usage:  
- The following properties should hold at call time:  
  Fact is a term which represents a goal, i.e., an atom or a structure.  
  Ref is a free variable.  
- The following properties hold upon exit:  
  Ref is a reference of a dynamic or data clause.  

current_fact/1:  
current_fact(Fact)  
Gives on backtracking all the facts defined as data or concurrent which unify with Fact. It is faster than calling the predicate explicitly, which do invoke the meta-interpreter. If the Fact has been defined as concurrent and has not been closed, current_fact/1 will wait (instead of failing) for more clauses to appear after the last clause of Fact is returned.  
Meta-predicate with arguments: current_fact(fact).  
Usage:  
- The following properties should hold at call time:  
  Fact is a term which represents a goal, i.e., an atom or a structure.  

PREDICATE
current_fact/2: PREDICATE

 current_fact(Fact, Ref)
 Fact is a fact of a data predicate and Ref is its reference identifying it uniquely. 
 Meta-predicate with arguments: current_fact(fact,?). 

 Usage 1: 
  - Description: Gives on backtracking all the facts defined as data which unify with Fact, instantiating Ref to a unique identifier for each fact. 
  - The following properties should hold at call time: 
    Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1) 
    Ref is a free variable. (term_typing:var/1) 
  - The following properties hold upon exit: 
    Ref is a reference of a dynamic or data clause. (data_facts:reference/1) 

 Usage 2: 
  - Description: Given Ref, unifies Fact with the fact identified by it. 
  - Calls should, and exit will be compatible with: 
    Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1) 
  - The following properties should hold at call time: 
    Ref is a reference of a dynamic or data clause. (data_facts:reference/1) 
  - The following properties hold upon exit: 
    Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1) 

retract_fact/1: PREDICATE

 retract_fact(Fact) 
 Unifies Fact with the first matching fact of a data predicate, and then erases it. On backtracking successively unifies with and erases new matching facts. If Fact is declared as concurrent and is non-closed, retract_fact/1 will wait for more clauses or for the closing of the predicate after the last matching clause has been removed. 
 Meta-predicate with arguments: retract_fact(fact). 

 Usage: 
  - The following properties should hold at call time: 
    Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1) 

retractall_fact/1: PREDICATE

 retractall_fact(Fact) 
 Erases all the facts of a data predicate unifying with Fact. Even if all facts are removed, the predicate continues to exist. 
 Meta-predicate with arguments: retractall_fact(fact). 

 Usage: 
  - The following properties should hold at call time: 
    Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
**current_fact_nb/1:**

*current_fact_nb*(_Fact_)

Behaves as *current_fact/1* but a fact is never waited on even if it is concurrent and non-closed.

*Meta-predicate* with arguments: *current_fact_nb*(fact).

*Usage:*

– *The following properties should hold at call time:*
  
  _Fact_ is a term which represents a goal, i.e., an atom or a structure.  

**retract_fact_nb/1:**

*retract_fact_nb*(_Fact_)

Behaves as *retract_fact/1*, but never waits on a fact, even if it has been declared as concurrent and is non-closed.

*Meta-predicate* with arguments: *retract_fact_nb*(fact).

*Usage:*

– *The following properties should hold at call time:*
  
  _Fact_ is a term which represents a goal, i.e., an atom or a structure.  

**close_predicate/1:**

*close_predicate*(_Pred_)

Changes the behavior of the predicate _Pred_ if it has been declared as a concurrent predicate: calls to this predicate will fail (instead of wait) if no more clauses of _Pred_ are available.

*Meta-predicate* with arguments: *close_predicate*(fact).

*Usage:*

– *The following properties should hold at call time:*
  
  _Pred_ is a term which represents a goal, i.e., an atom or a structure.  

**open_predicate/1:**

*open_predicate*(_Pred_)

Reverts the behavior of concurrent predicate _Pred_ to waiting instead of failing if no more clauses of _Pred_ are available.

*Meta-predicate* with arguments: *open_predicate*(fact).

*Usage:*

– *The following properties should hold at call time:*
  
  _Pred_ is a term which represents a goal, i.e., an atom or a structure.  
Chapter 25: Fast/concurrent update of facts

Chapter 25: Fast/concurrent update of facts

set_fact/1:

set_fact(Fact)

Sets Fact as the unique fact of the corresponding data predicate.

Meta-predicate with arguments: set_fact(fact).

Usage:

- \textit{The following properties should hold at call time:}

  - Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

erase/1:

erase(Ref)

Deletes the clause referenced by Ref.

Usage:

- \textit{The following properties should hold at call time:}

  - Ref is a reference of a dynamic or data clause. (data_facts:reference/1)

25.3 Documentation on internals (data_facts)

data/1:

Usage: :- data Predicates.

- \textit{Description:} Defines each predicate in Predicates as a data predicate. If a predicate is defined data in a file, it must be defined data in every file containing clauses for that predicate. The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

  - \textit{The following properties hold at call time:}

    - Predicates is a sequence or list of prednames. (basic_props:sequence_or_list/2)

concurrent/1:

Usage: :- concurrent Predicates.

- \textit{Description:} Defines each predicate in Predicates as a concurrent predicate. If a predicate is defined concurrent in a file, it must be defined concurrent in every file containing clauses for that predicate. The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

  - \textit{The following properties hold at call time:}

    - Predicates is a sequence or list of prednames. (basic_props:sequence_or_list/2)

reference/1:

Usage: reference(R)

- \textit{Description:} R is a reference of a dynamic or data clause.
Chapter 26: Extending the syntax

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#71 (2003/12/19, 16:54:53 CET)

This chapter documents the builtin directives in Ciao for extending the syntax of source files. Note that the ISO-Prolog directive char_conversion/2 is not implemented, since Ciao does not (yet) have a character conversion table.

26.1 Usage and interface (syntax_extensions)

- Library usage:
  These directives are builtin in Ciao, so nothing special has to be done to use them.

26.2 Documentation on internals (syntax_extensions)

op/3:
Description: Updates the operator table for reading the terms in the rest of the current text, in the same way as the builtin op/3 does. Its scope is local to the current text. Usually included in package files.
The following properties hold at call time:
- Priority is an integer.
- Op_spec specifies the type and associativity of an operator.
- Operator is an atom or a list of atoms.

new_declaration/1:
Usage: :- new_declaration(Predicate).
Description: Declares Predicate to be a valid declaration in the rest of the current text. Such declarations are simply ignored by the compiler or top level, but can be used by other code processing programs such as an automatic documentator. Also, they can easily translated into standard code (a set of facts and/or rules) by defining a suitable expansion (e.g., by add_sentence_trans/1, etc.). This is typically done in package files.
Equivalent to new_declaration(Predicate, off).
The following properties hold at call time:
- Predicate is a Name/Arity structure denoting a predicate name:
  \[
  \text{predname}(P/A) :- \\
  \text{atm}(P), \\
  \text{nnegint}(A). \\
  \]

new_declaration/2:
Usage: :- new_declaration(Predicate, In_Itf).

- Description: Declares \texttt{Predicate} to be a valid declaration in the rest of the current text. Such declarations will be included in the interface file for this file if \texttt{In_Itf} is 'on', not if it is 'off'. Including such declarations in interface files makes them visible while processing other modules which make use of this one.

- The following properties hold at call time:
  \texttt{Predicate} is a Name/Arity structure denoting a predicate name:
  \begin{verbatim}
  predname(P/A) :-
  atm(P),
  nnegint(A).
  \end{verbatim}
  (basic Props: predname/1)
  \texttt{In_Itf} is 'on' or 'off'
  (syntax extensions: switch/1)

load_compilation_module/1:
Usage: :- load_compilation_module(File).

- Description: Loads code defined in \texttt{File} into the compiler, usually including predicates which define translations of terms, for use with the declarations \texttt{add_sentence_trans/1} and similar ones. Normally included in package files.

- The following properties hold at call time:
  \texttt{File} is a source name.
  (streams basic: sourcename/1)

add_sentence_trans/1:
Usage: :- add_sentence_trans(Predicate).

- Description: Starts a translation, defined by \texttt{Predicate}, of the terms read by the compiler in the rest of the current text. For each subsequent term read by the compiler, the translation predicate is called to obtain a new term which will be used by the compiler as if it where the term present in the file. If the call fails, the term is used as such. A list may be returned also, to translate a single term into several terms. Before calling the translation predicate with actual program terms, it is called with an input of 0 to give an opportunity of making initializations for the module, discarding the result (note that normally a 0 could not be there). \texttt{Predicate} must be exported by a module previously loaded with a \texttt{load_compilation_module/1} declaration. Normally included in package files.

- The following properties hold at call time:
  \texttt{Predicate} is a translation predicate spec (has arity 2 or 3).
  (syntax extensions: translation predname/1)

add_term_trans/1:
Usage: :- add_term_trans(P).

- Description: Starts a translation, defined by \texttt{Predicate}, of the terms and sub-terms read by the compiler in the rest of the current text. This translation is performed after all translations defined by \texttt{add_sentence_trans/1} are done. For each subsequent term read by the compiler, and recursively any subterm included, the translation predicate is called to possibly obtain a new term to replace the old one. Care must
be taken of not introducing an endless loop of translations. **Predicate** must be exported by a module previously loaded with a `load_compilation_module/1` declaration. Normally included in package files.

- **The following properties hold at call time:**

  - **Predicate** is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

### add_goal_trans/1:

**DECLARATION**

**Usage:** :- add_goal_trans(Predicate).

- **Description:** Declares a translation, defined by **Predicate**, of the goals present in the clauses of the current text. This translation is performed after all translations defined by `add_sentence_trans/1` and `add_term_trans/1` are done. For each clause read by the compiler, the translation predicate is called with each goal present in the clause to possibly obtain other goal to substitute the original one, and the translation is subsequently applied to the resulting goal. Care must be taken of not introducing an endless loop of translations. **Predicate** must be exported by a module previously loaded with a `load_compilation_module/1` declaration. Bear in mind that this type of translation noticeably slows down compilation. Normally included in package files.

  - **The following properties hold at call time:**

    - **Predicate** is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

### add_clause_trans/1:

**DECLARATION**

**Usage:** :- add_clause_trans(Predicate).

- **Description:** Declares a translation, defined by **Predicate**, of the clauses of the current text. The translation is performed before `add_goal_trans/1` translations but after `add_sentence_trans/1` and `add_term_trans/1` translations. The usefulness of this translation is that information of the interface of related modules is available when it is performed. For each clause read by the compiler, the translation predicate is called with the first argument instantiated to a structure `clause(Head,Body)`, and the predicate must return in the second argument a similar structure, without changing the functor in **Head** (or fail, in which case the clause is used as is). Before executing the translation predicate with actual clauses it is called with an input of `clause(0,0)`, discarding the result.

  - **The following properties hold at call time:**

    - **Predicate** is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

### translation_predname/1:

**REGTYPE**

A translation predicate is a predicate of arity 2 or 3 used to make compile-time translations. The compiler invokes a translation predicate instantiating its first argument with the item to be translated, and if the predicate is of arity 3 its third argument with the name of the module where the translation is done. If the call is successful, the second argument is used as if that item were in the place of the original, else the original item is used.

**Usage:** `translation_predname(P)`

- **Description:** P is a translation predicate spec (has arity 2 or 3).
27 Message printing primitives

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#72 (2003/12/19, 16:56:14 CET)

This module provides predicates for printing in a unified way informational messages, and also for printing some terms in a specific way.

27.1 Usage and interface (io_aux)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    - message/2, message_lns/4, error/1, warning/1, note/1, message/1, debug/1,
      inform_user/1, display_string/1, display_list/1, display_term/1.

27.2 Documentation on exports (io_aux)

message/2: message(Type, Message)

Output to standard error Message, which is of type Type. The quiet prolog flag (see Chapter 24 [Changing system behaviour and various flags], page 131) controls which messages are actually output, depending on its type. Also, for error, warning and note messages, a prefix is output which denotes the severity of the message. Message is an item or a list of items from this list:

- $(String) String is a string, which is output with display_string/1.
- "'(Term) Term is output quoted. If the module write is loaded, the term is output with writeq/1, else with displayq/1.
- "~(Term) Term is output unquoted. If the module write is loaded, the term is output with write/1, else with display/1.
- [] (Term) Term is recursively output as a message, can be an item or a list of items from this list.
- Term Any other term is output with display/1.

Usage: message(Type, Message)
- The following properties should hold at call time:
  Type is an atom. (basic_props:atm/1)
  Type is an element of [error, warning, note, message, debug]. (basic_props:member/2)
message_lns/4:

\[
\text{message}_\text{lns}(\text{Type}, \text{L0}, \text{L1}, \text{Message})
\]

Output to standard error \text{Message}, which is of type \text{Type}, and occurs between lines \text{L0} and \text{L1}. This is the same as \text{message/2}, but printing the lines where the message occurs in a unified way (this is useful because automatic tools such as the emacs mode know how to parse them).

Usage: \text{message}_\text{lns}(\text{Type}, \text{L0}, \text{L1}, \text{Message})

- The following properties should hold at call time:
  \text{Type} is an atom. \quad (\text{basic_props:atm/1})
  \text{Type} is an element of \{\text{error, warning, note, message, debug}\}. \quad (\text{basic_props:member/2})

\text{error/1:}

Defined as

\[
\text{error}(	ext{Message}) : - \\
\text{message(error,Message)}.
\]

\text{warning/1:}

Defined as

\[
\text{warning}(	ext{Message}) : - \\
\text{message(warning,Message)}.
\]

\text{note/1:}

Defined as

\[
\text{note}(	ext{Message}) : - \\
\text{message(note,Message)}.
\]

\text{message/1:}

Defined as

\[
\text{message}(	ext{Message}) : - \\
\text{message(message,Message)}.
\]

\text{debug/1:}

Defined as

\[
\text{debug}(	ext{Message}) : - \\
\text{message(debug,Message)}.
\]
inform_user/1:  
inform_user(Message)  
Similar to message/1, but Message is output with display_list/1. This predicate is obsolete, and may disappear in future versions.

display_string/1:  
display_string(String)  
Output String as the sequence of characters it represents.  
Usage: display_string(String)  
   -- The following properties should hold at call time:  
      String is a string (a list of character codes).  
      (basic Props: string/1)

display_list/1:  
display_list(List)  
Outputs List. If List is a list, do display/1 on each of its elements, else do display/1 on List.

display_term/1:  
display_term(Term)  
Output Term in a way that a read/1 will be able to read it back, even if operators change.

27.3 Known bugs and planned improvements (io_aux)  
•  
   message/2 assumes that a module with name 'write' is library(write).
28 Attributed variables

Author(s): Christian Holzbaur, Daniel Cabeza, Manuel Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.5#157 (2000/5/30, 13:4:47 CEST)

These predicates allow the manipulation of attributed variables. Attributes are special terms which are attached to a (free) variable, and are hidden from the normal Prolog computation. They can only be treated by using the predicates below.

28.1 Usage and interface (attributes)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    - attach_attribute/2, get_attribute/2, update_attribute/2, detach_attribute/1.
  - Multifiles:
    - verify_attribute/2, combine_attributes/2.

28.2 Documentation on exports (attributes)

attach_attribute/2:
Usage: attach_attribute(Var, Attr)
- Description: Attach attribute Attr to Var.
- The following properties should hold at call time:
  - Var is a free variable. (term_typing:var/1)
  - Attr is currently a term which is not a free variable. (term_typing:nonvar/1)

get_attribute/2:
Usage: get_attribute(Var, Attr)
- Description: Unify Attr with the attribute of Var, or fail if Var has no attribute.
- The following properties should hold at call time:
  - Var is a free variable. (term_typing:var/1)
  - The following properties should hold upon exit:
    - Attr is currently a term which is not a free variable. (term_typing:nonvar/1)

update_attribute/2:
Usage: update_attribute(Var, Attr)
- Description: Change the attribute of attributed variable Var to Attr.
The following properties should hold at call time:
Var is a free variable. (term_typing:var/1)
Attr is currently a term which is not a free variable. (term_typing:nonvar/1)

detach_attribute/1:
Usage: detach_attribute(Var)
Description: Take out the attribute from the attributed variable Var.
The following properties should hold at call time:
Var is a free variable. (term_typing:var/1)

28.3 Documentation on multifiles (attributes)

verify_attribute/2:
The predicate is multifile.
Usage: verify_attribute(Attr, Term)
Description: A user defined predicate. This predicate is called when an attributed variable with attribute Attr is about to be unified with the non-variable term Term. The user should define this predicate (as multifile) in the modules implementing special unification.
The following properties should hold at call time:
Attr is currently a term which is not a free variable. (term_typing:nonvar/1)
Term is currently a term which is not a free variable. (term_typing:nonvar/1)

combine_attributes/2:
The predicate is multifile.
Usage: combine_attributes(Var1, Var2)
Description: A user defined predicate. This predicate is called when two attributed variables with attributes Var1 and Var2 are about to be unified. The user should define this predicate (as multifile) in the modules implementing special unification.
The following properties should hold at call time:
Var1 is a free variable. (term_typing:var/1)
Var2 is a free variable. (term_typing:var/1)

28.4 Other information (attributes)
Note that combine_attributes/2 and verify_attribute/2 are not called with the attributed variables involved, but with the corresponding attributes instead. The reasons are:
• There are simple applications which only refer to the attributes.
• If the application wants to refer to the attributed variables themselves, they can be made part the attribute term. The implementation of freeze/2 utilizes this technique. Note that this does not lead to cyclic structures, as the connection between an attributed variable and its attribute is invisible to the pure parts of the Prolog implementation.
• If attributed variables were passed as arguments, the user code would have to refer to the attributes through an extra call to get_attribute/2.
As the attribute is the first argument to each of the two predicates, indexing applies. Note that attributed variables themselves look like variables to the indexing mechanism.

However, future improvements may change or extend the interface to attributed variables in order to provide a richer and more expressive interface.

For customized output of attributed variables, please refer to the documentation of the predicate portray_attribute/2.
29 Gathering some basic internal info

Author(s): Daniel Cabeza, Manuel Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#9 (2003/4/7, 14:1:46 CEST)
This module provides predicates which return basic internal info.

29.1 Usage and interface (system_info)

- Library usage:
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- Exports:
  - Predicates:
    get_arch/1, get_os/1, this_module/1, current_module/1, ciaolibdir/1.

29.2 Documentation on exports (system_info)

get_arch/1: PREDICATE
This predicate will describe the computer architecture which is currently executing the
predicate.
Computer architectures are identified by a simple atom. This atom is implementation-
deefined, and may suffer any change from one Ciao Prolog version to another.
For example, Ciao Prolog running on an Intel-based machine will retrieve:
?- get_arch(I).
  I = i86 ? ;
no
?- Usage: get_arch(ArchDescriptor)
- Description: Unifies ArchDescriptor with a simple atom which describes the com-
  puter architecture currently executing the predicate.
- Calls should, and exit will be compatible with:
  ArchDescriptor is an atom. (basic_props:atm/1)

get_os/1: PREDICATE
This predicate will describe the Operating System which is running on the machine cur-
rently executing the Prolog program.
Operating Systems are identified by a simple atom. This atom is implementation-defined,
and may suffer any change from one Ciao Prolog version to another.
For example, Ciao Prolog running on Linux will retrieve:
?- get_os(I).

I = 'LINUX' ? ;

no
?- 

Usage: get_os(OsDescriptor)
- Description: Unifies OsDescriptor with a simple atom which describes the running Operating System when predicate was called.
- Calls should, and exit will be compatible with:
  OsDescriptor is an atom. (basic_props:atm/1)

this_module/1:  PREDICATE
Meta-predicate with arguments: this_module(addmodule).

Usage: this_module(Module)
- Description: Module is the internal module identifier for current module.
- Call and exit should be compatible with:
  Module is an internal module identifier (system_info:internal_module_id/1)

current_module/1:  PREDICATE
This predicate will successively unify its argument with all module names currently loaded. Module names will be simple atoms.
When called using a free variable as argument, it will retrieve on backtracking all modules currently loaded. This is useful when called from the Ciao toplevel.
When called using a module name as argument it will check whether the given module is loaded or not. This is useful when called from user programs.

Usage: current_module(Module)
- Description: Retrieves (on backtracking) all currently loaded modules into your application.
- Call and exit should be compatible with:
  Module is an internal module identifier (system_info:internal_module_id/1)
- The following properties should hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

ciao_libdir/1:  PREDICATE
Usage: ciao_libdir(CiaoPath)
- Description: CiaoPath is the path to the root of the Ciao libraries. Inside this directory, there are the directories 'lib', 'library' and 'contrib', which contain library modules.
- Call and exit should be compatible with:
  CiaoPath is an atom. (basic_props:atm/1)
29.3 Documentation on internals (system_info)

**internal_module_id/1:**  
For a user file it is a term user/1 with an argument different for each user file, for other modules is just the name of the module (as an atom).

**Usage:** `internal_module_id(M)`  

- **Description:** M is an internal module identifier
30 Other predicates and features defined by default

Author(s): Daniel Cabeza.

To simplify the use of Ciao Prolog to the first-timers, some other predicates and features are defined by default in normal cases, to provide more or less what other prologs define by default. Here are explicitly listed the predicates defined, coming from several libraries. Apart from those, the features defined in Chapter 40 [Definite clause grammars], page 211 and Chapter 52 [Enabling operators at run-time], page 265 are also activated.

30.1 Usage and interface (default_predicates)

- Library usage:
  
  No need of explicit loading. It is included by default in modules starting with a module/2 declaration or user files without a starting use_package/1 declaration. In the Ciao shell, it is loaded by default when no ~/.ciaorc exists. Note that :- module(modulename, exports) is equivalent to :- module(modulename, exports,[default]) If you do not want these predicates/features loaded for a given file (in order to make the executable smaller) you can ask for this explicitly using :- module(modulename, exports,[]) or in a user file :- use_package([]).

- Other modules used:
  
  System library modules:
  
  aggregates, dynamic, read, write, operators, iso_byte_char, iso_misc, format, lists, sort, between, compiler/compiler, system, prolog_sys, dec10_io, old_database, ttyout.

30.2 Documentation on exports (default_predicates)

op/3: (UNDOC_REEXPORT)
  Imported from operators (see the corresponding documentation for details).

current_op/3: (UNDOC_REEXPORT)
  Imported from operators (see the corresponding documentation for details).

append/3: (UNDOC_REEXPORT)
  Imported from lists (see the corresponding documentation for details).

delete/3: (UNDOC_REEXPORT)
  Imported from lists (see the corresponding documentation for details).

select/3: (UNDOC_REEXPORT)
  Imported from lists (see the corresponding documentation for details).
nth/3: (UNDOC_REEXPORT)
Imported from lists (see the corresponding documentation for details).

last/2: (UNDOC_REEXPORT)
Imported from lists (see the corresponding documentation for details).

reverse/2: (UNDOC_REEXPORT)
Imported from lists (see the corresponding documentation for details).

length/2: (UNDOC_REEXPORT)
Imported from lists (see the corresponding documentation for details).

use_module/1: (UNDOC_REEXPORT)
Imported from compiler (see the corresponding documentation for details).

use_module/2: (UNDOC_REEXPORT)
Imported from compiler (see the corresponding documentation for details).

ensure_loaded/1: (UNDOC_REEXPORT)
Imported from compiler (see the corresponding documentation for details).

~/2: (UNDOC_REEXPORT)
Imported from aggregates (see the corresponding documentation for details).

findnsols/5: (UNDOC_REEXPORT)
Imported from aggregates (see the corresponding documentation for details).

findnsols/4: (UNDOC_REEXPORT)
Imported from aggregates (see the corresponding documentation for details).

findall/4: (UNDOC_REEXPORT)
Imported from aggregates (see the corresponding documentation for details).

findall/3: (UNDOC_REEXPORT)
Imported from aggregates (see the corresponding documentation for details).
Chapter 30: Other predicates and features defined by default

bagof/3: Imported from aggregates (see the corresponding documentation for details).

setof/3: Imported from aggregates (see the corresponding documentation for details).

wellformed_body/3: Imported from dynamic (see the corresponding documentation for details).

data/1: Imported from dynamic (see the corresponding documentation for details).

dynamic/1: Imported from dynamic (see the corresponding documentation for details).

current_predicate/2: Imported from dynamic (see the corresponding documentation for details).

current_predicate/1: Imported from dynamic (see the corresponding documentation for details).

clause/3: Imported from dynamic (see the corresponding documentation for details).

clause/2: Imported from dynamic (see the corresponding documentation for details).

abolish/1: Imported from dynamic (see the corresponding documentation for details).

retractall/1: Imported from dynamic (see the corresponding documentation for details).

retract/1: Imported from dynamic (see the corresponding documentation for details).
assert/2: (UNDOC_REEXPORT)
Imported from dynamic (see the corresponding documentation for details).

assert/1: (UNDOC_REEXPORT)
Imported from dynamic (see the corresponding documentation for details).

assertz/2: (UNDOC_REEXPORT)
Imported from dynamic (see the corresponding documentation for details).

assertz/1: (UNDOC_REEXPORT)
Imported from dynamic (see the corresponding documentation for details).

asserta/2: (UNDOC_REEXPORT)
Imported from dynamic (see the corresponding documentation for details).

asserta/1: (UNDOC_REEXPORT)
Imported from dynamic (see the corresponding documentation for details).

second_prompt/2: (UNDOC_REEXPORT)
Imported from read (see the corresponding documentation for details).

read_top_level/3: (UNDOC_REEXPORT)
Imported from read (see the corresponding documentation for details).

read_term/3: (UNDOC_REEXPORT)
Imported from read (see the corresponding documentation for details).

read_term/2: (UNDOC_REEXPORT)
Imported from read (see the corresponding documentation for details).

read/2: (UNDOC_REEXPORT)
Imported from read (see the corresponding documentation for details).

read/1: (UNDOC_REEXPORT)
Imported from read (see the corresponding documentation for details).
printable_char/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

prettyvars/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

numbervars/3: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

portray_clause/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

portray_clause/2: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

write_list1/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

print/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

print/2: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

write_canonical/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

write_canonical/2: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

writeq/1: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).

writeq/2: (UNDONE_REEXPORT)
  Imported from write (see the corresponding documentation for details).
write/1: (UNDOC_REEXPRT)
    Imported from write (see the corresponding documentation for details).

write/2: (UNDOC_REEXPRT)
    Imported from write (see the corresponding documentation for details).

write_option/1: (UNDOC_REEXPRT)
    Imported from write (see the corresponding documentation for details).

write_term/2: (UNDOC_REEXPRT)
    Imported from write (see the corresponding documentation for details).

write_term/3: (UNDOC_REEXPRT)
    Imported from write (see the corresponding documentation for details).

put_char/2: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).

put_char/1: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).

peek_char/2: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).

peek_char/1: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).

get_char/2: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).

get_char/1: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).

put_byte/2: (UNDOC_REEXPRT)
    Imported from iso_byte_char (see the corresponding documentation for details).
put_byte/1: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

peek_byte/2: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

peek_byte/1: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

get_byte/2: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

get_byte/1: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

number_chars/2: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

atom_chars/2: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

char_code/2: (UNDOP_REEXPORT)
Imported from iso_byte_char (see the corresponding documentation for details).

unify_with_occurs_check/2: (UNDOP_REEXPORT)
Imported from iso_misc (see the corresponding documentation for details).

sub_atom/5: (UNDOP_REEXPORT)
Imported from iso_misc (see the corresponding documentation for details).

compound/1: (UNDOP_REEXPORT)
Imported from iso_misc (see the corresponding documentation for details).

once/1: (UNDOP_REEXPORT)
Imported from iso_misc (see the corresponding documentation for details).
\(\leq/2:\) imported from \texttt{iso} \_\texttt{misc} (see the corresponding documentation for details).

\texttt{format} \_\texttt{control}/1: imported from \texttt{format} (see the corresponding documentation for details).

\texttt{format}/3: imported from \texttt{format} (see the corresponding documentation for details).

\texttt{format}/2: imported from \texttt{format} (see the corresponding documentation for details).

\texttt{keylist}/1: imported from \texttt{sort} (see the corresponding documentation for details).

\texttt{keysort}/2: imported from \texttt{sort} (see the corresponding documentation for details).

\texttt{sort}/2: imported from \texttt{sort} (see the corresponding documentation for details).

\texttt{between}/3: imported from \texttt{between} (see the corresponding documentation for details).

\texttt{cyg2win}/3: imported from \texttt{system} (see the corresponding documentation for details).

\texttt{rename} \_\texttt{file}/2: imported from \texttt{system} (see the corresponding documentation for details).

\texttt{delete} \_\texttt{directory}/1: imported from \texttt{system} (see the corresponding documentation for details).

\texttt{delete} \_\texttt{file}/1: imported from \texttt{system} (see the corresponding documentation for details).
chmod/3:
Imported from system (see the corresponding documentation for details).

chmod/2:
Imported from system (see the corresponding documentation for details).

fmode/2:
Imported from system (see the corresponding documentation for details).

modif_time0/2:
Imported from system (see the corresponding documentation for details).

modif_time/2:
Imported from system (see the corresponding documentation for details).

file_properties/6:
Imported from system (see the corresponding documentation for details).

file_property/2:
Imported from system (see the corresponding documentation for details).

file_exists/2:
Imported from system (see the corresponding documentation for details).

file_exists/1:
Imported from system (see the corresponding documentation for details).

mktemp/2:
Imported from system (see the corresponding documentation for details).

directory_files/2:
Imported from system (see the corresponding documentation for details).

exec/3:
Imported from system (see the corresponding documentation for details).
exec/4: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

popen_mode/1: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

popen/3: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

system/2: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

system/1: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

shell/2: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

shell/1: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

shell/0: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

cd/1: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

working_directory/2: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

make_dirpath/1: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).

make_dirpath/2: (UNDOS.REEXPORT)
Imported from system (see the corresponding documentation for details).
make_directory/1:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

make_directory/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

umask/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

current_executable/1:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

current_host/1:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

get_pid/1:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

extract_paths/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

copy_file/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

del_env/1:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

set_env/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

current_env/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).

setenvstr/2:  (UNDOC_REEXPORT)
Imported from system (see the corresponding documentation for details).
getenvstr/2:  
Imported from system (see the corresponding documentation for details).

datetime_struct/1:  
Imported from system (see the corresponding documentation for details).

datetime/9:  
Imported from system (see the corresponding documentation for details).

datetime/1:  
Imported from system (see the corresponding documentation for details).

time/1:  
Imported from system (see the corresponding documentation for details).

pause/1:  
Imported from system (see the corresponding documentation for details).

new_atom/1:  
Imported from prolog_sys (see the corresponding documentation for details).

garbage_collect/0:  
Imported from prolog_sys (see the corresponding documentation for details).

current_atom/1:  
Imported from prolog_sys (see the corresponding documentation for details).

predicate_property/2:  
Imported from prolog_sys (see the corresponding documentation for details).

statistics/2:  
Imported from prolog_sys (see the corresponding documentation for details).

statistics/0:  
Imported from prolog_sys (see the corresponding documentation for details).
close_file/1:  
Imported from dec10_io (see the corresponding documentation for details).

told/0:  
Imported from dec10_io (see the corresponding documentation for details).

telling/1:  
Imported from dec10_io (see the corresponding documentation for details).

tell/1:  
Imported from dec10_io (see the corresponding documentation for details).

seen/0:  
Imported from dec10_io (see the corresponding documentation for details).

seeing/1:  
Imported from dec10_io (see the corresponding documentation for details).

see/1:  
Imported from dec10_io (see the corresponding documentation for details).

current_key/2:  
Imported from old_database (see the corresponding documentation for details).

recorded/3:  
Imported from old_database (see the corresponding documentation for details).

recordz/3:  
Imported from old_database (see the corresponding documentation for details).

recorda/3:  
Imported from old_database (see the corresponding documentation for details).

ttydisplay_string/1:  
Imported from ttyout (see the corresponding documentation for details).
ttyskip0/0: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttydisplayq/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttydisplay/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttyflush/0: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttytab/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttyskip/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttyput/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttynl/0: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttyget/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).

ttyget1/1: (UNDQC_REEXPORT)
    Imported from ttyout (see the corresponding documentation for details).
PART III - ISO-Prolog library (iso)

Author(s): The CLIP Group.

This part documents the iso package which provides to Ciao programs (most of) the ISO-Prolog functionality, including the ISO-Prolog builtins not covered by the basic library.
31 ISO-Prolog package

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#74 (2003/12/19, 17:1:5 CET)

This library package allows the use of the ISO-Prolog predicates in Ciao programs. The compatibility is not at 100% yet.

31.1 Usage and interface (iso)

- Library usage:
  
  ```prolog
  :- use_package(iso).
  
  or
  
  :- module(...,...,[iso]).
  ```

- Other modules used:
  
  - System library modules:
    
    ```prolog
    aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read, write.
    ```
32 All solutions predicates


Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.5#115 (2000/4/12, 12:17:22 CEST)

This module implements the standard solution aggregation predicates.

When there are many solutions to a problem, and when all those solutions are required to be collected together, this can be achieved by repeatedly backtracking and gradually building up a list of the solutions. The following built-in predicates are provided to automate this process.

32.1 Usage and interface (aggregates)

- Library usage:
  ```prolog
  :- use_module(library(aggregates)).
  ```

- Exports:
  - Predicates:
    - `setof/3`, `bagof/3`, `findall/3`, `findall/4`, `findsols/4`, `findsols/5`, `~/2`.
  - Other modules used:
    - System library modules:
      - `sort`, `lists`.

32.2 Documentation on exports (aggregates)

**setof/3:**

```prolog
setof(Template, Generator, Set)
```

Finds the Set of instances of the Template satisfying Generator. The set is in ascending order (see Chapter 18 [Comparing terms], page 95 for a definition of this order) without duplicates, and is non-empty. If there are no solutions, `setof` fails. `setof` may succeed in more than one way, binding free variables in `Generator` to different values. This can be avoided by using existential quantifiers on the free variables in front of `Generator`, using `~/2`. For example, given the clauses:

```prolog
father(bill, tom).
father(bill, ann).
father(bill, john).
father(harry, july).
father(harry, daniel).
```

The following query produces two alternative solutions via backtracking:

```prolog
?- setof(X,father(F,X),Sons).
```

F = bill,
Sons = [ann, john, tom] ;

F = harry,
Sons = [daniel, july] ;
```
no
?-

Meta-predicate with arguments: \texttt{setof(? goal,?).}

\textbf{General properties:} \texttt{setof(X, Y, Z)}

- \textit{The following properties hold globally:}
  
  This predicate is understood natively by CiaoPP as \texttt{findall(X,Y,Z)}.  
  \caption{basic\_props:native/2}

\textbf{Usage:}

- \textit{Calls should, and exit will be compatible with:}
  
  Set is a list.  
  \caption{basic\_props:list/1}

- \textit{The following properties should hold at call time:}
  
  Template is any term.  
  \caption{basic\_props:term/1}

  Generator is a term which represents a goal, i.e., an atom or a structure.  
  \caption{basic\_props:callable/1}

- \textit{The following properties hold upon exit:}
  
  Template is any term.  
  \caption{basic\_props:term/1}

  Set is a list.  
  \caption{basic\_props:list/1}

- \textit{The following properties hold globally:}
  
  Template is not further instantiated.  
  \caption{basic\_props:not\_further\_inst/2}

\textbf{bagof/3:}

\texttt{bagof(Template, Generator, Bag)}

Finds all the instances of the Template produced by the Generator, and returns them in the Bag in the order in which they were found. If the Generator contains free variables which are not bound in the Template, it assumes that this is like any other Prolog question and that you want bindings for those variables. This can be avoided by using existential quantifiers on the free variables in front of the Generator, using \texttt{^}/2.

Meta-predicate with arguments: \texttt{bagof(? goal,?).}

\textbf{General properties:} \texttt{bagof(X, Y, Z)}

- \textit{The following properties hold globally:}
  
  This predicate is understood natively by CiaoPP as \texttt{findall(X,Y,Z)}.  
  \caption{basic\_props:native/2}

\textbf{Usage:}

- \textit{Calls should, and exit will be compatible with:}
  
  Bag is a list.  
  \caption{basic\_props:list/1}

- \textit{The following properties should hold at call time:}
  
  Template is any term.  
  \caption{basic\_props:term/1}

  Generator is a term which represents a goal, i.e., an atom or a structure.  
  \caption{basic\_props:callable/1}

- \textit{The following properties hold upon exit:}
  
  Template is any term.  
  \caption{basic\_props:term/1}

  Bag is a list.  
  \caption{basic\_props:list/1}

- \textit{The following properties hold globally:}
  
  Template is not further instantiated.  
  \caption{basic\_props:not\_further\_inst/2}
PREDICATE

findall/3:

\texttt{findall} (Template, Generator, List)

A special case of bagof, where all free variables in the Generator are taken to be existentially quantified. Faster than the other aggregation predicates.

Meta-predicate with arguments: \texttt{findall}(?,goal,?).

Usage:

\begin{itemize}
  \item \textit{Calls should, and exit will be compatible with:}
    \begin{itemize}
      \item List is a list. \texttt{(basic_props:list/1)}
    \end{itemize}
  \item \textit{The following properties should hold at call time:}
    \begin{itemize}
      \item Template is any term. \texttt{(basic_props:term/1)}
      \item Generator is a term which represents a goal, i.e., an atom or a structure. \texttt{(basic_props:callable/1)}
    \end{itemize}
  \item \textit{The following properties hold upon exit:}
    \begin{itemize}
      \item Template is any term. \texttt{(basic_props:term/1)}
      \item List is a list. \texttt{(basic_props:list/1)}
    \end{itemize}
  \item \textit{The following properties hold globally:}
    \begin{itemize}
      \item Template is not further instantiated. \texttt{(basic_props:not_further_inst/2)}
    \end{itemize}
\end{itemize}

This predicate is understood natively by CiaoPP.

\texttt{findall}/4:

Meta-predicate with arguments: \texttt{findall}(?,goal,?,?).

Usage: \texttt{findall} (Template, Generator, List, Tail)

\begin{itemize}
  \item \textit{Description:} As \texttt{findall}/3, but returning in \texttt{Tail} the tail of \texttt{List}.
\end{itemize}

\texttt{findnsols}/4:

\texttt{findnsols} (N, Template, Generator, List)

As \texttt{findall}/3, but generating at most \texttt{N} solutions of \texttt{Generator}. Thus, the length of \texttt{List} will not be greater than \texttt{N}. If \texttt{N} is \textless 0, returns directly an empty list. This predicate is especially useful if \texttt{Generator} may have an infinite number of solutions.

Meta-predicate with arguments: \texttt{findnsols}(?,?,goal,?).

Usage:

\begin{itemize}
  \item \textit{Calls should, and exit will be compatible with:}
    \begin{itemize}
      \item List is a list. \texttt{(basic_props:list/1)}
    \end{itemize}
  \item \textit{The following properties should hold at call time:}
    \begin{itemize}
      \item \texttt{N} is an integer. \texttt{(basic_props:int/1)}
      \item Template is any term. \texttt{(basic_props:term/1)}
      \item Generator is a term which represents a goal, i.e., an atom or a structure. \texttt{(basic_props:callable/1)}
    \end{itemize}
  \item \textit{The following properties hold upon exit:}
    \begin{itemize}
      \item Template is any term. \texttt{(basic_props:term/1)}
      \item List is a list. \texttt{(basic_props:list/1)}
    \end{itemize}
  \item \textit{The following properties hold globally:}
    \begin{itemize}
      \item Template is not further instantiated. \texttt{(basic_props:not_further_inst/2)}
    \end{itemize}
\end{itemize}
**findnsols/5:**

*findnsols*(N, Template, Generator, List, Tail)

As *findnsols*/4, but returning in Tail the tail of List.

*Meta-predicate* with arguments: *findnsols*(?, ?, goal, ?, ?).

**Usage:**

- *The following properties should hold at call time:*
  
  - N is an integer.  
    
    (basic_props:int/1)
  
  - Template is any term.  
    
    (basic_props:term/1)
  
  - Generator is a term which represents a goal, i.e., an atom or a structure.  
    
    (basic_props:callable/1)

- *The following properties hold upon exit:*
  
  - Template is any term.  
    
    (basic_props:term/1)

- *The following properties hold globally:*
  
  - Template is not further instantiated.  
    
    (basic_props:not_further_inst/2)

**~/2:**

*Meta-predicate* with arguments: (?)~/goal.

**General properties:** _X ~ Y

- *The following properties hold globally:*
  
  - This predicate is understood natively by CiaoPP as call(Y).  
    
    (basic_props:native/2)

**Usage:** X ~ P

- *Description:* Existential quantification: X is existentially quantified in P. E.g., in A~p(A,B), A is existentially quantified. Used only within aggregation predicates. In all other contexts, simply, execute the procedure call P.
33 Dynamic predicates

Author(s): The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#75 (2003/12/19, 17:4:51 CET)

This module implements the assert/retract family of predicates to manipulate dynamic predicates.

The predicates defined in this module allow modification of the program as it is actually running. Clauses can be added to the program (asserted) or removed from the program (retracted). For these predicates, the argument which corresponds to the clause head must be instantiated to an atom or a compound term. The argument corresponding to the clause must be instantiated either to a term Head :- Body or, if the body part is empty, to Head. An empty body part is represented as true. Note that using this library is very detrimental to global analysis, and that for most uses the predicates listed in Chapter 25 [Fast/concurrent update of facts], page 137 suffice.

33.1 Usage and interface (dynamic)

- Library usage:
  :- use_module(library(dynamic)).

- Exports:
  - Predicates:
    asserta/1, asserta/2, assertz/1, assertz/2, assert/1, assert/2, retract/1, retractall/1, abolish/1, clause/2, clause/3, current_predicate/1, current_predicate/2, dynamic/1, data/1, wellformed_body/3.
  - Multifiles:
    do_on_abolish/1.

- Other modules used:
  - System library modules:
    prolog_sys.

33.2 Documentation on exports (dynamic)

asserta/1: PREDICATE

Meta-predicate with arguments: asserta(clause).
Usage: asserta(Clause)

- Description: The current instance of Clause is interpreted as a clause and is added to the current program. The predicate concerned must be dynamic. The new clause becomes the first clause for the predicate concerned. Any uninstantiated variables in Clause will be replaced by new private variables.
- The following properties should hold at call time:
  Clause is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)
asserta/2:  
Meta-predicate with arguments: asserta(clause,?).  
Usage: asserta(Clause, Ref)  
- Description: Like asserta/1. Ref is a unique identifier of the asserted clause.  
- The following properties should hold at call time:  
  Clause is currently a term which is not a free variable.  
  Ref is a free variable.  
- The following properties hold globally:  
  This predicate is understood natively by CiaoPP.  

assertz/1:  
Meta-predicate with arguments: assertz(clause).  
Usage: assertz(Clause)  
- Description: Like asserta/1, except that the new clause becomes the last clause for the predicate concerned.  
- The following properties should hold at call time:  
  Clause is currently a term which is not a free variable.  
- The following properties hold globally:  
  This predicate is understood natively by CiaoPP.  

assertz/2:  
Meta-predicate with arguments: assertz(clause,?).  
Usage: assertz(Clause, Ref)  
- Description: Like asserta/1. Ref is a unique identifier of the asserted clause.  
- The following properties should hold at call time:  
  Clause is currently a term which is not a free variable.  
  Ref is a free variable.  
- The following properties hold globally:  
  This predicate is understood natively by CiaoPP.  

assert/1:  
Meta-predicate with arguments: assert(clause).  
Usage: assert(Clause)  
- Description: Identical to assertz/1. Included for compatibility.  
- The following properties should hold at call time:  
  Clause is currently a term which is not a free variable.  
- The following properties hold globally:  
  This predicate is understood natively by CiaoPP.
assert/2: PREDICATE
Meta-predicate with arguments: assert(clause,?).
Usage: assert(Clause, Ref)
  Description: Identical to assertz/2. Included for compatibility.
  The following properties should hold at call time:
  Clause is currently a term which is not a free variable. (term_typing:nonvar/1)
  Ref is a free variable. (term_typing:var/1)
  The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

retract/1: PREDICATE
Meta-predicate with arguments: retract(clause).
Usage: retract(Clause)
  Description: The first clause in the program that matches Clause is erased. The predicate concerned must be dynamic.
  The predicate retract/1 may be used in a non-determinate fashion, i.e., it will successively retract clauses matching the argument through backtracking. If reactivated by backtracking, invocations of the predicate whose clauses are being retracted will proceed unaffected by the retracts. This is also true for invocations of clause for the same predicate. The space occupied by a retracted clause will be recovered when instances of the clause are no longer in use.
  The following properties should hold at call time:
  Clause is currently a term which is not a free variable. (term_typing:nonvar/1)
  The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

retractall/1: PREDICATE
Meta-predicate with arguments: retractall(fact).
Usage: retractall(Head)
  Description: Erase all clauses whose head matches Head, where Head must be instantiated to an atom or a compound term. The predicate concerned must be dynamic. The predicate definition is retained.
  The following properties should hold at call time:
  Head is currently a term which is not a free variable. (term_typing:nonvar/1)
  The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

abolish/1: PREDICATE
Meta-predicate with arguments: abolish(spec).
Usage: abolish(Spec)
  Description: Erase all clauses of the predicate specified by the predicate spec Spec. The predicate definition itself is also erased (the predicate is deemed undefined after execution of the abolish). The predicates concerned must all be user defined.
The following properties should hold at call time:
Spec is currently a term which is not a free variable. (term_typing:nonvar/1)

The following properties hold globally:
This predicate is understood natively by CiaoPP. (basic_props:native/1)

clause/2:
Meta-predicate with arguments: clause(fact,?).
Usage: clause(Head, Body)
- Description: The clause 'Head :- Body' exists in the current program. The predicate concerned must be dynamic.

The following properties should hold at call time:
Head is currently a term which is not a free variable. (term_typing:nonvar/1)

The following properties hold globally:
This predicate is understood natively by CiaoPP. (basic_props:native/1)

clause/3:
Like clause(Head,Body), plus the clause is uniquely identified by Ref.
Meta-predicate with arguments: clause(fact,?,?,?).
General properties:
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 1: clause(Head, Body, Ref)
- Description: Head must be instantiated to an atom or a compound term.
- The following properties should hold at call time:
  Head is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: clause(Head, Body, Ref)
- Description: Ref must be instantiated to a valid identifier.
- The following properties should hold at call time:
  Ref is currently a term which is not a free variable. (term_typing:nonvar/1)

current_predicate/1:
Usage: current_predicate(Spec)
- Description: A predicate in the current module is named Spec.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_predicate/2:
Usage: current_predicate(Spec, Module)
- Description: A predicate in Module is named Spec. Module never is an engine module.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)
**dynamic/1:**

`dynamic Spec`

`Spec` is of the form `F/A`. The predicate named `F` with arity `A` is made dynamic in the current module at runtime (useful for predicate names generated on-the-fly). If the predicate functor name `F` is uninstatiated, a new, unique, predicate name is generated at runtime.

Usage: `dynamic Spec`

- *The following properties should hold at call time:*

  `Spec` is currently a term which is not a free variable. *(term_typing:nonvar/1)*

**data/1:**

`data Spec`

`Spec` is of the form `F/A`. The predicate named `F` with arity `A` is made data in the current module at runtime (useful for predicate names generated on-the-fly). If the predicate functor name `F` is uninstatiated, a new, unique, predicate name is generated at runtime.

Usage: `data Spec`

- *The following properties should hold at call time:*

  `Spec` is currently a term which is not a free variable. *(term_typing:nonvar/1)*

**wellformed_body/3:**

`wellformed_body(BodyIn, Env, BodyOut)`

`BodyIn` is a well-formed clause body. `BodyOut` is its counterpart with no single-variable meta-goals (i.e., with `call(X)` for `X`). `Env` denotes if global cuts are admissible in `BodyIn` (+ if they are, − if they are not).

### 33.3 Documentation on multfiles (dynamic)

**do_on_abolish/1:**

`do_on_abolish(Head)`

A hook predicate which will be called when the definition of the predicate of `Head` is abolished.

The predicate is *multifile.*
Chapter 34: Term input

Author(s): First versions from SICStus 0.6 code; additional changes and documentation by Daniel Cabeza and Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#78 (2003/12/20, 1:52:48 CET)

This module provides facilities to read terms in Prolog syntax. This is very convenient in many cases (and not only if you are writing a Prolog compiler), because Prolog terms are easy to write and can convey a lot of information in a human-readable fashion.

34.1 Usage and interface (read)

- Library usage:
  ```prolog
  :- use_module(library(read)).
  ```

- Exports:
  - Predicates:
    - `read/1`, `read/2`, `read_term/2`, `read_term/3`, `read_top_level/3`, `second_prompt/2`.
  - Multifiles:
    - `define_flag/3`.

- Other modules used:
  - System library modules:
    - `tokenize`, `operators`, `lists`.

34.2 Documentation on exports (read)

read/1: PREDICATE

```
read(Term)
```

Like `read(Stream, Term)` with `Stream` associated to the current input stream.

read/2: PREDICATE

```
Usage: read(Stream, Term)
```

- **Description:** The next term, delimited by a full-stop (i.e., a . followed by either a space or a control character), is read from `Stream` and is unified with `Term`. The syntax of the term must agree with current operator declarations. If the end of `Stream` has been reached, `Term` is unified with the term `end_of_file`. Further calls to `read/2` for the same stream will then cause an error, unless the stream is connected to the terminal (in which case a prompt is opened on the terminal).

- **The following properties should hold at call time:**
  - `Stream` is currently a term which is not a free variable. \(\text{term_typing:nonvar/1}\)
  - **The following properties hold upon exit:**
    - `Stream` is an open stream. \(\text{streams_basic:stream/1}\)
    - `Term` is any term. \(\text{basic Props:term/1}\)
read_term/2:
Usage: read_term(Term, Options)
- Description: Like read_term/3, but reading from the current input
- The following properties should hold at call time:
  Options is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold upon exit:
  Term is any term.
  Options is a list of read_options.

read_term/3:
Usage: read_term(Stream, Term, Options)
- Description: Reads a Term from Stream with the ISO-Prolog Options. These options can control the behavior of read term (see read_option/1).
- The following properties should hold at call time:
  Stream is currently a term which is not a free variable. (term_typing:nonvar/1)
  Options is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold upon exit:
  Stream is an open stream. (streams_basic:stream/1)
  Term is any term. (basic_props:term/1)
  Options is a list of read_options. (basic_props:list/2)

read_top_level/3:
read_top_level(Stream, Data, Variables)
Predicate used to read in the Top Level.

second_prompt/2:
Usage: second_prompt(Old, New)
- Description: Changes the prompt (the second prompt, as opposed to the first one, used by the toplevel) used by read/2 and friends to New, and returns the current one in Old.
- The following properties should hold upon exit:
  Old is currently instantiated to an atom. (term_typing:atom/1)
  New is currently instantiated to an atom. (term_typing:atom/1)

34.3 Documentation on multfiles (read)

define_flag/3:
Defines flags as follows:

```prolog
define_flag(read_hiord,[on,off],off).
```
(See Chapter 24 [Changing system behaviour and various flags], page 131).
If flag is on (it is off by default), a variable followed by a parenthesized lists of arguments is read as a \texttt{call/N} term, except if the variable is anonymous, in which case it is read as an anonymous predicate abstraction head. For example, \texttt{P(X)} is read as \texttt{call(P,X)} and \texttt{_(X,Y)} as \texttt{'(X,Y)'}.
The predicate is \textit{multifile}.
34.4 Documentation on internals (read)

**read_option/1:**

**Usage:** read_option(Option)

- **Description:** Option is an allowed `read_term/[2,3]` option. These options are:
  
  - `read_option(variables(_V)).`
  - `read_option(variable_names(_N)).`
  - `read_option(singletons(_S)).`
  - `read_option(lines(_StartLine,_EndLine)).`
  - `read_option(dictionary(_Dict)).`

  They can be used to return the singleton variables in the term, a list of variables, etc.

- **The following properties should hold upon exit:**
  
  Option is currently instantiated to an atom. (term_typing:atom/1)

34.5 Known bugs and planned improvements (read)

- The comma cannot be redefined as an operator, it is defined in any case as `op(1000, xfy,[',']).`
35 Term output

Author(s): Adapted from shared code written by Richard A. O'Keefe. Changes by Mats Carlsson, Daniel Cabeza, Manuel Hermenegildo, and Manuel Carro..

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#290 (2004/2/13, 20:20:3 CET)

This library provides different predicates for term output, additional to the kernel predicates display/1 - display/2 and displayq/1-displayq/2. All the predicates defined in ISO-Prolog are included, plus other traditionally provided by Prolog Implementations. Output predicates are provided in two versions: one that uses the current output stream and other in which the stream is specified explicitly, as an additional first argument.

35.1 Usage and interface (write)

- Library usage:
  :=- use_module(library(write)).
- Exports:
  - Predicates:
    write_term/3, write_term/2, write/2, write/1, writeq/2, writeq/1, write_canonical/2, write_canonical/1, print/2, print/1, write_list1/1, portray_clause/2, portray_clause/1, numbervars/3, prettyvars/1, printable_char/1.
  - Properties:
    write_option/1.
  - Multifiles:
    define_flag/3, portray_attribute/2, portray/1.
- Other modules used:
  - System library modules:
    operators, sort.

35.2 Documentation on exports (write)

write_term/3:

Usage: write_term(Stream, Term, OptList)

- Description: Outputs the term Term to the stream Stream, with the list of write-options OptList. See write_option/1 type for default options.
- The following properties should hold at call time:
  OptList is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold upon exit:
  Stream is an open stream. (streams_basic:stream/1)
  Term is any term. (basic_props:term/1)
  OptList is a list of write_options. (basic_props:list/2)
- The following properties hold globally:
  Stream is not further instantiated. (basic_props:not_further_inst/2)
**write_term/2:**

Usage: `write_term(Term, OptList)`

- Description: Behaves like `current_output(S), write_term(S,Term,OptList)`.
- The following properties should hold at call time:
  - `OptList` is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties hold upon exit:
  - `Term` is any term. (basic_props:term/1)
  - `OptList` is a list of write_options. (basic_props:list/2)

**write_option/1:**

`Opt` is a valid write option which affects the predicate `write_term/3` and similar ones. Possible write_options are:

- **quoted**(bool): If `bool` is true, atoms and functors that can't be read back by `read_term/3` are quoted, if it is false, each atom and functor is written as its name. Default value is false.
- **ignore_ops**(flag): If `flag` is true, each compound term is output in functional notation, if it is ops, curly bracketed notation and list notation is enabled when outputing compound terms, if it is false, also operator notation is enabled when outputing compound terms. Default value is false.
- **numbervars**(bool): If `bool` is true, a term of the form `$VAR$(N) where N is an integer, is output as a variable name consisting of a capital letter possibly followed by an integer, a term of the form `$VAR$(Atom) where Atom is an atom, as this atom (without quotes), and a term of the form `$VAR$(String) where String is a character string, as the atom corresponding to this character string. See predicates numbervars/3 and prettyvars/1. If `bool` is false this cases are not treated in any special way. Default value is false.
- **portrayed**(bool): If `bool` is true, then call multifile predicates portray/1 and portray_attribute/2, to provide the user handlers for pretty printing some terms. portray_attribute/2 is called whenever an attributed variable is to be printed, portray/1 is called whenever a non-variable term is to be printed. If either call succeeds, then it is assumed that the term has been output, else it is printed as usual. If `bool` is false, these predicates are not called. Default value is false. This option is set by the toplevel when writing the final values of variables, and by the debugging package when writing the goals in the tracing messages. Thus you can vary the forms of these messages if you wish.
- **max_depth**(depth): `depth` is a positive integer or cero. If it is positive, it denotes the depth limit on printing compound terms. If it is cero, there is no limit. Default value is 0 (no limit).
- **priority**(prio): `prio` is an integer between 1 and 1200. If the term to be printed has higher priority than `prio`, it will be printed parenthesized. Default value is 1200 (no term parenthesized).

Usage: `write_option(Opt)`
- Description: `Opt` is a valid write option.

**write/2:**

Usage: `write(Stream, Term)`
- **Description:** Behaves like \texttt{write\_term(Stream, Term, \{numbervars(true)\})}.

- **The following properties hold upon exit:**
  - \texttt{Stream} is an open stream.  \hphantom{a}
    \texttt{(streams\_basic:stream/1)}
  - \texttt{Term} is any term.  \hphantom{a}
    \texttt{(basic\_props:term/1)}

- **The following properties hold globally:**
  - \texttt{Stream} is not further instantiated.  \hphantom{a}
    \texttt{(basic\_props:not\_further\_inst/2)}

\textbf{write/1:}

\textbf{Usage:} \texttt{write(Term)}

- **Description:** Behaves like \texttt{current\_output(S), write(S,Term)}.

- **The following properties hold upon exit:**
  - \texttt{Term} is any term.  \hphantom{a}
    \texttt{(basic\_props:term/1)}

\textbf{writeq/2:}

\textbf{Usage:} \texttt{writeq(Stream, Term)}

- **Description:** Behaves like \texttt{write\_term(Stream, Term, \{quoted(true), numbervars(true)\})}.

- **The following properties hold upon exit:**
  - \texttt{Stream} is an open stream.  \hphantom{a}
    \texttt{(streams\_basic:stream/1)}
  - \texttt{Term} is any term.  \hphantom{a}
    \texttt{(basic\_props:term/1)}

- **The following properties hold globally:**
  - \texttt{Stream} is not further instantiated.  \hphantom{a}
    \texttt{(basic\_props:not\_further\_inst/2)}

\textbf{writeq/1:}

\textbf{Usage:} \texttt{writeq(Term)}

- **Description:** Behaves like \texttt{current\_output(S), writeq(S,Term)}.

- **The following properties hold upon exit:**
  - \texttt{Term} is any term.  \hphantom{a}
    \texttt{(basic\_props:term/1)}

\textbf{write\_canonical/2:}

\textbf{Usage:} \texttt{write\_canonical(Stream, Term)}

- **Description:** Behaves like \texttt{write\_term(Stream, Term, \{quoted(true), ignore\_ops(true)\})}. The output of this predicate can always be parsed by \texttt{read\_term/2} even if the term contains special characters or if operator declarations have changed.

- **The following properties hold upon exit:**
  - \texttt{Stream} is an open stream.  \hphantom{a}
    \texttt{(streams\_basic:stream/1)}
  - \texttt{Term} is any term.  \hphantom{a}
    \texttt{(basic\_props:term/1)}

- **The following properties hold globally:**
  - \texttt{Stream} is not further instantiated.  \hphantom{a}
    \texttt{(basic\_props:not\_further\_inst/2)}
write_canonical/1:
Usage: write_canonical(Term)
- Description: Behaves like current_output(S), write_canonical(S,Term).
- The following properties hold upon exit:
  Term is any term.

print/2:
Usage: print(Stream, Term)
- Description: Behaves like write_term(Stream, Term, [numbervars(true), portrayed(true)]).
- The following properties hold upon exit:
  Stream is an open stream.
  Term is any term.
- The following properties hold globally:
  Stream is not further instantiated.

print/1:
Usage: print(Term)
- Description: Behaves like current_output(S), print(S,Term).
- The following properties hold upon exit:
  Term is any term.

write_list1/1:
Usage:
- Description: Writes a list to current output one element in each line.
- Calls should, and exit will be compatible with:
  Arg1 is a list.

portray_clause/2:
Usage: portray_clause(Stream, Clause)
- Description: Outputs the clause Clause onto Stream, pretty printing its variables and using indentation, including a period at the end. This predicate is used by listing/0.
- The following properties hold upon exit:
  Stream is an open stream.
  Clause is any term.
- The following properties hold globally:
  Stream is not further instantiated.
**portray_clause/1:**

Usage: portray_clause(Clause)

- *Description:* Behaves like `current_output(S), portray_clause(S,Term)`.
- *The following properties hold upon exit:*
  
  Clause is any term. (basic_props:term/1)

**numbervars/3:**

Usage: numbervars(Term, N, M)

- *Description:* Unifies each of the variables in term Term with a term of the form `$VAR'(I)` where I is an integer from N onwards. M is unified with the last integer used plus 1. If the resulting term is output with a write option `numbervars(true)`, in the place of the variables in the original term will be printed a variable name consisting of a capital letter possibly followed by an integer. When N is 0 you will get the variable names A, B, ..., Z, A1, B1, etc.
- *The following properties should hold at call time:*
  
  N is currently a term which is not a free variable. (term_typing:nonvar/1)
  
  *The following properties hold upon exit:*
  
  Term is any term. (basic_props:term/1)
  
  N is an integer. (basic_props:int/1)
  
  M is an integer. (basic_props:int/1)

**prettyvars/1:**

Usage: prettyvars(Term)

- *Description:* Similar to `numbervars(Term,0,_),` except that singleton variables in Term are unified with `$VAR'(,')`, so that when the resulting term is output with a write option `numbervars(true)`, in the place of singleton variables _ is written. This predicate is used by `portray_clause/2`.
- *The following properties hold upon exit:*
  
  Term is any term. (basic_props:term/1)

**printable_char/1:**

Usage: printable_char(Char)

- *Description:* Char is the code of a character which can be printed.
- *The following properties should hold at call time:*
  
  Char is currently a term which is not a free variable. (term_typing:nonvar/1)
  
  *The following properties should hold at call time:*
  
  Char is currently instantiated to a number. (term_typing:number/1)
35.3 Documentation on multifiles \textit{(write)}

\textbf{define_flag/3:} \hspace{1cm} \textbf{PREDICATE}

Defines flags as follows:

\begin{verbatim}
define_flag(write_strings,[on,off],off).
\end{verbatim}

(See Chapter 24 [Changing system behaviour and various flags], page 131).

If flag is \textit{on}, lists which may be written as strings are.

The predicate is \textit{multifile}.

\textbf{portray_attribute/2:} \hspace{1cm} \textbf{PREDICATE}

The predicate is \textit{multifile}.

\textbf{Usage:} \texttt{portray_attribute(Attr, Var)}

\begin{itemize}
  \item \textit{Description: A user defined predicate.} When an attributed variable \texttt{Var} is about to be printed, this predicate receives the variable and its attribute \texttt{Attr}. The predicate should either print something based on \texttt{Attr} or \texttt{Var}, or do nothing and fail. In the latter case, the default printer (\texttt{write/1}) will print the attributed variable like an unbound variable, e.g. \texttt{_673}.
  \item \textit{The following properties should hold at call time:}
    \begin{itemize}
      \item \texttt{Attr} is currently a term which is not a free variable. \hspace{1cm} \texttt{(term_typing:nonvar/1)}
      \item \texttt{Var} is a free variable. \hspace{1cm} \texttt{(term_typing:var/1)}
    \end{itemize}
\end{itemize}

\textbf{portray/1:} \hspace{1cm} \textbf{PREDICATE}

The predicate is \textit{multifile}.

\textbf{Usage:} \texttt{portray(Term)}

\begin{itemize}
  \item \textit{Description: A user defined predicate.} This should either print the \texttt{Term} and succeed, or do nothing and fail. In the latter case, the default printer (\texttt{write/1}) will print the \texttt{Term}.
\end{itemize}
36 Defining operators

Author(s): Adapted from SICStus 0.6 code; modifications and documentation by Daniel Cabeza and Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#287 (2004/2/13, 18:59:4 CET)

Operators allow writing terms in a more clear way than the standard functional notation. Standard operators in Ciao are defined by this predicate (but note that the compiler itself defines more operators at compile time):

\[
\text{standard_ops :-}
\begin{align*}
\text{op}(1200, &\text{xfx}, [-]), \\
\text{op}(1200, &\text{fx}, [\text{:-}, ?-]), \\
\text{op}(1100, &\text{xfy}, []), \\
\text{op}(1050, &\text{xfy}, [\rightarrow]), \\
\text{op}(1000, &\text{xfy}, ['']), \\
\text{op}(900, &\text{fy}, [+]), \\
\text{op}(700, &\text{xfx}, [=, \text{\,=-, \text{\,=}%, @<, @>, @=<, @>=, \text{\,=\,=}%, \text{\,=}%, <, <=, >, >=])], \\
\text{op}(550, &\text{xfx}, []), \\
\text{op}(500, &\text{yfx}, [+,-,\text{\,=/, \text{\,=, \#}]}, \\
\text{op}(500, &\text{fy}, [+,-,\text{\,=, \#}]}, \\
\text{op}(400, &\text{yfx}, [*,\text{\,=/, rem, mod, <>, >=])}, \\
\text{op}(200, &\text{fy}, [+,-,\text{\,=, \#}]}, \\
\text{op}(200, &\text{xfx}, [**])}, \\
\text{op}(200, &\text{xfy}, [\text{\,-}])}. \\
\end{align*}
\]

36.1 Usage and interface (operators)

- Library usage:
  \[- \text{use_module(library(operators)).} \]

- Exports:
  - Predicates:
    - \text{op}/3, \text{current_op}/3, \text{current_prefixop}/3, \text{current_infixop}/4, \text{current_postfixop}/3.

36.2 Documentation on exports (operators)

\text{op}/3: PREDICATE

\begin{align*}
\text{op}(\text{Precedence}, \text{Type}, \text{Name})
\end{align*}

Declares the atom \text{Name} to be an operator of the stated \text{Type} and \text{Precedence} (0 \leq \text{Precedence} \leq 1200). \text{Name} may also be a list of atoms in which case all of them are declared to be operators. If \text{Precedence} is 0 then the operator properties of \text{Name} (if any) are cancelled. Note that, unlike in ISO-Prolog, it is allowed to define two operators with the same name, one infix and the other postfix.

Usage: \text{\includegraphics[width=0.1\textwidth]{ISO.png}}
The following properties should hold at call time:

- **Precedence** is an integer. (basic_props:int/1)
- **Type** specifies the type and associativity of an operator. (basic_props:operator_specifier/1)
- **Name** is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

The following properties hold globally:

- This predicate is understood natively by CiaoPP. (basic_props:native/1)

---

**current_op/3:**

```
current_op(Precedence, Type, Op)
```

The atom **Op** is currently an operator of type **Type** and precedence **Precedence**. Neither **Op** nor the other arguments need be instantiated at the time of the call; i.e., this predicate can be used to generate as well as to test.

**Usage:**

- **Calls should, and exit will be compatible with:**
  - **Precedence** is an integer. (basic_props:int/1)
  - **Type** specifies the type and associativity of an operator. (basic_props:operator_specifier/1)
  - **Op** is an atom. (basic_props:atm/1)

- **The following properties hold upon exit:**
  - **Precedence** is an integer. (basic_props:int/1)
  - **Type** specifies the type and associativity of an operator. (basic_props:operator_specifier/1)
  - **Op** is an atom. (basic_props:atm/1)

- **The following properties hold globally:**
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

---

**current_prefixop/3:**

```
current_prefixop(Op, Less, Precedence)
```

Similar to **current_op/3**, but it concerns only the prefix operators. It returns **only one solution**. Not a predicate for general use.

---

**current_infixop/4:**

```
current_infixop(Op, LeftLess, Prec, RightLess)
```

Similar to **current_op/3**, but it concerns only infix operators. It returns **only one solution**. Not a predicate for general use.

---

**current_postfixop/3:**

```
current_postfixop(Op, Less, Precedence)
```

Similar to **current_op/3**, but it concerns only the postfix operators. It returns **only one solution**. Not a predicate for general use.
37 The Iso Byte Char module

Author(s): The CLIP Group, Daniel Cabeza, Documentation written by Edison Mera, based on ISO Prolog standard. Minor mods by M. Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#217 (2003/12/21, 15:33:54 CET)

This module provides some basic predicates according to the ISO specification of byte and char manipulation.

37.1 Usage and interface (iso_byte_char)

- Library usage:
  ```prolog```
  ```?- use_module(library(iso_byte_char)).```  
- Exports:
  
  - Predicates:
    - `char_code/2`
    - `atom_chars/2`
    - `number_chars/2`
    - `get_byte/1`
    - `get_byte/2`
    - `peek_byte/1`
    - `peek_byte/2`
    - `put_byte/1`
    - `put_byte/2`
    - `get_char/1`
    - `get_char/2`
    - `peek_char/1`
    - `peek_char/2`
    - `put_char/1`
    - `put_char/2`

37.2 Documentation on exports (iso_byte_char)

**char_code/2:**

```prolog```

`char_code(Char, Code)`

Succeeds iff the character code of the one char atom `Char` is `Code`.

Usage 1:

- **Calls should, and exit will be compatible with:**
  - `Code` is an integer which is a character code.  
    (basic_props:character_code/1)
- **The following properties should hold at call time:**
  - `Char` is an atom.  
    (basic_props:atm/1)
- **The following properties hold upon exit:**
  - `Code` is an integer which is a character code.  
    (basic_props:character_code/1)

Usage 2:

- **Calls should, and exit will be compatible with:**
  - `Code` is an integer which is a character code.  
    (basic_props:character_code/1)
- **The following properties should hold at call time:**
  - `Char` is a free variable.  
    (term_typing:var/1)
- **The following properties hold upon exit:**
  - `Char` is an atom.  
    (basic_props:atm/1)
  - `Code` is an integer which is a character code.  
    (basic_props:character_code/1)
atom_chars/2:

atom_chars(Atom, Chars)

Succeeds iff Chars is a list whose elements are the one-char atoms whose names are the successive characters of the name of atom Atom

Usage 1:

- Calls should, and exit will be compatible with:
  Chars is a list of character_codes. (basic_props:list/2)
- The following properties should hold at call time:
  Atom is an atom. (basic_props:atm/1)
- The following properties hold upon exit:
  Chars is a list of character_codes. (basic_props:list/2)

Usage 2:

- The following properties should hold at call time:
  Atom is a free variable. (term_typing:var/1)
  Chars is a list of character_codes. (basic_props:list/2)
- The following properties hold upon exit:
  Atom is an atom. (basic_props:atm/1)

count_chars/2:

count_chars(Number, Chars)

Success iff Chars is a list whose elements are the one-char atoms corresponding to a character sequence of Number which could be output

Usage 1:

- Calls should, and exit will be compatible with:
  Chars is a list of atms. (basic_props:list/2)
- The following properties should hold at call time:
  Number is a number. (basic_props:num/1)
- The following properties hold upon exit:
  Chars is a list of atms. (basic_props:list/2)

Usage 2:

- The following properties should hold at call time:
  Number is a free variable. (term_typing:var/1)
  Chars is a list of atms. (basic_props:list/2)
- The following properties hold upon exit:
  Number is a number. (basic_props:num/1)

get_byte/1:

Usage:

- Description: Same as get_byte/2, but use the current input.
- Calls should, and exit will be compatible with:
  Arg1 is an integer. (basic_props:int/1)
- The following properties hold upon exit:
  Arg1 is an integer. (basic_props:int/1)
get_byte/2:  
get_byte(Stream, Byte)  
Is true if Byte unifies with the next byte to be input from the target Stream.  
Usage:  
   - Calls should, and exit will be compatible with:  
     Byte is an integer.  
     (basic Props:int/1)  
   - The following properties should hold at call time:  
     Stream is an open stream.  
     (streams_basic:stream/1)  
   - The following properties hold upon exit:  
     Stream is an open stream.  
     Byte is an integer.  
     (basic Props:int/1)  
   - The following properties hold globally:  
     Stream is not further instantiated.  
     (basic Props:not further inst/2)  

peek_byte/1:  
Usage:  
   - Description: Same as peek_byte/2, but use the current input.  
   - Calls should, and exit will be compatible with:  
     Arg1 is an integer.  
     (basic Props:int/1)  
   - The following properties hold upon exit:  
     Arg1 is an integer.  
     (basic Props:int/1)  

peek_byte/2:  
peek_byte(Stream, Byte)  
Is true if Byte unifies with the next byte to be input from the target Stream.  
Usage:  
   - Calls should, and exit will be compatible with:  
     Byte is an integer.  
     (basic Props:int/1)  
   - The following properties should hold at call time:  
     Stream is an open stream.  
     (streams_basic:stream/1)  
   - The following properties hold upon exit:  
     Stream is an open stream.  
     Byte is an integer.  
     (basic Props:int/1)  
   - The following properties hold globally:  
     Stream is not further instantiated.  
     (basic Props:not further inst/2)  

put_byte/1:  
Usage:  
   - Description: Same as put_byte/2, but use the current input.  
   - The following properties should hold at call time:  
     Arg1 is an integer.  
     (basic Props:int/1)
**put_byte/2:**

```prolog
put_byte(Stream, Byte)
```

Is true. Procedurally, `put_byte/2` is executed as follows:

a) Outputs the byte `Byte` to the target stream.

b) Changes the stream position of the target stream to take account of the byte which has been output.

c) The goal succeeds.

**Usage:**

- The following properties should hold at call time:
  - Stream is an open stream.  
    (streams_basic:stream/1)
  - Byte is an integer.  
    (basic_props:int/1)

- The following properties hold upon exit:
  - Stream is an open stream.  
    (streams_basic:stream/1)

- The following properties hold globally:
  - Stream is not further instantiated.  
    (basic_props:not_further_inst/2)

**get_char/1:**

```prolog
get_char(Stream, Arg1)
```

Is true if `Arg1` unifies with the next character to be input from the target `Stream`.

**Usage:**

- Description: Same as `get_char/2`, but use the current input.

- Calls should, and exit will be compatible with:
  - `Arg1` is an atom.  
    (basic_props:atm/1)

**get_char/2:**

```prolog
get_char(Stream, Char)
```

Is true if `Char` unifies with the next character to be input from the target `Stream`.

**Usage:**

- Calls should, and exit will be compatible with:
  - `Char` is an atom.  
    (basic_props:atm/1)

- The following properties should hold at call time:
  - Stream is an open stream.  
    (streams_basic:stream/1)

- The following properties hold upon exit:
  - Stream is an open stream.  
    (streams_basic:stream/1)
  - `Char` is an atom.  
    (basic_props:atm/1)

- The following properties hold globally:
  - Stream is not further instantiated.  
    (basic_props:not_further_inst/2)

**peek_char/1:**

```prolog
peek_char(Stream)
```

**Usage:**

- Description: Similar to `peek_code/1`, but using char instead of code.
– Calls should, and exit will be compatible with:
  Arg1 is an atom.  

– The following properties hold upon exit:
  Arg1 is an atom.  

peek_char/2:
Usage:  
– Description: Similar to peek_code/2, but using char instead of code.
– Calls should, and exit will be compatible with:
  Arg2 is an atom.  

– The following properties should hold at call time:
  Arg1 is an open stream.  

– The following properties hold upon exit:
  Arg1 is an open stream.  
  Arg2 is an atom.  

– The following properties hold globally:
  Arg1 is not further instantiated.  

put_char/1:
Usage:  
– Description: Similar to put_code/1, but using char instead of code.
– The following properties should hold at call time:
  Arg1 is an atom.  

put_char/2:
Usage:  
– Description: Similar to put_code/2, but using char instead of code.
– The following properties should hold at call time:
  Arg1 is an open stream.  
  Arg2 is an atom.  

– The following properties hold upon exit:
  Arg1 is an open stream.  

– The following properties hold globally:
  Arg1 is not further instantiated.  


38 Miscellaneous ISO Prolog predicates

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#199 (2004/2/17, 17:20:14 CET)
This module implements some miscellaneous ISO Prolog predicates.

38.1 Usage and interface (iso_misc)

- Library usage:
  :- use_module(library(iso_misc)).

- Exports:
  - Predicates:
    
  - Other modules used:
    

38.2 Documentation on exports (iso_misc)

\=/2:
  \ X \= Y
  X and Y are not unifiable.

once/1:
  once(G)
  Finds the first solution of goal G (if any). once/1 behaves as call/1, except that no further solutions are explored on backtracking.
  Meta-predicate with arguments: once(goal).

compound/1:
  compound(T)
  T is currently instantiated to a compound term.

sub_atom/5:
  sub_atom(Atom, Before, Length, After, Sub_atom)
  Is true iff atom Atom can be broken into three pieces, AtomL, Sub_atom and AtomR such that Before is the number of characters of the name of AtomL, Length is the number of characters of the name of Sub_atom and After is the number of characters of the name of AtomR.
**unify_with_occurs_check/2:**

```prolog
unify_with_occurs_check(X, Y)
```

Attempts to compute and apply a most general unifier of the two terms X and Y. Is true if and only if X and Y are unifiable.
39 Incomplete ISO Prolog predicates

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#141 (2003/12/31, 11:56:25 CET)
This module implements some ISO Prolog predicates, but that are not complete yet.

39.1 Usage and interface (iso_incomplete)

- Library usage:
  :- use_module(library(iso_incomplete)).
- Exports:
  - Predicates:
    close/2, stream_property/2.

39.2 Documentation on exports (iso_incomplete)

close/2:  
No further documentation available for this predicate.

stream_property/2:  
No further documentation available for this predicate.
PART IV - Classic Prolog library (classic)

Author(s): The CLIP Group.

This part documents some Ciao libraries which provide additional predicates and functionalities that, despite not being in the ISO standard, are present in many popular Prolog systems. This includes definite clause grammars (DCGs), “Quintus-style” internal database, list processing predicates, DEC-10 Prolog-style input/output, formatted output, dynamic loading of modules, activation of operators at run-time, etc.
40 Definite clause grammars

Author(s): The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#86 (2003/12/21, 2:3:40 CET)

This library package allows the use of DCGs (Definite Clause Grammars) [Col78,PW80] in a Ciao module/program.

Definite clause grammars are an extension of the well-known context-free grammars. Prolog's grammar rules provide a convenient notation for expressing definite clause grammars. A DCG rule in Prolog takes the general form

\[ \text{head} \rightarrow \text{body}. \]

meaning “a possible form for head is body”. Both body and head are sequences of one or more items linked by the standard Prolog conjunction operator ",".

Definite clause grammars extend context-free grammars in the following ways:

1. A non-terminal symbol may be any Prolog term (other than a variable or number).
2. A terminal symbol may be any Prolog term. To distinguish terminals from non-terminals, a sequence of one or more terminal symbols is written within a grammar rule as a Prolog list. An empty sequence is written as the empty list \([\ ]\). If the terminal symbols are ASCII character codes, such lists can be written (as elsewhere) as strings. An empty sequence is written as the empty list, \([\ ]\) or "".
3. Extra conditions, in the form of Prolog procedure calls, may be included in the right-hand side of a grammar rule. Such procedure calls are written enclosed in \(\{\}\) brackets.
4. The left-hand side of a grammar rule consists of a non-terminal, optionally followed by a sequence of terminals (again written as a Prolog list).
5. Alternatives may be stated explicitly in the right-hand side of a grammar rule, using the disjunction operator ",", or, also, as traditionally in Prolog, using \(|\) (which is treated specially when this package is loaded).
6. The cut symbol may be included in the right-hand side of a grammar rule, as in a Prolog clause. The cut symbol does not need to be enclosed in \(\{\}\) brackets.

As an example, here is a simple grammar which parses an arithmetic expression (made up of digits and operators) and computes its value.

\[ \begin{align*}
\text{expr}(Z) & \rightarrow \text{term}(X), \text{"+"}, \text{expr}(Y), \{Z \text{ is } X + Y\}. \\
\text{expr}(Z) & \rightarrow \text{term}(X), \text{"-"}, \text{expr}(Y), \{Z \text{ is } X - Y\}. \\
\text{expr}(X) & \rightarrow \text{term}(X).
\end{align*} \]

\[ \begin{align*}
\text{term}(Z) & \rightarrow \text{number}(X), \text{"*"}, \text{term}(Y), \{Z \text{ is } X \times Y\}. \\
\text{term}(Z) & \rightarrow \text{number}(X), \text{"/"}, \text{term}(Y), \{Z \text{ is } X / Y\}. \\
\text{term}(X) & \rightarrow \text{number}(Z).
\end{align*} \]

\[ \begin{align*}
\text{number}(C) & \rightarrow \text{"+"}, \text{number}(C). \\
\text{number}(C) & \rightarrow \text{"-"}, \text{number}(X), \{C \text{ is } -X\}. \\
\text{number}(X) & \rightarrow [C], \{0'0=<C, C=<0'9, X \text{ is } C - 0'0\}.
\end{align*} \]

In the last rule, \(C\) is the ASCII code of some digit.

The query

\[- \text{expr}(Z, \text{"-2+3*5+1"}, [\]). \]

will compute \(Z=14\). The two extra arguments are explained below.

Now, in fact, grammar rules are merely a convenient “syntactic sugar” for ordinary Prolog clauses. Each grammar rule takes an input string, analyses some initial portion, and produces
the remaining portion (possibly enlarged) as output for further analysis. The arguments required for the input and output strings are not written explicitly in a grammar rule, but the syntax implicitly defines them. We now show how to translate grammar rules into ordinary clauses by making explicit the extra arguments.

A rule such as
\[ p(X) \rightarrow q(X). \]
translates into
\[ p(X, S0, S) :- q(X, S0, S). \]

If there is more than one non-terminal on the right-hand side, as in
\[ p(X, Y) \rightarrow q(X), r(X, Y), s(Y). \]
then corresponding input and output arguments are identified, as in
\[ p(X, Y, S0, S) :- q(X, S0, S1), r(X, Y, S1, S2), r(Y, S2, S). \]

Terminals are translated using the built-in predicate \( 'C'/3 \) (this predicate is not normally useful in itself; it has been given the name \( 'C' \) simply to avoid using up a more useful name). Then, for instance
\[ p(X) \rightarrow [\text{go}, \text{to}], q(X), [\text{stop}]. \]
is translated by
\[ p(X, S0, S) :- 'C'(S0, \text{go}, S1), 'C'(S1, \text{to}, S2), q(X, S2, S3), 'C'(S3, \text{stop}, S). \]

Extra conditions expressed as explicit procedure calls naturally translate as themselves, e.g.
\[ p(X) \rightarrow [X], \{\text{integer}(X), X>0\}, q(X). \]
translates to
\[ p(X, S0, S) :- 'C'(S0, X, S1), \text{integer}(X), X>0, q(X, S1, S). \]

Similarly, a cut is translated literally.

Terminals on the left-hand side of a rule translate into an explicit list in the output argument of the main non-terminal, e.g.
\[ \text{is}(N), [\text{not}] \rightarrow [\text{aint}]. \]
becomes
\[ \text{is}(N, S0, [\text{not}|S]) :- 'C'(S0, \text{aint}, S). \]

Disjunction has a fairly obvious translation, e.g.
\[ \text{args}(X, Y) \rightarrow (\text{dir}(X), [\text{to}], \text{indir}(Y) ; \text{indir}(Y), \text{dir}(X) \). \]

translates to
Chapter 40: Definite clause grammars

40.1 Usage and interface (dcg)

- **Library usage:**
  
  `- use_package(dcg).

  or

  `- module(...,...,[dcg]).

```prolog
args(X, Y, S0, S) :-
    (   dir(X, S0, S1),
        'C'(S1, to, S2),
        indir(Y, S2, S)
    ;   indir(Y, S0, S1),
        dir(X, S1, S)
    ).
```
41 Definite clause grammars (expansion)

Author(s): Daniel Cabeza.
Version: 1.11#87 (2003/12/21, 2:9:27 CET)
This module implements the Definite clause grammars (expansion).

41.1 Usage and interface (dcg_expansion)

- Library usage:
  :- use_module(library(dcg_expansion)).
- Exports:
  - Predicates:
    phrase/2, phrase/3, dcg_translation/2.
- Other modules used:
  - System library modules:
    terms, assertions/doc_props.

41.2 Documentation on exports (dcg_expansion)

phrase/2:
  phrase(Phrase, List)
Like phrase(Phrase,List,[]).
Meta-predicate with arguments: phrase(goal,?).

phrase/3:
  Meta-predicate with arguments: phrase(goal,?,?,?).
Usage: phrase(Phrase, List, Remainder)
  - Description: The list List is a phrase of type Phrase (according to the current
grammar rules), where Phrase is either a non-terminal or more generally a grammar
rule body. Remainder is what remains of the list after a phrase has been found.
  - The following properties should hold at call time:
    Phrase is currently a term which is not a free variable. (term_typing:nonvar/1)
  - The following properties should hold globally:
    Documentation is still incomplete: phrase(Phrase,List,Remainder) may not con-
    form the functionality documented. (doc_props:doc_incomplete/1)

dcg_translation/2:
  Performs the code expansion of source clauses that use DCGs.
42 Formatted output

Author(s): The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#89 (2003/12/21, 2:17:58 CET)

The format family of predicates is due to Quintus Prolog. They act as a Prolog interface to the C stdio function printf(), allowing formatted output.

Output is formatted according to an output pattern which can have either a format control sequence or any other character, which will appear verbatim in the output. Control sequences act as place-holders for the actual terms that will be output. Thus

?- format("Hello ~q!", world).

will print Hello world!.

If there is only one item to print it may be supplied alone. If there are more they have to be given as a list. If there are none then an empty list should be supplied. There has to be as many items as control characters.

The character ~ introduces a control sequence. To print a ~ verbatim just repeat it:

?- format("Hello ~~world!", []).

will result in Hello ~world!.

A format may be spread over several lines. The control sequence \c followed by a \LF \r will translate to the empty string:

?- format("Hello \c world!", []).

will result in Hello world!.

42.1 Usage and interface (format)

- Library usage:
  :- use_module(library(format)).

- Exports:
  - Predicates:
    format/2, format/3.
  - Regular Types:
    format_control/1.

- Other modules used:
  - System library modules:
    write, assertions/doc_props.
42.2 Documentation on exports (format)

format/2:

General properties: format(C, A)

- The following properties hold globally:
  This predicate is understood natively by CiaoPP as format(C,A).

Usage: format(format_control(Format), Arguments)

- Description: Print Arguments onto current output stream according to format Format.

format/3:

General properties: format(S, C, A)

- The following properties hold globally:
  This predicate is understood natively by CiaoPP as format(S,C,A).

Usage: format(Stream, format_control(Format), Arguments)

- Description: Print Arguments onto Stream according to format Format.

- The following properties should hold at call time:
  Stream is currently a term which is not a free variable.

format_control/1:

The general format of a control sequence is ~NC. The character C determines the type of the control sequence. N is an optional numeric argument. An alternative form of N is *. *

implies that the next argument in Arguments should be used as a numeric argument in the control sequence. Example:

?- format("Hello~4cworl\d!", [0'x]).

and

?- format("Hello~*cworl\d!", [4,0'x]).

both produce

Helloxxxxxworl\d!

The following control sequences are available.

- ~a The argument is an atom. The atom is printed without quoting.
- ~Nc (Print character.) The argument is a number that will be interpreted as an ASCII code. N defaults to one and is interpreted as the number of times to print the character.
- ~Ne
- ~NE
- ~Nf
- ~Ng
- ~NG (Print float). The argument is a float. The float and N will be passed to the C printf() function as
printf("%.Ne", Arg)
printf("%.NE", Arg)
printf("%.Nf", Arg)
printf("%.Ng", Arg)
printf("%.NG", Arg)

If N is not supplied the action defaults to
printf("%e", Arg)
printf("%E", Arg)
printf("%f", Arg)
printf("%g", Arg)
printf("%G", Arg)

• ~Nd (Print decimal.) The argument is an integer. N is interpreted as the number of
digits after the decimal point. If N is 0 or missing, no decimal point will be printed.
Example:
  ?- format("Hello ~1d world!", [42]).
  ?- format("Hello ~d world!", [42]).

will print as
Hello 4.2 world!
Hello 42 world!
respectively.

• ~Nd (Print decimal.) The argument is an integer. Identical to ~Nd except that , will
separate groups of three digits to the left of the decimal point. Example:
  ?- format("Hello ~1D world!", [12345]).

will print as
Hello 1,234.5 world!

• ~Nr (Print radix.) The argument is an integer. N is interpreted as a radix. N should
be >= 2 and <= 36. If N is missing the radix defaults to 8. The letters a-z will denote
digits larger than 9. Example:
  ?- format("Hello ~2r world!", [15]).
  ?- format("Hello ~16r world!", [15]).

will print as
Hello 1111 world!
Hello f world!
respectively.

• ~NR (Print radix.) The argument is an integer. Identical to ~Nr except that the
letters A-Z will denote digits larger than 9. Example:
  ?- format("Hello ~16R world!", [15]).

will print as
Hello F world!

• ~Ns (Print string.) The argument is a list of ASCII codes. Exactly N characters will
be printed. N defaults to the length of the string. Example:
  ?- format("Hello ~4s ~4s!", ["new", "world"]).
  ?- format("Hello ~s world!", ["new"]).

will print as
Hello new world!
Hello new world!
respectively.
• ~i (Ignore argument.) The argument may be of any type. The argument will be ignored. Example:
  
  ?- format("Hello ~i's world!", ["old","new"]).
  
  will print as
  Hello new world!

• ~k (Print canonical.) The argument may be of any type. The argument will be passed to write_canonical/2 (Chapter 35 [Term output], page 191). Example:

  ?- format("Hello "k world!", [[a,b,c]]).
  
  will print as
  Hello .(a,.(b,.(c,[]))) world!

• ~p (print.) The argument may be of any type. The argument will be passed to print/2 (Chapter 35 [Term output], page 191). Example:

  suposing the user has defined the predicate
  
  :­ multifile portray/1.
  portraits([X|Y]) :- print(cons(X,Y)).
  
  then
  
  ?- format("Hello "p world!", [[a,b,c]]).
  
  will print as
  Hello cons(a,cons(b,cons(c,[]))) world!

• ~q (Print quoted.) The argument may be of any type. The argument will be passed to writeq/2 (Chapter 35 [Term output], page 191). Example:

  ?- format("Hello "q world!", [['A', 'B']]).
  
  will print as
  Hello ['A','B'] world!

• ~w (write.) The argument may be of any type. The argument will be passed to write/2 (Chapter 35 [Term output], page 191). Example:

  ?- format("Hello "w world!", [['A', 'B']]).
  
  will print as
  Hello [A,B] world!

• ~Nn (Print newline.) Print N newlines. N defaults to 1. Example:

  ?- format("Hello ~n world!", []).
  
  will print as
  Hello world!

• ~N (Fresh line.) Print a newline, if not already at the beginning of a line.

• ~ (Print tilde.) Prints ~

The following control sequences are also available for compatibility, but do not perform any useful functions.

• ~N (Set tab.) Set a tab stop at position N, where N defaults to the current position, and advance the current position there.

• ~N+ (Advance tab.) Set a tab stop at N positions past the current position, where N defaults to 8, and advance the current position there.

• ~Nt (Set fill character.) Set the fill character to be used in the next position movement to N, where N defaults to _SPC_.

Usage: format_control(C)
Description: \( C \) is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with \texttt{name/2}.

The following properties should hold globally:

Documentation is still incomplete: \texttt{format_control(C)} may not conform the functionality documented. \texttt{(doc_props:doc_incomplete/1)}
43 List processing

Author(s): The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#142 (2003/12/31, 12:29:49 CET)

This module provides a set of predicates for list processing.

43.1 Usage and interface (lists)

- Library usage:
  
  ```prolog
  :- use_module(library(lists)).
  ```

- Exports:
  - Predicates:
    - nonsingle/1, append/3, reverse/2, reverse/3, delete/3, delete_non_ground/3, select/3, length/2, nth/3, add_after/4, add_before/4, dlist/3, list_concat/2, list_insert/2, insert_last/3, contains_ro/2, contains1/2, nocontainsx/2, last/2, list_lookup/3, list_lookup/4, intset_insert/3, intset_delete/3, intset_in/2, intset_sequence/3, intersection/3, union/3, difference/3, equal_lists/2, list_to_list_of_lists/2, powerset/2, cross_product/2.
  - Properties:
    - list1/2, sublist/2, subordlist/2.

43.2 Documentation on exports (lists)

nonsingle/1: PREDICATE

Usage: nonsingle(X)

- Description: X is not a singleton.

append/3: PREDICATE

Usage: append(Xs, Ys, Zs)

- Description: Zs is Ys appended to Xs.

reverse/2: PREDICATE

Usage: reverse(Xs, Ys)

- Description: Reverses the order of elements in Xs.
  - The following properties should hold at call time:
    - Xs is a list.
    - Ys is any term.
  - The following properties should hold upon exit:
    - Xs is a list.
    - Ys is a list.
reverse/3:
Usage: reverse(A, B, C)
Description: Reverse the order of elements in A, and append it with B.

delete/3:
Usage: delete(L1, E, L2)
Description: L2 is L1 without the occurrences of E.

delete_non_ground/3:
Usage: delete_non_ground(L1, E, L2)
Description: L2 is L1 without the occurrences of E. E can be a nonground term so that all the elements in L1 it unifies with will be deleted.

select/3:
Usage: select(X, Xs, Ys)
Description: Xs and Ys have the same elements except for one occurrence of X.

length/2:
General properties: length(A, B)
The following properties hold globally:
This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 1: length(L, N)
Description: Computes the length of L.
The following properties should hold at call time:
L is a list. (basic_props:list/1)
N is a free variable. (term_typing:var/1)
The following properties hold upon exit:
L is a list. (basic_props:list/1)
N is an integer. (basic_props:int/1)

Usage 2: length(L, N)
Description: Outputs L of length N.
The following properties should hold at call time:
L is a free variable. (term_typing:var/1)
N is an integer. (basic_props:int/1)
The following properties hold upon exit:
L is a list. (basic_props:list/1)
N is an integer. (basic_props:int/1)

Usage 3: length(L, N)
Description: Checks that L is of length N.
– The following properties should hold at call time:

L is a list.  
N is an integer.  

– The following properties hold upon exit:

L is a list.  
N is an integer.

nth/3:
PREDICATE

nth(N, List, Elem)
N is the position in List of Elem. N counts from one.

Usage 1:
– Description: Unifies Elem and the Nth element of List.
– Call and exit should be compatible with:

List is a list.  
Elem is any term.  

– The following properties should hold at call time:

N is an integer.  

– The following properties should hold upon exit:

List is a list.  
Elem is any term.

Usage 2:
– Description: Finds the positions where Elem is in List. Positions are found in ascending order.
– Call and exit should be compatible with:

List is a list.  
Elem is any term.  

– The following properties should hold at call time:

N is a free variable.  

– The following properties should hold upon exit:

N is an integer.  
List is a list.  
Elem is any term.

add_after/4:
PREDICATE

Usage: add_after(L0, E0, E, L)
– Description: Adds element E after element E0 (or at end) to list L0 returning in L the new list (uses term comparison).
– The following properties should hold at call time:

L0 is currently a term which is not a free variable.  
E0 is currently a term which is not a free variable.  
E is currently a term which is not a free variable.  
L is a free variable.
add_before/4:
Usage: add_before(L0, E0, E, L)
  – Description: Adds element E before element E0 (or at start) to list L0 returning in
    L the new list (uses term comparison).
  – The following properties should hold at call time:
    L0 is currently a term which is not a free variable. (term_typing:nonvar/1)
    E0 is currently a term which is not a free variable. (term_typing:nonvar/1)
    E is currently a term which is not a free variable. (term_typing:nonvar/1)
    L is a free variable. (term_typing:var/1)

list1/2:
Meta-predicate with arguments: list1(?.,pred(1)).
Usage: list1(X, Y)
  – Description: X is a list of Ys of at least one element.

dlist/3:
Usage: dlist(List, DList, Tail)
  – Description: List is the result of removing Tail from the end of DList (makes a
    difference list from a list).

list_concat/2:
Usage: list_concat(list(LL,list), list(L))
  – Description: L is the concatenation of all the lists in LL.

list_insert/2:
Usage: list_insert(List, Term)
  – Description: Adds Term to the end of List if there is no element in List identical to
    Term.
  – The following properties should hold at call time:
    List is a free variable. (term_typing:var/1)
    Term is currently a term which is not a free variable. (term_typing:nonvar/1)

insert_last/3:
Usage: insert_last(L0, E, L)
  – Description: Adds element E at end of list L0 returning L.
  – The following properties should hold at call time:
    L0 is currently a term which is not a free variable. (term_typing:nonvar/1)
    E is currently a term which is not a free variable. (term_typing:nonvar/1)
    L is a free variable. (term_typing:var/1)
contains_ro/2:
   Usage:
   – Description: Impure membership (does not instantiate a variable in its first argument).

contains1/2:
   Usage:
   – Description: First membership.

nocontainsx/2:
   Usage: nocontainsx(L, X)
   – Description: X is not identical to any element of L.

last/2:
   Usage: last(L, X)
   – Description: X is the last element of list L.

list_lookup/3:
   Usage: list_lookup(List, Key, Value)
   – Description: Same as list_lookup/4, but use -/2 as functor.

list_lookup/4:
   Usage: list_lookup(List, Functor, Key, Value)
   – Description: Look up Functor(Key,Value) pair in variable ended key-value pair list L or else add it at the end.

intset_insert/3:
   Usage: intset_insert(A, B, Set)
   – Description: Insert the element B in the ordered set of numbers A.

intset_delete/3:
   Usage: intset_delete(A, B, Set)
   – Description: Delete from the ordered set A the element B.

intset_in/2:
   Usage: intset_in(E, Set)
   – Description: Succeeds iff E is element of Set
intset_sequence/3:  
Usage: intset_sequence(N, L1, L2)
- Description: Generates an ordered set of numbers from 0 to N-1, and append it to L1.

intersection/3:  
Usage: intersection(List1, List2, List)
- Description: List has the elements which are both in List1 and List2.
- The following properties should hold at call time:
  List1 is currently a term which is not a free variable.  (term_typing:nonvar/1)
  List2 is currently a term which is not a free variable.  (term_typing:nonvar/1)
  List is a free variable.  (term_typing:var/1)

union/3:  
Usage: union(List1, List2, List)
- Description: List has the elements which are in List1 followed by the elements which are in List2 but not in List1.
- The following properties should hold at call time:
  List1 is currently a term which is not a free variable.  (term_typing:nonvar/1)
  List2 is currently a term which is not a free variable.  (term_typing:nonvar/1)
  List is a free variable.  (term_typing:var/1)

difference/3:  
Usage: difference(List1, List2, List)
- Description: List has the elements which are in List1 but not in List2.
- The following properties should hold at call time:
  List1 is currently a term which is not a free variable.  (term_typing:nonvar/1)
  List2 is currently a term which is not a free variable.  (term_typing:nonvar/1)
  List is a free variable.  (term_typing:var/1)

sublist/2:  
Usage: sublist(List1, List2)
- Description: List2 contains all the elements of List1.
- If the following properties should hold at call time:
  List2 is currently a term which is not a free variable.  (term_typing:nonvar/1)

subordlist/2:  
Usage: subordlist(List1, List2)
- Description: List2 contains all the elements of List1 in the same order.
- If the following properties should hold at call time:
  List2 is currently a term which is not a free variable.  (term_typing:nonvar/1)
**equal_lists/2:**

**Usage:** equal_lists(List1, List2)

- **Description:** List1 has all the elements of List2, and vice versa.
- **The following properties should hold at call time:**
  - List1 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - List2 is currently a term which is not a free variable. (term_typing:nonvar/1)

**list_to_list_of_lists/2:**

**Usage:** list_to_list_of_lists(List, LList)

- **Description:** LList is the list of one element lists with elements of List.
- **The following properties should hold at call time:**
  - List is currently a term which is not a free variable. (term_typing:nonvar/1)
  - LList is a free variable. (term_typing:var/1)

**powerset/2:**

**Usage:** powerset(List, LList)

- **Description:** LList is the powerset of List, i.e., the list of all lists which have elements of List. If List is ordered, LList and all its elements are ordered.
- **The following properties should hold at call time:**
  - List is currently a term which is not a free variable. (term_typing:nonvar/1)
  - LList is a free variable. (term_typing:var/1)

**cross_product/2:**

**Usage:** cross_product(LList, List)

- **Description:** List is the cartesian product of the lists in LList, that is, the list of lists formed with one element of each list in LList, in the same order.
- **The following properties should hold at call time:**
  - LList is currently a term which is not a free variable. (term_typing:nonvar/1)
  - List is a free variable. (term_typing:var/1)
44 Sorting lists

Author(s): Richard A. O'Keefe. All changes by UPM CLIP Group.
This module implements some sorting list predicates.

44.1 Usage and interface (sort)

- Library usage:
  :- use_module(library(sort)).
- Exports:
  - Predicates:
    sort/2, keysort/2.
  - Regular Types:
    keylist/1.

44.2 Documentation on exports (sort)

sort/2: PREDICATE
sort(List1, List2)
The elements of List1 are sorted into the standard order (see Chapter 18 [Comparing terms], page 95) and any identical elements are merged, yielding List2. The time and space complexity of this operation is at worst $O(N \lg N)$ where $N$ is the length of List1.

Usage:
- **Description**: List2 is the sorted list corresponding to List1.
- **Calls should, and exit will be compatible with**: List2 is a list. (basic_props:list/1)
- **The following properties should hold at call time**: List1 is a list. (basic_props:list/1)
- **The following properties hold upon exit**: List2 is a list. (basic_props:list/1)
- **The following properties hold globally**: This predicate is understood natively by CiaoPP. (basic_props:native/1)

keysort/2: PREDICATE
keysort(List1, List2)
List1 is sorted into order according to the value of the keys of its elements, yielding the list List2. No merging takes place. This predicate is stable, i.e., if an element A occurs before another element B with the same key in the input, then A will occur before B also in the output. The time and space complexity of this operation is at worst $O(N \lg N)$ where $N$ is the length of List1.

Usage:
- *Description:* List2 is the (key-)sorted list corresponding to List1.
- *Calls should, and exit will be compatible with:*
  List2 is a list of pairs of the form Key-Value.  
  (sort: keylist/1)
- *The following properties should hold at call time:*
  List1 is a list of pairs of the form Key-Value.  
  (sort: keylist/1)
- *The following properties hold upon exit:*
  List2 is a list of pairs of the form Key-Value.  
  (sort: keylist/1)
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP.  
  (basic_props: native/1)

**keylist/1:**

  *Usage:* keylist(L)
  - *Description:* L is a list of pairs of the form Key-Value.

**44.3 Documentation on internals (sort)**

**keypair/1:**

  *Usage:* keypair(P)
  - *Description:* P is a pair of the form "K-", where K is considered the key.
45 compiler (library)

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#176 (2002/1/14, 17:27:0 CET)

45.1 Usage and interface (compiler)

- Library usage:
  ```prolog```
  :- use_module(library(compiler)).
  ```prolog```

- Exports:
  - *Predicates:*
    ```prolog```
    make_po/1, ensure_loaded/1, ensure_loaded/2, use_module/1, 
    use_module/2, use_module/3, unload/1, set_debug_mode/1, set_nodebug_mode/1, 
    set_debug_module/1, set_nodebug_module/1, set_debug_module_source/1, mode_of_module/2, module_of/2.
  ```prolog```

- Other modules used:
  - *System library modules:*
    ```prolog```
    compiler/c_itf_internal.
  ```prolog```

45.2 Documentation on exports (compiler)

**make_po/1:**

No further documentation available for this predicate.

**ensure_loaded/1:**

No further documentation available for this predicate.

**ensure_loaded/2:**

No further documentation available for this predicate.

**use_module/1:**

No further documentation available for this predicate.

**use_module/2:**

No further documentation available for this predicate.

*Meta-predicate* with arguments: use_module(?,,addmodule).

**use_module/3:**

No further documentation available for this predicate.
unload/1:
   No further documentation available for this predicate.

set_debug_mode/1:
   No further documentation available for this predicate.

set_nodebug_mode/1:
   No further documentation available for this predicate.

set_debug_module/1:
   No further documentation available for this predicate.

set_nodebug_module/1:
   No further documentation available for this predicate.

set_debug_module_source/1:
   No further documentation available for this predicate.

mode_of_module/2:
   No further documentation available for this predicate.

module_of/2:
   No further documentation available for this predicate.
46 Enumeration of integers inside a range

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#159 (2004/1/9, 16:54:42 CET)
This module enumerates integers between two numbers, or checks that an integer lies within a range.

46.1 Usage and interface (between)

- Library usage:
  :- use_module(library(between)).
- Exports:
  - *Predicates:*
    between/3.

46.2 Documentation on exports (between)

between/3:

Usage: between(Min, Max, N)

- *Description:* N is an integer which is greater than or equal to Min and smaller than or equal to Max. Both Min and Max can be either integer or real numbers.
- *The following properties should hold at call time:*

  - Min is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Max is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Min is currently instantiated to a number. (term_typing:number/1)
  - Max is currently instantiated to a number. (term_typing:number/1)
  - N is an integer. (basic_props:int/1)
47 Operating system utilities

Author(s): Daniel Cabeza, Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#184 (2004/2/11, 18:35:28 CET)

This module contains predicates for invoking services which are typically provided by the operating system. Note that the predicates which take names of files or directories as arguments in this module expect atoms, not path aliases. I.e., generally these predicates will not call absolute_file_name/2 on names of files or directories taken as arguments.

47.1 Usage and interface (system)

- Library usage:
  
  :- use_module(library(system)).

- Exports:
  
  - Predicates:
    
    pause/1, time/1, datime/1, datime/9, getenvstr/2, setenvstr/2, current_env/2, set_env/2, del_env/1, copy_file/2, extract_paths/2, get_pid/1, current_host/1, current_executable/1, umask/2, make_directory/2, make_directory/1, make_dirpath/2, make_dirpath/1, working_directory/2, cd/1, shell/0, shell/1, shell/2, system/1, system/2, popen/3, exec/4, exec/3, directory_files/2, mktemp/2, file_exists/1, file_exists/2, file_property/2, file_properties/6, modif_time/2, modif_time0/2, fmode/2, chmod/2, chmod/3, delete_file/1, delete_directory/1, rename_file/2, cyg2win/3.

  - Regular Types:
    
    datime_struct/1, popen_mode/1.

  - Multifiles:
    
    define_flag/3.

- Other modules used:
  
  - System library modules:
    
    lists.

47.2 Documentation on exports (system)

pause/1:

  pause(Seconds)

  Make this thread sleep for some Seconds.

  Usage:

  - The following properties should hold at call time:

    Seconds is an integer.

    (basic_props:int/1)

  (basic_props:int/1)

  time/1:

  time(Time)

  (basic_props:int/1)
Time is unified with the number of seconds elapsed since January, 1, 1970 (UTC).

Usage:
- Calls should, and exit will be compatible with:
  Time is an integer. (basic_props:int/1)
- The following properties hold upon exit:
  Time is an integer. (basic_props:int/1)

datetime/1:  

\texttt{datetime(Datetime)}

Datime is unified with a term of the form \texttt{datetime(Year,Month,Day,Hour,Minute,Second)} which contains the current date and time.

Usage:
- Calls should, and exit will be compatible with:
  \texttt{system:datetime_struct(Datetime)} (system:datetime_struct/1)
- The following properties hold upon exit:
  \texttt{system:datetime_struct(Datetime)} (system:datetime_struct/1)

datetime/9:  

\texttt{datetime(Time, Year, Month, Day, Hour, Min, Sec, WeekDay, YearDay)}

Time is as in time/1. WeekDay is the number of days since Sunday, in the range 0 to 6. YearDay is the number of days since January 1, in the range 0 to 365.

Usage 1:
- Description: If Time is given, the rest of the arguments are unified with the date and time to which the Time argument refers.
- Calls should, and exit will be compatible with:
  Year is an integer. (basic_props:int/1)
  Month is an integer. (basic_props:int/1)
  Day is an integer. (basic_props:int/1)
  Hour is an integer. (basic_props:int/1)
  Min is an integer. (basic_props:int/1)
  Sec is an integer. (basic_props:int/1)
  WeekDay is an integer. (basic_props:int/1)
  YearDay is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Time is an integer. (basic_props:int/1)
- The following properties hold upon exit:
  Year is an integer. (basic_props:int/1)
  Month is an integer. (basic_props:int/1)
  Day is an integer. (basic_props:int/1)
  Hour is an integer. (basic_props:int/1)
  Min is an integer. (basic_props:int/1)
  Sec is an integer. (basic_props:int/1)
  WeekDay is an integer. (basic_props:int/1)
  YearDay is an integer. (basic_props:int/1)
Usage 2:

- **Description:** Bound **Time** to current time and the rest of the arguments refer to current time.
- **Calls should, and exit will be compatible with:**
  - **Year** is an integer. (basic_props:int/1)
  - **Month** is an integer. (basic_props:int/1)
  - **Day** is an integer. (basic_props:int/1)
  - **Hour** is an integer. (basic_props:int/1)
  - **Min** is an integer. (basic_props:int/1)
  - **Sec** is an integer. (basic_props:int/1)
  - **WeekDay** is an integer. (basic_props:int/1)
  - **YearDay** is an integer. (basic_props:int/1)

- **The following properties should hold at call time:**
  - **Time** is a free variable. (term_typing:var/1)

- **The following properties hold upon exit:**
  - **Time** is an integer. (basic_props:int/1)
  - **Year** is an integer. (basic_props:int/1)
  - **Month** is an integer. (basic_props:int/1)
  - **Day** is an integer. (basic_props:int/1)
  - **Hour** is an integer. (basic_props:int/1)
  - **Min** is an integer. (basic_props:int/1)
  - **Sec** is an integer. (basic_props:int/1)
  - **WeekDay** is an integer. (basic_props:int/1)
  - **YearDay** is an integer. (basic_props:int/1)

**datime_struct/1:**

A regular type, defined as follows:

```
   datime_struct(datime(Year,Month,Day,Hour,Min,Sec)) :-
     int(Year),
     int(Month),
     int(Day),
     int(Hour),
     int(Min),
     int(Sec).
```

**getenvstr/2:**

**getenvstr**(Name, Value)

The environment variable **Name** has **Value**. Fails if variable **Name** is not defined.

**Usage:**

- **Calls should, and exit will be compatible with:**
  - **Value** is a string (a list of character codes). (basic_props:string/1)

- **The following properties should hold at call time:**
  - **Name** is an atom. (basic_props:atm/1)
The following properties hold upon exit:
Value is a string (a list of character codes).

**Predicate**

**setenvstr/2:**

setenvstr(Name, Value)
The environment variable Name is assigned Value.

**Usage:**

- The following properties should hold at call time:
  Name is an atom.  
  Value is a string (a list of character codes).

**Predicate**

**current_env/2:**

current_env(Name, Value)
If Name is an atom, then unifies the environment variable Name with its value. Note that this predicate can be used to enumerate all the environment variables using backtracking.

**Usage:**

- Calls should, and exit will be compatible with:
  Name is an atom.  
  Value is an atom.  
- The following properties hold upon exit:
  Name is an atom.  
  Value is an atom.

**Predicate**

**set_env/2:**

set_env(Name, Value)
The environment variable Name is assigned Value.

**Usage:**

- The following properties should hold at call time:
  Name is an atom.  
  Value is an atom.

**Predicate**

**del_env/1:**

del_env(Name)
The environment variable Name is removed.

**Usage:**

- The following properties should hold at call time:
  Name is an atom.
copy_file/2:  
\text{copy_file(Source, Destination)}

Copies the file \text{Source}

\textbf{Usage:}
\begin{itemize}
  \item \textit{The following properties should hold at call time:}
    \begin{itemize}
      \item Source is an atom. \hspace{1cm} (basic_props:atm/1)
      \item Destination is an atom. \hspace{1cm} (basic_props:atm/1)
    \end{itemize}
\end{itemize}

extract_paths/2:  
\text{extract_paths(String, Paths)}

Interpret \text{String} as the value of a UNIX environment variable holding a list of paths and return in \text{Paths} the list of the paths. Paths in \text{String} are separated by colons, and an empty path is considered a shorthand for \texttt{.} (current path). The most typical environment variable with this format is \texttt{PATH}. For example, this is a typical use:

\begin{verbatim}
?- set_prolog_flag(write_strings, on).
yes
?- getenvstr('PATH', PATH), extract_paths(PATH, Paths).

PATH = ":/home/bardo/bin:/home/clip/bin:/opt/bin:/bin",
Paths = [".","/home/bardo/bin","/home/clip/bin","/opt/bin","/bin"] ?

yes
?-
\end{verbatim}

\textbf{Usage:}
\begin{itemize}
  \item \textit{Calls should, and exit will be compatible with:}
    \begin{itemize}
      \item Paths is a list of strings. \hspace{1cm} (basic_props:list/2)
      \item String is a string (a list of character codes). \hspace{1cm} (basic_props:string/1)
      \item Paths is a list of strings. \hspace{1cm} (basic_props:list/2)
    \end{itemize}
\end{itemize}

get_pid/1:  
\text{get_pid(Pid)}

Unifies \text{Pid} with the process identifier of the current process or thread.

\textbf{Usage:}
\begin{itemize}
  \item \textit{Calls should, and exit will be compatible with:}
    \begin{itemize}
      \item Pid is an integer. \hspace{1cm} (basic_props:int/1)
    \end{itemize}
  \item \textit{The following properties hold upon exit:}
    \begin{itemize}
      \item Pid is an integer. \hspace{1cm} (basic_props:int/1)
    \end{itemize}
\end{itemize}
current_host/1:
  current_host(Hostname)
Hostname is unified with the fully qualified name of the host.
Usage:
  - Calls should, and exit will be compatible with:
    Hostname is an atom.  \(\text{basic_props:atm/1}\)
  - The following properties hold upon exit:
    Hostname is an atom.  \(\text{basic_props:atm/1}\)

current_executable/1:
  current_executable(Path)
Unifies Path with the path to the current executable.
Usage:
  - Calls should, and exit will be compatible with:
    Path is an atom.  \(\text{basic_props:atm/1}\)
  - The following properties hold upon exit:
    Path is an atom.  \(\text{basic_props:atm/1}\)

umask/2:
  umask(OldMask, NewMask)
The process file creation mask was OldMask, and it is changed to NewMask.
Usage 1:
  - Calls should, and exit will be compatible with:
    OldMask is an integer.  \(\text{basic_props:int/1}\)
  - The following properties should hold at call time:
    NewMask is an integer.  \(\text{basic_props:int/1}\)
  - The following properties hold upon exit:
    OldMask is an integer.  \(\text{basic_props:int/1}\)
Usage 2: umask(OldMask, NewMask)
  - Description: Gets the process file creation mask without changing it.
  - The following properties should hold at call time:
    OldMask is a free variable.  \(\text{term_typing:var/1}\)
    NewMask is a free variable.  \(\text{term_typing:var/1}\)
    The terms OldMask and NewMask are strictly identical.  \(\text{term_compare::=/2}\)
  - The following properties hold upon exit:
    OldMask is an integer.  \(\text{basic_props:int/1}\)
    NewMask is an integer.  \(\text{basic_props:int/1}\)

make_directory/2:
  make_directory(DirName, Mode)
Creates the directory DirName with a given Mode. This is, as usual, operated against the
current umask value.
Usage:
– The following properties should hold at call time:
  DirName is an atom.
  Mode is an integer.

### make_directory/1:

**Predicate**

\[
\text{make_directory(DirName)}
\]

Equivalent to `make_directory(D,0o777)`.

**Usage:**

– The following properties should hold at call time:
  DirName is an atom.

### make_dirpath/2:

**Predicate**

\[
\text{make_dirpath(Path, Mode)}
\]

Creates the whole Path for a given directory with a given Mode. As an example, `make_dirpath('/tmp/var/mydir/otherdir')`.

**Usage:**

– The following properties should hold at call time:
  Path is a source name.
  Mode is an integer.

### make_dirpath/1:

**Predicate**

\[
\text{make_dirpath(Path)}
\]

Equivalent to `make_dirpath(D,0o777)`.

**Usage 1:**

– The following properties should hold at call time:
  Path is an atom.

**Usage 2:**

– The following properties should hold at call time:
  Path is an atom.

### working_directory/2:

**Predicate**

\[
\text{working_directory(OldDir, NewDir)}
\]

Unifies current working directory with OldDir, and then changes the working directory to NewDir. Calling `working_directory(Dir,Dir)` simply unifies Dir with the current working directory without changing anything else.

**Usage 1:**

– Calls should, and exit will be compatible with:
  OldDir is an atom.
  NewDir is an atom.
- The following properties hold upon exit:
  OldDir is an atom. (basic_props:atm/1)

Usage 2: working_directory(OldDir, NewDir)
- Description: Gets current working directory.
- The following properties should hold at call time:
  OldDir is a free variable. (term_typing:var/1)
  NewDir is a free variable. (term_typing:var/1)
  The terms OldDir and NewDir are strictly identical. (term_compare:== /2)
- The following properties hold upon exit:
  OldDir is an atom. (basic_props:atm/1)
  NewDir is an atom. (basic_props:atm/1)

cd/1:  PREDICATE
  cd(Path)
  Changes working directory to Path.
  Usage:
  - The following properties should hold at call time:
    Path is an atom. (basic_props:atm/1)

shell/0:  PREDICATE
  Usage:
  - Description: Execs the shell specified by the environment variable SHELL. When the shell process terminates, control is returned to Prolog.

shell/1:  PREDICATE
  shell(Command)
  Command is executed in the shell specified by the environment variable SHELL. It succeeds if the exit code is zero and fails otherwise.
  Usage:
  - The following properties should hold at call time:
    Command is an atom. (basic_props:atm/1)

shell/2:  PREDICATE
  shell(Command, ReturnCode)
  Executes Command in the shell specified by the environment variable SHELL and stores the exit code in ReturnCode.
  Usage:
  - Calls should, and exit will be compatible with:
    ReturnCode is an integer. (basic_props:int/1)
  - The following properties should hold at call time:
    Command is an atom. (basic_props:atm/1)
  - The following properties hold upon exit:
    ReturnCode is an integer. (basic_props:int/1)
system/1:  
\[
\text{system(Command)} \\
\text{Executes Command using the shell } /bin/sh. \\
\text{Usage:} \\
\quad \text{The following properties should hold at call time:} \\
\quad \text{Command is an atom.} \\
\]

system/2:  
\[
\text{system(Command, ReturnCode)} \\
\text{Executes Command in the } /bin/sh \text{ shell and stores the exit code in ReturnCode.} \\
\text{Usage:} \\
\quad \text{Calls should, and exit will be compatible with:} \\
\quad \text{ReturnCode is an integer.} \\
\quad \text{The following properties should hold at call time:} \\
\quad \text{Command is an atom.} \\
\quad \text{The following properties hold upon exit:} \\
\quad \text{ReturnCode is an integer.} \\
\]

popen/3:  
\[
\text{popen(Command, Mode, Stream)} \\
\text{Open a pipe to process Command in a new shell with a given Mode and return a communication Stream (as in UNIX popen(3)). If Mode is read the output from the process is sent to Stream. If Mode is write, Stream is sent as input to the process. Stream may be read from or written into using the ordinary stream I/O predicates. Stream must be closed explicitly using close/1, i.e., it is not closed automatically when the process dies.} \\
\text{Usage:} \\
\quad \text{The following properties should hold at call time:} \\
\quad \text{Command is an atom.} \\
\quad \text{Mode is 'read' or 'write'.} \\
\quad \text{Stream is a free variable.} \\
\quad \text{The following properties hold upon exit:} \\
\quad \text{Stream is an open stream.} \\
\]

popen_mode/1:  
\[
\text{Usage: popen_mode(M)} \\
\quad \text{Description: } M \text{ is 'read' or 'write'.} \\
\]

exec/4:  
\[
\text{exec(Command, StdIn, StdOut, StdErr)} \\
\text{Starts the process Command and returns the standard I/O streams of the process in StdIn, StdOut, and StdErr.} \\
\text{Usage:} \\
\]
The following properties should hold at call time:

- **Command** is an atom. (basic.props:atm/1)
- **StdIn** is a free variable. (term.typing:var/1)
- **StdOut** is a free variable. (term.typing:var/1)
- **StdErr** is a free variable. (term.typing:var/1)

The following properties hold upon exit:

- **StdIn** is an open stream. (streams.basic:stream/1)
- **StdOut** is an open stream. (streams.basic:stream/1)
- **StdErr** is an open stream. (streams.basic:stream/1)

**exec/3:**

```
exec(Command, StdIn, StdOut)
```

Starts the process **Command** and returns the standard I/O streams of the process in **StdIn** and **StdOut**. **Standard error** is connected to whichever the parent process had it connected to.

**Usage:**

- The following properties should hold at call time:
  - **Command** is an atom. (basic.props:atm/1)
  - **StdIn** is a free variable. (term.typing:var/1)
  - **StdOut** is a free variable. (term.typing:var/1)

- The following properties hold upon exit:
  - **StdIn** is an open stream. (streams.basic:stream/1)
  - **StdOut** is an open stream. (streams.basic:stream/1)

**directory_files/2:**

```
directory_files(Directory, FileList)
```

FileList is the unordered list of entries (files, directories, etc.) in **Directory**.

**Usage:**

- Calls should, and exit will be compatible with:
  - **FileList** is a list of atms. (basic.props:list/2)

- The following properties should hold at call time:
  - **Directory** is an atom. (basic.props:atm/1)

- The following properties hold upon exit:
  - **FileList** is a list of atms. (basic.props:list/2)

**mktemp/2:**

```
mktemp(Template, Filename)
```

Returns a unique **Filename** based on **Template**: **Template** must be a valid file name with six trailing X, which are substituted to create a new file name.

**Usage:**

- Calls should, and exit will be compatible with:
  - **Filename** is an atom. (basic.props:atm/1)
The following properties should hold at call time:
Template is an atom. (basic_props:atm/1)

The following properties hold upon exit:
Filename is an atom. (basic_props:atm/1)

**PREDICATE**

`file_exists/1`:

`file_exists(File)`
Succeeds if File (a file or directory) exists (and is accessible).

**Usage:**
The following properties should hold at call time:
File is an atom. (basic_props:atm/1)

**PREDICATE**

`file_exists/2`:

`file_exists(File, Mode)`
File (a file or directory) exists and it is accessible with `Mode`, as in the Unix call `access(2)`.
Typically, `Mode` is 4 for read permission, 2 for write permission and 1 for execute permission.

**Usage:**
The following properties should hold at call time:
File is an atom. (basic_props:atm/1)
Mode is an integer. (basic_props:int/1)

**PREDICATE**

`file_property/2`:

`file_property(File, Property)`
File has the property `Property`. The possible properties are:

`type(Type)`
Type is one of regular, directory, symlink, fifo, socket or unknown.

`linkto(Linkto)`
If `File` is a symbolic link, `Linkto` is the file pointed to by the link (and the other properties come from that file, not from the link itself).

`mod_time(ModTime)`
ModTime is the time of last modification (seconds since January, 1, 1970).

`mode(Protection)`
Protection is the protection mode.

`size(Size)` Size is the size.

If `Property` is uninstantiated, the predicate will enumerate the properties on backtracking.

**Usage:**
Calls should, and exit will be compatible with:
Property is a compound term. (basic_props:struct/1)

The following properties should hold at call time:
File is an atom. (basic_props:atm/1)

The following properties hold upon exit:
Property is a compound term. (basic_props:struct/1)
file_properties/6:

file_properties(Path, Type, Linkto, Time, Protection, Size)

The file Path has the following properties:

- File type Type (one of regular, directory, symlink, fifo, socket or unknown).
- If Path is a symbolic link, Linkto is the file pointed to. All other properties come from the file pointed, not the link. Linkto is "" if Path is not a symbolic link.
- Time of last modification Time (seconds since January, 1, 1970).
- Protection mode Protection.
- Size in bytes Size.

Usage:

- Calls should, and exit will be compatible with:
  - Type is an atom. (basic_props:atm/1)
  - Linkto is an atom. (basic_props:atm/1)
  - Time is an integer. (basic_props:int/1)
  - Protection is an integer. (basic_props:int/1)
  - Size is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  - Path is an atom. (basic_props:atm/1)
- The following properties hold upon exit:
  - Type is an atom. (basic_props:atm/1)
  - Linkto is an atom. (basic_props:atm/1)
  - Time is an integer. (basic_props:int/1)
  - Protection is an integer. (basic_props:int/1)
  - Size is an integer. (basic_props:int/1)

modif_time/2:

modif_time(File, Time)

The file File was last modified at Time, which is in seconds since January, 1, 1970. Fails if File does not exist.

Usage:

- Calls should, and exit will be compatible with:
  - Time is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  - File is an atom. (basic_props:atm/1)
- The following properties hold upon exit:
  - Time is an integer. (basic_props:int/1)

modif_time0/2:

modif_time0(File, Time)

If File exists, Time is its latest modification time, as in modif_time/2. Otherwise, if File does not exist, Time is zero.

Usage:
Chapter 47: Operating system utilities

- Calls should, and exit will be compatible with:
  Time is an integer. (basic_props:int/1)

- The following properties should hold at call time:
  File is an atom. (basic_props:atm/1)

- The following properties hold upon exit:
  Time is an integer. (basic_props:int/1)

**fmode/2:**

fmode(File, Mode)
The file File has protection mode Mode.

**Usage:**

- Calls should, and exit will be compatible with:
  Mode is an integer. (basic_props:int/1)

- The following properties should hold at call time:
  File is an atom. (basic_props:atm/1)

- The following properties hold upon exit:
  Mode is an integer. (basic_props:int/1)

**chmod/2:**

chmod(File, NewMode)
Change the protection mode of file File to NewMode.

**Usage:**

- The following properties should hold at call time:
  File is an atom. (basic_props:atm/1)

  NewMode is an integer. (basic_props:int/1)

**chmod/3:**

chmod(File, OldMode, NewMode)
The file File has protection mode OldMode and it is changed to NewMode.

**Usage 1:**

- Calls should, and exit will be compatible with:
  OldMode is an integer. (basic_props:int/1)

- The following properties should hold at call time:
  File is an atom. (basic_props:atm/1)

  NewMode is an integer. (basic_props:int/1)

- The following properties hold upon exit:
  OldMode is an integer. (basic_props:int/1)

**Usage 2:** chmod(File, OldMode, NewMode)

- Description: Equivalent to fmode(File,OldMode)
The following properties should hold at call time:

- File is an atom. 
- OldMode is a free variable. 
- NewMode is a free variable. 
- The terms OldMode and NewMode are strictly identical.

The following properties hold upon exit:

- File is an atom. 
- OldMode is an atom. 
- NewMode is an atom.

---

delete_file/1: PREDICATE

define delete_file(File)
Delete the file File.
Usage:
- The following properties should hold at call time:
  - File is an atom.

---

delete_directory/1: PREDICATE

define delete_directory(File)
Delete the directory Directory.
Usage:
- The following properties should hold at call time:
  - File is an atom.

---

rename_file/2: PREDICATE

define rename_file(File1, File2)
Change the name of File1 to File2.
Usage:
- The following properties should hold at call time:
  - File1 is an atom.
  - File2 is an atom.

---

cyg2win/3: PREDICATE

Usage: cyg2win(CygWinPath, WindowsPath, SwapSlash)
- Description: Converts a path in the CygWin style to a Windows-style path, rewriting the driver part. If SwapSlash is swap, slashes are converted to backslash. If it is noswap, they are preserved.
- The following properties should hold at call time:
  - CygWinPath is a string (a list of character codes).
  - WindowsPath is a free variable.
  - SwapSlash is currently instantiated to an atom.
– The following properties should hold upon exit:
  CygWinPath is a string (a list of character codes).  (basic_props:string/1)
  WindowsPath is a string (a list of character codes).  (basic_props:string/1)
  SpawSlash is currently instantiated to an atom.  (term_typing:atom/1)

47.3 Documentation on multifiles (system)

define_flag/3:
  No further documentation available for this predicate.
  The predicate is multfile.

47.4 Known bugs and planned improvements (system)

• shell/n commands have a bug in Windows: if the environment variable SHELL is instantiated to some Windows shell implementation, then it is very possible that shell/
• If exec/4 does not find the command to be executed, there is no visible error message: it is sent to a error output which has already been assigned to a different stream, disconnected from the one the user sees.
• If the arguments to cyg2win/3 are not strings, strange results appear, as a very mild type checking is performed.
Chapter 48: Prolog system internal predicates

48 Prolog system internal predicates

Author(s): Manuel Carro, Daniel Cabeza, Mats Carlsson.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#57 (2003/11/24, 18:3:15 CET)

This module implements some miscellaneous predicates which provide access to some internal statistics, special properties of the predicates, etc.

48.1 Usage and interface (prolog_sys)

- Library usage:
  :- use_module(library(prolog_sys)).

- Exports:
  - Predicates:
    statistics/0, statistics/2, predicate_property/2, current_atom/1, garbage_collect/0, new_atom/1.

48.2 Documentation on exports (prolog_sys)

**statistics/0:**

**Usage:**

- **Description:** Prints statistics about the system.

**statistics/2:**

**Usage 1:** statistics(Time_option, Time_result)

- **Description:** Gather information about time (either process time or wall time) since last consult or since start of program. Results are returned in milliseconds.

- **The following properties should hold at call time:**
  Options to get information about execution time. Time_option must be one of runtime, usertime, systemtime or walltime. (prolog_sys:time_option/1)
  Time_result is any term. (basic_props:term/1)

- **The following properties hold upon exit:**
  Options to get information about execution time. Time_option must be one of runtime, usertime, systemtime or walltime. (prolog_sys:time_option/1)
  Time_result is a two-element list of numbers. The first number is the time since the start of the execution; the second number is the time since the previous consult to time. (prolog_sys:time_result/1)

**Usage 2:** statistics(Click_option, Click_result)

- **Description:** Gather information about clock clicks (either run, user, system or wall click) since last consult or since start of program.

- **The following properties should hold at call time:**
  Options to get information about execution clicks. (prolog_sys:click_option/1)
  Click_result is any term. (basic_props:term/1)
The following properties hold upon exit:
Options to get information about execution clicks. (prolog_sys:click_option/1)
Click_result is a two-element list of numbers. The first number is the number of
clicks since the start of the execution; the second number is the number of clicks since
the previous consult to click. (prolog_sys:click_result/1)

Usage 3: statistics(Clockfreq_option, Clockfreq_result)
- Description: Gather information about frequency of the clocks used to measure the
clicks (either run-user, system or wall clock). Results are returned in hertzios.
- The following properties should hold at call time:
  Options to get information about the frequency of clocks used to get the clicks.
  (prolog_sys:clockfreq_option/1)
  Clockfreq_result is any term. (basic_props:term/1)
- The following properties hold upon exit:
  Options to get information about the frequency of clocks used to get the clicks.
  (prolog_sys:clockfreq_option/1)
  Clockfreq_result is a number. It gives the frequency in hertzios used by the clock
  used to get the clicks. (prolog_sys:clockfreq_result/1)

Usage 4: statistics(Memory_option, Memory_result)
- Description: Gather information about memory consumption.
- The following properties should hold at call time:
  Options to get information about memory usage. (prolog_sys:memory_option/1)
  Memory_result is any term. (basic_props:term/1)
- The following properties hold upon exit:
  Options to get information about memory usage. (prolog_sys:memory_option/1)
  Result is a two-element list of integers. The first element is the space taken up by
  the option selected, measured in bytes; the second integer is zero for program space
  (which grows as necessary), and the amount of free space otherwise. (prolog_ sys:memory_result/1)

Usage 5: statistics(Garbage_collection_option, Gc_result)
- Description: Gather information about garbage collection.
- The following properties should hold at call time:
  Options to get information about garbage collection. (prolog_sys:garbage_collection_option/1)
  Gc_result is any term. (basic_props:term/1)
- The following properties hold upon exit:
  Options to get information about garbage collection. (prolog_sys:garbage_collection_option/1)
  Gc_result is a tree-element list of integers, related to garbage collection and memory
  management. When stack_shifts is selected, the first one is the number of shifts
  (reallocations) of the local stack; the second is the number of shifts of the trail, and
  the third is the time spent in these shifts. When garbage_collection is selected, the
  numbers are, respectively, the number of garbage collections performed, the number of
  bytes freed, and the time spent in garbage collection. (prolog_sys:gc_result/1)

Usage 6: statistics(Symbol_option, Symbol_result)
- Description: Gather information about number of symbols and predicates.
– *The following properties should hold at call time:*

  Option to get information about the number of symbols in the program.  
  (`prolog_sys:symbol_option/1`)

  Symbol_result is any term.  
  (`basic_props:term/1`)

– *The following properties hold upon exit:*

  Option to get information about the number of symbols in the program.  
  (`prolog_sys:symbol_option/1`)

  Symbol_result is a two-element list of integers. The first one is the number of atom, 
  functor, and predicate names in the symbol table. The second is the number of 
  predicates known to be defined (although maybe without clauses).  
  (`prolog_sys:symbol_result/1`)

Usage 7: `statistics(Option, Arg2)`

– *Description:* If `Option` is unbound, it is bound to the values on the other cases.

– *Calls should, and exit will be compatible with:*

  `Arg2` is any term.  
  (`basic_props:term/1`)

– *The following properties hold upon exit:*

  `Arg2` is any term.  
  (`basic_props:term/1`)

**predicate_property/2:**  

Usage: `predicate_property(Head, Property)`

– *Description:* The predicate with clause `Head` is `Property`.

– *The following properties should hold at call time:*

  `Head` is any term.  
  (`basic_props:term/1`)

  `Property` is any term.  
  (`basic_props:term/1`)

– *The following properties hold upon exit:*

  `Head` is a term which represents a goal, i.e., an atom or a structure.  
  (`basic_props:callable/1`)

  `Property` is an atom.  
  (`basic_props:atm/1`)

**current_atom/1:**  

Usage: `current_atom(Atom)`

– *Description:* Enumerates on backtracking all the existing atoms in the system.

– *The following properties should hold at call time:*

  `Atom` is a free variable.  
  (`term_typing:var/1`)

– *The following properties hold upon exit:*

  `Atom` is an atom.  
  (`basic_props:atm/1`)

**garbage_collect/0:**  

Usage:

– *Description:* Forces garbage collection when called.
new_atom/1:  
Usage: new_atom(Atom)

- Description: Returns, on success, a new atom, not existing before in the system.
  The entry argument must be a variable. The idea behind this atom generation is to
  provide a fast source of identifiers for new objects, concurrent predicates, etc. on the
  fly.

- The following properties should hold at call time:
  Atom is a free variable. (term_typing:var/1)

- The following properties hold upon exit:
  Atom is an atom. (basic_props:atm/1)

48.3 Documentation on internals (prolog_sys)

time_option/1:  
Usage: time_option(M)

- Description: Options to get information about execution time. M must be one of runtime, usertime, systemtime or walltime.

memory_option/1:  
Usage: memory_option(M)

- Description: Options to get information about memory usage.

garbage_collection_option/1:  
Usage: garbage_collection_option(M)

- Description: Options to get information about garbage collection.

symbol_option/1:  
Usage: symbol_option(M)

- Description: Option to get information about the number of symbols in the program.

time_result/1:  
Usage: time_result(Result)

- Description: Result is a two-element list of numbers. The first number is the time
  since the start of the execution; the second number is the time since the previous
  consult to time.

click_result/1:  
Usage: click_result(Result)

- Description: Result is a two-element list of numbers. The first number is the number
  of clicks since the start of the execution; the second number is the number of clicks
  since the previous consult to click.
clockfreq_result/1:  
Usage: clockfreq_result(Result)

Description: Result is a number. It gives the frequency in hertzios used by the clock used to get the clicks.

memory_result/1:  
Usage: memory_result(Result)

Description: Result is a two-element list of integers. The first element is the space taken up by the option selected, measured in bytes; the second integer is zero for program space (which grows as necessary), and the amount of free space otherwise.

gc_result/1:  
Usage: gc_result(Result)

Description: Result is a tree-element list of integers, related to garbage collection and memory management. When stack_shifts is selected, the first one is the number of shifts (reallocations) of the local stack; the second is the number of shifts of the trail, and the third is the time spent in these shifts. When garbage_collection is selected, the numbers are, respectively, the number of garbage collections performed, the number of bytes freed, and the time spent in garbage collection.

symbol_result/1:  
Usage: symbol_result(Result)

Description: Result is a two-element list of integers. The first one is the number of atom, functor, and predicate names in the symbol table. The second is the number of predicates known to be defined (although maybe without clauses).

48.4 Known bugs and planned improvements (prolog_sys)

- The space used by the process is not measured here: process data, code, and stack also take up memory. The memory reported for atoms is not what is actually used, but the space used up by the hash table (which is enlarged as needed).
49 DEC-10 Prolog file IO

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#273 (2004/1/5, 20:14:41 CET)
This module implements the support for DEC-10 Prolog style file I/O.

49.1 Usage and interface (dec10_io)

- Library usage:
  :- use_module(library(dec10_io)).
- Exports:
  - Predicates:
    see/1, seeing/1, seen/0, tell/1, telling/1, told/0, close_file/1.

49.2 Documentation on exports (dec10_io)

see/1: No further documentation available for this predicate.

seeing/1: No further documentation available for this predicate.

seen/0: No further documentation available for this predicate.

tell/1: No further documentation available for this predicate.

telling/1: No further documentation available for this predicate.

told/0: No further documentation available for this predicate.

close_file/1: No further documentation available for this predicate.
Chapter 50: Quintus-like internal database

Author(s): The CLIP Group.

Version: 1.11#90 (2003/12/21, 2:20:48 CET)

The predicates described in this section were introduced in early implementations of Prolog to provide efficient means of performing operations on large quantities of data. The introduction of indexed dynamic predicates have rendered these predicates obsolete, and the sole purpose of providing them is to support existing code. There is no reason whatsoever to use them in new code.

These predicates store arbitrary terms in the database without interfering with the clauses which make up the program. The terms which are stored in this way can subsequently be retrieved via the key on which they were stored. Many terms may be stored on the same key, and they can be individually accessed by pattern matching. Alternatively, access can be achieved via a special identifier which uniquely identifies each recorded term and which is returned when the term is stored.

50.1 Usage and interface (old_database)

- Library usage:
  ```prolog
  :- use_module(library(old_database)).
  ```
- Exports:
  - Predicates:
    - recorda/3, recordz/3, recorded/3, current_key/2.

50.2 Documentation on exports (old_database)

recorda/3:

```
recorda(Key, Term, Ref)
```

The term `Term` is recorded in the internal database as the first item for the key `Key`, where `Ref` is its implementation-defined identifier. The key must be given, and only its principal functor is significant. Any uninstantiated variables in the `Term` will be replaced by new private variables, along with copies of any subgoals blocked on these variables.

Usage: recorda(Key, Term, Ref)

- The following properties should hold at call time:
  - Key is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Ref is a free variable. (term_typing:var/1)
- The following properties hold globally:
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

recordz/3:

```
recordz(Key, Term, Ref)
```

Like `recorda/3`, except that the new term becomes the last item for the key `Key`.

Usage: recordz(Key, Term, Ref)
The following properties should hold at call time:

- **Key** is currently a term which is not a free variable.  
  (**term_typing:nonvar/1**) 
- **Ref** is a free variable.  
  (**term_typing:var/1**)

The following properties hold globally:

- This predicate is understood natively by CiaoPP.  
  (**basic_props:native/1**) 

**recorded/3:**

**PREDICATE**

```prolog
recorded(Key, Term, Ref)
```

The internal database is searched for terms recorded under the key `Key`. These terms are successively unified with `Term` in the order they occur in the database. At the same time, `Ref` is unified with the implementation-defined identifier uniquely identifying the recorded item. If the key is instantiated to a compound term, only its principal functor is significant. If the key is uninstantiated, all terms in the database are successively unified with `Term` in the order they occur.

**Usage:** `recorded(Key, Term, Ref)`

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP.  
    (**basic_props:native/1**) 

**current_key/2:**

**PREDICATE**

```prolog
current_key(KeyName, KeyTerm)
```

`KeyTerm` is the most general form of the key for a currently recorded term, and `KeyName` is the name of that key. This predicate can be used to enumerate in undefined order all keys for currently recorded terms through backtracking.

**Usage:** `current_key(Name, Key)`

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP.  
    (**basic_props:native/1**)
51 ttyout (library)

Version: 0.4#5 (1998/2/24)

51.1 Usage and interface (ttyout)

- Library usage:
  ```prolog
  :- use_module(library(ttyout)).
  ```

- Exports:
  - Predicates:
    - `ttyget/1`, `ttyget1/1`, `ttynl/0`, `ttyput/1`, `ttyskip/1`, `ttytab/1`, `ttyflush/0`, `ttydisplay/1`, `ttydisplayq/1`, `ttyskipexl/0`, `ttyskip_string/1`.

51.2 Documentation on exports (ttyout)

**ttyget/1:**

Usage: `ttyget(X)`

- The following properties hold upon exit:
  - `X` is a non-negative integer. (basic_props:nnegint/1)

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

**ttyget1/1:**

No further documentation available for this predicate.

**ttynl/0:**

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

**ttyput/1:**

Usage: `ttyput(X)`

- The following properties should hold at call time:
  - `X` is a non-negative integer. (basic_props:nnegint/1)

- The following properties hold globally:
  - This predicate is understood natively by CiaoPP. (basic_props:native/1)

**ttyskip/1:**

No further documentation available for this predicate.
**ttytab/1:**
No further documentation available for this predicate.

**ttyflush/0:**
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP.  *(basic Props:native/1)*

**ttydisplay/1:**
No further documentation available for this predicate.

**ttydisplayq/1:**
No further documentation available for this predicate.

**ttyskipeol/0:**
No further documentation available for this predicate.

**ttydisplay_string/1:**
No further documentation available for this predicate.
52 Enabling operators at run-time

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#80 (2003/12/20, 14:33:4 CET)

This library package allows the use of the statically defined operators of a module for the reading performed at run-time by the program that uses the module. Simply by using this package the operator definitions appearing in the module are enabled during the execution of the program.

52.1 Usage and interface (runtime_ops)

- Library usage:
  
  ```prolog
  :- use_package(runtime_ops).
  ```
  or
  ```prolog
  :- module(...,...,[runtime_ops]).
  ```

- Other modules used:
  - System library modules:
    ```prolog
    operators.
    ```
Author(s): The CLIP Group.

Ciao allows annotating the program code with assertions. Such assertions include type and instantiation mode declarations, but also more general properties as well as comments in the style of the literate programming. These assertions document predicates (and modules and whole applications) and can be used by the Ciao preprocessor/compiler while debugging and optimizing the program or library, and by the Ciao documenter to build the program or library reference manual.
53 The Ciao assertion package

Author(s): Manuel Hermenegildo, Francisco Bueno, German Puebla.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.5#8 (1999/12/9, 21:1:11 MET)

The assertions package adds a number of new declaration definitions and new operator definitions which allow including program assertions in user programs. Such assertions can be used to describe predicates, properties, modules, applications, etc. These descriptions can be formal specifications (such as preconditions and post-conditions) or machine-readable textual comments.

This module is part of the assertions library. It defines the basic code-related assertions, i.e., those intended to be used mainly by compilation-related tools, such as the static analyzer or the run-time test generator.

Giving specifications for predicates and other program elements is the main functionality documented here. The exact syntax of comments is described in the autodocumenter (lpdoc [Knu84,Her99]) manual, although some support for adding machine-readable comments in assertions is also mentioned here.

There are two kinds of assertions: predicate assertions and program point assertions. All predicate assertions are currently placed as directives in the source code, i.e., preceded by `:-`. Program point assertions are placed as goals in clause bodies.

53.1 More info

The facilities provided by the library are documented in the description of its component modules. This documentation is intended to provide information only at a “reference manual” level. For a more tutorial introduction to the subject and some more examples please see the document “An Assertion Language for Debugging of Constraint Logic Programs (Technical Report CLIP2/97.1)”. The assertion language implemented in this library is modeled after this design document, although, due to implementation issues, it may differ in some details. The purpose of this manual is to document precisely what the implementation of the library supports at any given point in time.

53.2 Some attention points

- Formatting commands within text strings: many of the predicates defined in these modules include arguments intended for providing textual information. This includes titles, descriptions, comments, etc. The type of this argument is a character string. In order for the automatic generation of documentation to work correctly, this character string should adhere to certain conventions. See the description of the docstring/1 type/grammar for details.

- Referring to variables: In order for the automatic documentation system to work correctly, variable names (for example, when referring to arguments in the head patterns of pred declarations) must be surrounded by an @var command. For example, @var{VariableName} should be used for referring to the variable “VariableName”, which will appear then formatted as follows: VariableName. See the description of the docstring/1 type/grammar for details.
53.3 Usage and interface (assertions)

- **Library usage:**
  The recommended procedure in order to make use of assertions in user programs is to include the `assertions` syntax library, using one of the following declarations, as appropriate:
  
  ```prolog
  :- module(...,...,[assertions]).
  :- include(library(assertions)).
  :- use_package([assertions]).
  ```

- **Exports:**
  - **Predicates:**
    
    - `check/1`, `trust/1`, `true/1`, `false/1`.

- **New operators defined:**
  
  ```prolog
  =>/2 [975,xfx], ::/2 [978,xfx], dec1/1 [1150,fx], dec1/2 [1150,xfx], pred/1 [1150,fx], pred/2 [1150,xfx], prop/1 [1150,fx], prop/2 [1150,xfx], modedef/1 [1150,fx], calls/1 [1150,fx], calls/2 [1150,xfx], success/1 [1150,fx], success/2 [1150,xfx], comp/1 [1150,fx], comp/2 [1150,xfx], entry/1 [1150,fx].
  ```

- **New declarations defined:**
  
  ```prolog
  pred/1, pred/2, calls/1, calls/2, success/1, success/2, comp/1, comp/2, prop/1, prop/2, entry/1, modedef/1, decl/1, decl/2, comment/2.
  ```

- **Other modules used:**
  - **System library modules:**
    
    `assertions/assertions_props`.

53.4 Documentation on new declarations (assertions)

**pred/1:**

This assertion provides information on a predicate. The body of the assertion (its only argument) contains properties or comments in the formats defined by `assertions_props:assrt_body/1`.

More than one of these assertions may appear per predicate, in which case each one represents a possible "mode" of use (usage) of the predicate. The exact scope of the usage is defined by the properties given for calls in the body of each assertion (which should thus distinguish the different usages intended). All of them together cover all possible modes of usage.

For example, the following assertions describe (all the and the only) modes of usage of predicate `length/2` (see lists):

```prolog
:- pred length(L,N) : list * var => list * integer
  # "Computes the length of L.".
:- pred length(L,N) : var * integer => list * integer
  # "Outputs L of length N.".
:- pred length(L,N) : list * integer => list * integer
  # "Checks that L is of length N.".
```

**Usage:**

```prolog
:- pred(AssertionBody).
- The following properties should hold at call time:
  
  `AssertionBody` is an assertion body. (assertions_props:assrt_body/1)
```
pred/2: DECLARATION
This assertion is similar to a pred/1 assertion but it is explicitly qualified. Non-qualified pred/1 assertions are assumed the qualifier check.
Usage: :- pred(AssertionStatus, AssertionBody).
  - The following properties should hold at call time:
    AssertionStatus is an acceptable status for an assertion. (assertions_props:assrt_status/1)
    AssertionBody is an assertion body. (assertions_props:assrt_body/1)

calls/1: DECLARATION
This assertion is similar to a pred/1 assertion but it only provides information about the calls to a predicate. If one or several calls assertions are given they are understood to describe all possible calls to the predicate.
For example, the following assertion describes all possible calls to predicate is/2 (see arithmetic):
  :- calls is(term,arithexpression).
Usage: :- calls(AssertionBody).
  - The following properties should hold at call time:
    AssertionBody is a call assertion body. (assertions_props:c_assrt_body/1)

calls/2: DECLARATION
This assertion is similar to a calls/1 assertion but it is explicitly qualified. Non-qualified calls/1 assertions are assumed the qualifier check.
Usage: :- calls(AssertionStatus, AssertionBody).
  - The following properties should hold at call time:
    AssertionStatus is an acceptable status for an assertion. (assertions_props:assrt_status/1)
    AssertionBody is a call assertion body. (assertions_props:c_assrt_body/1)
success/1: DECLARATION
This assertion is similar to a pred/1 assertion but it only provides information about the answers to a predicate. The described answers might be conditioned to a particular way of calling the predicate.
For example, the following assertion specifies the answers of the length/2 predicate if it is called as in the first mode of usage above (note that the previous pred assertion already conveys such information, however it also compelled the predicate calls, while the success assertion does not):
  :- success length(L,N) : list * var => list * integer.
  - The following properties should hold at call time:
    AssertionBody is a predicate assertion body. (assertions_props:s_assrt_body/1)
success/2:

This assertion is similar to a success/1 assertion but it is explicitly qualified. Non-qualified success/1 assertions are assumed the qualifier check.


- The following properties should hold at call time:
  - AssertionStatus is an acceptable status for an assertion. (assertions_props:assrt_status/1)
  - AssertionBody is a predicate assertion body. (assertions_props:s_assrt_body/1)

comp/1:

This assertion is similar to a pred/1 assertion but it only provides information about the global execution properties of a predicate (note that such kind of information is also conveyed by pred assertions). The described properties might be conditioned to a particular way of calling the predicate.

For example, the following assertion specifies that the computation of append/3 (see lists) will not fail if it is called as described (but does not compel the predicate to be called that way):

:- comp append(Xs,Ys,Zs) : var * var * var + not_fail.

Usage: :- comp(AssertionBody).

- The following properties should hold at call time:
  - AssertionBody is a comp assertion body. (assertions_props:g_assrt_body/1)

comp/2:

This assertion is similar to a comp/1 assertion but it is explicitly qualified. Non-qualified comp/1 assertions are assumed the qualifier check.

Usage: :- comp(AssertionStatus, AssertionBody).

- The following properties should hold at call time:
  - AssertionStatus is an acceptable status for an assertion. (assertions_props:assrt_status/1)
  - AssertionBody is a comp assertion body. (assertions_props:g_assrt_body/1)

prop/1:

This assertion is similar to a pred/1 assertion but it flags that the predicate being documented is also a “property.”

Properties are standard predicates, but which are **guaranteed to terminate for any possible instantiation state of their argument(s)**, do not perform side-effects which may interfere with the program behaviour, and do not further instantiate their arguments or add new constraints.

Provided the above holds, properties can thus be safely used as run-time checks. The program transformation used in ciaoopp for run-time checking guarantees the third requirement. It also performs some basic checks on properties which in most cases are enough for the second requirement. However, it is the user’s responsibility to guarantee termination of the properties defined. (See also Chapter 55 [Declaring regular types], page 283 for some considerations applicable to writing properties.)
The set of properties is thus a strict subset of the set of predicates. Note that properties can be used to describe characteristics of arguments in assertions and they can also be executed (called) as any other predicates.

**Usage:** :- prop(AssertionBody).
- *The following properties should hold at call time:*
  
  AssertionBody is an assertion body.  
  
  **prop/2:**

<table>
<thead>
<tr>
<th>DECLAREATION</th>
</tr>
</thead>
</table>
  
  This assertion is similar to a prop/1 assertion but it is explicitly qualified. Non-qualified prop/1 assertions are assumed the qualifier check.

  **Usage:** :- prop(AssertionStatus, AssertionBody).
- *The following properties should hold at call time:*
  
  AssertionStatus is an acceptable status for an assertion.  
  AssertionBody is an assertion body.

  **entry/1:**

<table>
<thead>
<tr>
<th>DECLAREATION</th>
</tr>
</thead>
</table>
  
  This assertion provides information about the external calls to a predicate. It is identical syntactically to a calls/1 assertion. However, they describe only external calls, i.e., calls to the exported predicates of a module from outside the module, or calls to the predicates in a non-modular file from other files (or the user).

  These assertions are trusted by the compiler. As a result, if their descriptions are erroneous they can introduce bugs in programs. Thus, entry/1 assertions should be written with care.

  An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program. The main use is in providing information on the ways in which exported predicates of a module will be called from outside the module. This will greatly improve the precision of the analyzer, which otherwise has to assume that the arguments that exported predicates receive are any arbitrary term.

  **Usage:** :- entry(AssertionBody).
- *The following properties should hold at call time:*
  
  AssertionBody is a call assertion body.

  **modedef/1:**

<table>
<thead>
<tr>
<th>DECLAREATION</th>
</tr>
</thead>
</table>
  
  This assertion is used to define modes. A mode defines in a compact way a set of call and success properties. Once defined, modes can be applied to predicate arguments in assertions. The meaning of this application is that the call and success properties defined by the mode hold for the argument to which the mode is applied. Thus, a mode is conceptually a “property macro”.

  The syntax of mode definitions is similar to that of pred declarations. For example, the following set of assertions:

  
  :~ modedef +A : nonvar(A) # "A is bound upon predicate entry.".

  :~ pred p(+A,B) : integer(A) => ground(B).

  is equivalent to:
:- pred p(A,B) : (nonvar(A),integer(A)) => ground(B)
  # "A is bound upon predicate entry.".
Usage: :: modedef(AssertionBody).
- The following properties should hold at call time:
  AssertionBody is an assertion body. (assertions_props:assrt_body/1)

decl/1: DECLARATION
This assertion is similar to a pred/1 assertion but it is used for declarations instead than for predicates.
Usage: :: decl(AssertionBody).
- The following properties should hold at call time:
  AssertionBody is an assertion body. (assertions_props:assrt_body/1)

decl/2: DECLARATION
This assertion is similar to a decl/1 assertion but it is explicitly qualified. Non-qualified decl/1 assertions are assumed the qualifier check.
Usage: :: decl(AssertionStatus, AssertionBody).
- The following properties should hold at call time:
  AssertionStatus is an acceptable status for an assertion. (assertions_props:assrt_status/1)
  AssertionBody is an assertion body. (assertions_props:assrt_body/1)

comment/2: DECLARATION
Usage: :: comment(Pred, Comment).
- Description: This assertion gives a text Comment for a given predicate Pred.
- The following properties should hold at call time:
  Pred is a head pattern. (assertions_props:head_pattern/1)
  Comment is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by stringcommand/1. See the lpdoc manual for documentation on comments. (assertions_props:docstring/1)

53.5 Documentation on exports (assertions)

check/1: PREDICATE
Usage: check(PropertyConjunction)
- Description: This assertion provides information on a clause program point (position in the body of a clause). Calls to a check/1 assertion can appear in the body of a clause in any place where a literal can normally appear. The property defined by PropertyConjunction should hold in all the run-time stores corresponding to that program point. See also (undefined) [Run-time checking of assertions], page (undefined).
The following properties should hold at call time:

PropertyConjunction is either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

(assertions_props:property_conjunction/1)

Trust/1:

Usage: trust(PropertyConjunction)

Description: This assertion also provides information on a clause program point. It is identical syntactically to a check/1 assertion. However, the properties stated are not taken as something to be checked but are instead trusted by the compiler. While the compiler may, in some cases, detect an inconsistency between a trust/1 assertion and the program, in all other cases the information given in the assertion will be taken to be true. As a result, if these assertions are erroneous they can introduce bugs in programs. Thus, trust/1 assertions should be written with care.

An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program (either because the information is not present or because the analyzer being used is not precise enough). In particular, providing information on external predicates which may not be accessible at the time of compiling the module can greatly improve the precision of the analyzer. This can be easily done with trust assertion.

The following properties should hold at call time:

PropertyConjunction is either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

(assertions_props:property_conjunction/1)

True/1:

Usage: true(PropertyConjunction)

Description: This assertion is identical syntactically to a check/1 assertion. However, the properties stated have been proved to hold by the analyzer. Thus, these assertions often represent the analyzer output.

The following properties should hold at call time:

PropertyConjunction is either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

(assertions_props:property_conjunction/1)

False/1:

Usage: false(PropertyConjunction)

Description: This assertion is identical syntactically to a check/1 assertion. However, the properties stated have been proved not to hold by the analyzer. Thus, these assertions often represent the analyzer output.

The following properties should hold at call time:

PropertyConjunction is either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

(assertions_props:property_conjunction/1)
Chapter 54: Types and properties related to assertions

54 Types and properties related to assertions

Author(s): Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#156 (2001/11/24, 13:23:30 CET)

This module is part of the assertions library. It provides the formal definition of the syntax of several forms of assertions and describes their meaning. It does so by defining types and properties related to the assertions themselves. The text describes, for example, the overall fields which are admissible in the bodies of assertions, where properties can be used inside these bodies, how to combine properties for a given predicate argument (e.g., conjunctions), etc. and provides some examples.

54.1 Usage and interface (assertions_props)

Library usage:
:- use_module(library(assertions_props)).
Exports:
- Properties:
  head_pattern/1, nabody/1, docstring/1.
- Regular Types:
  assrt_body/1, complex_arg_property/1, property_conjunction/1, property_starterm/1, complex_goal_property/1, dictionary/1, c_assrt_body/1, s_assrt_body/1, g_assrt_body/1, assrt_status/1, assrt_type/1, predfunctor/1, propfunctor/1.
- Other modules used:
  System library modules:
  dcg_expansion.

54.2 Documentation on exports (assertions_props)

assrt_body/1: REGTYPE
This predicate defines the different types of syntax admissible in the bodies of pred/1, decl/1, etc. assertions. Such a body is of the form:
Pr [:: DP] [: CP] [=> AP] [+ GP] [# CO]

where (fields between [...] are optional):
- Pr is a head pattern (head_pattern/1) which describes the predicate or property and possibly gives some implicit call/answer information.
- DP is a (possibly empty) complex argument property (complex_arg_property/1) which expresses properties which are compatible with the predicate, i.e., instantiations made by the predicate are compatible with the properties in the sense that applying the property at any point to would not make it fail.
- CP is a (possibly empty) complex argument property (complex_arg_property/1) which applies to the calls to the predicate.
• AP is a (possibly empty) complex argument property (complex_arg_property/1) which applies to the answers to the predicate (if the predicate succeeds). These only apply if the (possibly empty) properties given for calls in the assertion hold.
• GP is a (possibly empty) complex goal property (complex_goal_property/1) which applies to the whole execution of a call to the predicate. These only apply if the (possibly empty) properties given for calls in the assertion hold.
• CO is a comment string (docstring/1). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see stringcommand/1). See the lpdoc manual for documentation on assertion comments.

Usage: assert_body(X)
– Description: X is an assertion body.

head_pattern/1: PROPERTY
A head pattern can be a predicate name (functor/arity) (predname/1) or a term. Thus, both p/3 and p(A,B,C) are valid head patterns. In the case in which the head pattern is a term, each argument of such a term can be:
• A variable. This is useful in order to be able to refer to the corresponding argument positions by name within properties and in comments. Thus, p(Input,Parameter,Output) is a valid head pattern.
• A variable, as above, but preceded by a “mode.” This mode determines in a compact way certain call or answer properties. For example, the head pattern p(Input,+Parameter,Output) is valid, as long as +/1 is declared as a mode. Acceptable modes are documented in library(basicmodes) and library(isomodes). User defined modes are documented in modedef/1.
• Any term. In this case this term determines the instantiation state of the corresponding argument position of the predicate calls to which the assertion applies.
• A ground term preceded by a “mode.” The ground term determines a property of the corresponding argument. The mode determines if it applies to the calls and/or the successes. The actual property referred to is that given by the term but with one more argument added at the beginning, which is a new variable which, in a rewriting of the head pattern, appears at the argument position occupied by the term. For example, the head pattern p(Input,+list(int),Output) is valid for mode +/1 defined in library(isomodes), and equivalent in this case to having the head pattern p(Input,A,Output) and stating that the property list(A,int) holds for the calls of the predicate.
• Any term preceded by a “mode.” In this case, only one variable is admitted, it has to be the first argument of the mode, and it represents the argument position. I.e., it plays the role of the new variable mentioned above. Thus, no rewriting of the head pattern is performed in this case. For example, the head pattern p(Input,+((Parameter,list(int)),Output) is valid for mode +/2 defined in library(isomodes), and equivalent in this case to having the head pattern p(Input,Parameter,Output) and stating that the property list(Parameter,int) holds for the calls of the predicate.

Usage: head_pattern(Pr)
– Description: Pr is a head pattern.
complex_arg_property/1:

complex_arg_property(Props)

Props is a (possibly empty) complex argument property. Such properties can appear in two formats, which are defined by property_conjunction/1 and property_starterm/1 respectively. The two formats can be mixed provided they are not in the same field of an assertion. I.e., the following is a valid assertion:

:- pred foo(X,Y) : nonvar * var => (ground(X),ground(Y)).

Usage: complex_arg_property(Props)

Description: Props is a (possibly empty) complex argument property

property_conjunction/1:

This type defines the first, unabridged format in which properties can be expressed in the bodies of assertions. It is essentially a conjunction of properties which refer to variables. The following is an example of a complex property in this format:

• (integer(X),list(Y,Integer)): X has the property integer/1 and Y has the property list/2, with second argument integer.

Usage: property_conjunction(Props)

Description: Props is either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

property_starterm/1:

This type defines a second, compact format in which properties can be expressed in the bodies of assertions. A property_starterm/1 is a term whose main functor is */2 and, when it appears in an assertion, the number of terms joined by */2 is exactly the arity of the predicate it refers to. A similar series of properties as in property_conjunction/1 appears, but the arity of each property is one less: the argument position to which they refer (first argument) is left out and determined by the position of the property in the property_starterm/1. The idea is that each element of the */2 term corresponds to a head argument position. Several properties can be assigned to each argument position by grouping them in curly brackets. The following is an example of a complex property in this format:

• integer * list(integer): the first argument of the procedure (or function, or ...) has the property integer/1 and the second one has the property list/2, with second argument integer.

• {integer,var} * list(integer): the first argument of the procedure (or function, or ...) has the properties integer/1 and var/1 and the second one has the property list/2, with second argument integer.

Usage: property_starterm(Props)

Description: Props is either a term or several terms separated by */2. The main functor of each of those terms corresponds to that of the definition of a property, and the arity should be one less than in the definition of such property. All arguments of each such term are ground.
**complex_goal_property/1:**

```prolog
complex_goal_property(Props)
```

`Props` is a (possibly empty) complex goal property. Such properties can be either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. Such properties apply to all executions of all goals of the predicate which comply with the assertion in which the `Props` appear.

The arguments of the terms in `Props` are implicitly augmented with a first argument which corresponds to a goal of the predicate of the assertion in which the `Props` appear. For example, the assertion

```prolog
:- comp var(A) + not_further_inst(A).
```

has property `not_further_inst/1` as goal property, and establishes that in all executions of `var(A)` it should hold that `not_further_inst(var(A),A)`.  

**Usage:** `complex_goal_property(Props)`

- **Description:** `Props` is either a term or a conjunction of terms. The main functor and arity of each of those terms corresponds to the definition of a property. A first implicit argument in such terms identifies goals to which the properties apply.

**nabody/1:**

**Usage:** `nabody(ABody)`

- **Description:** `ABody` is a normalized assertion body.

**dictionary/1:**

**Usage:** `dictionary(D)`

- **Description:** `D` is a dictionary of variable names.

**c_assrt_body/1:**

This predicate defines the different types of syntax admissible in the bodies of `call/1`, `entry/1`, etc. assertions. The following are admissible:

```prolog
Pr : CP#[ CO]
```

where (fields between `[…]` are optional):

- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the calls to the predicate.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

**Usage:** `c_assrt_body(X)`

- **Description:** `X` is a call assertion body.
s_assert_body/1:

This predicate defines the different types of syntax admissible in the bodies of \texttt{pred/1}, \texttt{func/1}, etc. assertions. The following are admissible:

\begin{align*}
\text{Pr} : \text{CP} & => \text{AP} \# \text{CO} \\
\text{Pr} : \text{CP} & => \text{AP} \\
\text{Pr} & => \text{AP} \# \text{CO} \\
\text{Pr} & => \text{AP}
\end{align*}

where:

- \text{Pr} is a head pattern (\texttt{head_pattern/1}) which describes the predicate or property and possibly gives some implicit call/answer information.
- \text{CP} is a (possibly empty) complex argument property (\texttt{complex_arg_property/1}) which applies to the \texttt{calls} to the predicate.
- \text{AP} is a (possibly empty) complex argument property (\texttt{complex_arg_property/1}) which applies to the \texttt{answers} to the predicate (if the predicate succeeds). These only apply if the (possibly empty) properties given for calls in the assertion hold.
- \text{CO} is a comment string (\texttt{docstring/1}). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see \texttt{stringcommand/1}).

The format of the different parts of the assertion body are given by \texttt{n_assert_body/5} and its auxiliary types.

Usage: \texttt{s_assert_body(X)}

- Description: \texttt{X} is a predicate assertion body.

---

\textbf{g_assert_body/1:}

This predicate defines the different types of syntax admissible in the bodies of \texttt{comp/1} assertions. The following are admissible:

\begin{align*}
\text{Pr} : \text{CP} + \text{GP} & => \text{AP} \# \text{CO} \\
\text{Pr} : \text{CP} + \text{GP} \\
\text{Pr} + \text{GP} & => \text{AP} \# \text{CO} \\
\text{Pr} + \text{GP}
\end{align*}

where:

- \text{Pr} is a head pattern (\texttt{head_pattern/1}) which describes the predicate or property and possibly gives some implicit call/answer information.
- \text{CP} is a (possibly empty) complex argument property (\texttt{complex_arg_property/1}) which applies to the \texttt{calls} to the predicate.
- \text{GP} contains (possibly empty) complex goal property (\texttt{complex_goal_property/1}) which applies to the whole execution of a call to the predicate. These only apply if the (possibly empty) properties given for calls in the assertion hold.
- \text{CO} is a comment string (\texttt{docstring/1}). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see \texttt{stringcommand/1}).

The format of the different parts of the assertion body are given by \texttt{n_assert_body/5} and its auxiliary types.

Usage: \texttt{g_assert_body(X)}

- Description: \texttt{X} is a comp assertion body.
**assrt_status/1:**

The types of assertion status. They have the same meaning as the program-point assertions, and are as follows:

- `assrt_status(true).`
- `assrt_status(false).`
- `assrt_status(check).`
- `assrt_status(checked).`
- `assrt_status(trust).`

**Usage:** `assrt_status(X)`

- **Description:** X is an acceptable status for an assertion.

**assrt_type/1:**

The admissible kinds of assertions:

- `assrt_type(pred).`
- `assrt_type(prop).`
- `assrt_type(decl).`
- `assrt_type(func).`
- `assrt_type(calls).`
- `assrt_type(success).`
- `assrt_type(comp).`
- `assrt_type(entry).`
- `assrt_type(modedef).`

**Usage:** `assrt_type(X)`

- **Description:** X is an admissible kind of assertion.

**predfunctor/1:**

**Usage:** `predfunctor(X)`

- **Description:** X is a type of assertion which defines a predicate.

**propfunctor/1:**

**Usage:** `propfunctor(X)`

- **Description:** X is a type of assertion which defines a property.

**docstring/1:**

**Usage:** `docstring(String)`

- **Description:** String is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments.
55 Declaring regular types

Author(s): Manuel Hermenegildo, Pedro Lopez, Francisco Bueno.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.5#9 (1999/12/9, 21:57:42 MET)

This library package adds some new declaration definitions and new operator definitions to user programs. These new declarations and operators provide some very simple syntactic sugar to support regular type definitions in source code. Regular types are just properties which have the additional characteristic of being regular types (basic_props:regtype/1).

For example, this library package allows writing:

```prolog
:- regtype tree(X) # "X is a tree.".
```

instead of the more cumbersome:

```prolog
:- prop tree(X) + regtype # "X is a tree.".
```

Regular types can be used as properties to describe predicates and play an essential role in program debugging (see the Ciao Prolog preprocessor (ciaopp) manual).

In this chapter we explain some general considerations worth taking into account when writing properties in general, not just regular types. The exact syntax of regular types is also described.

55.1 Defining properties

Given the classes of assertions in the Ciao assertion language, there are two fundamental classes of properties. Properties used in assertions which refer to execution states (i.e., calls/1, success/1, and the like) are called properties of execution states. Properties used in assertions related to computations (i.e., comp/1) are called properties of computations. Different considerations apply when writing a property of the former or of the later kind.

Consider a definition of the predicate string_concat/3 which concatenates two character strings (represented as lists of ASCII codes):

```prolog
string_concat([],L,L).
string_concat([X|Xs],L,[X|NL]):- string_concat(Xs,L,NL).
```

Assume that we would like to state in an assertion that each argument “is a list of integers.” However, we must decide which one of the following two possibilities we mean exactly: “the argument is instantiated to a list of integers” (let us call this property instantiated_to_intlist/1), or “if any part of the argument is instantiated, this instantiation must be compatible with the argument being a list of integers” (we will call this property compatible_with_intlist/1). For example, instantiated_to_intlist/1 should be true for the terms [] and [1,2], but should not for X, [a,2], and [X,2]. In turn, compatible_with_intlist/1 should be true for [], X, [1,2], and [X,2], but should not be for [X|1], [a,2], and 1. We refer to properties such as instantiated_to_intlist/1 above as instantiation properties and to those such as compatible_with_intlist/1 as compatibility properties (corresponding to the traditional notions of “instantiation types” and “compatibility types”).

It turns out that both of these notions are quite useful in practice. In the example above, we probably would like to use compatible_with_intlist/1 to state that on success of string_concat/3 all three argument must be compatible with lists of integers in an assertion like:

```prolog
:- success string_concat(A,B,C) => ( compatible_with_intlist(A),
                                    compatible_with_intlist(B),
                                    compatible_with_intlist(C) ).
```
With this assertion, no error will be flagged for a call to `string_concat/3` such as `string_concat([20],L,R)`, which on success produces the resulting atom `string_concat([20],L,[20|L])`, but a call `string_concat([],a,R)` would indeed flag an error.

On the other hand, and assuming that we are running on a Prolog system, we would probably like to use `instantiated_to_intlist/1` for `sumlist/2` as follows:

```prolog
:- calls sumlist(L,N) : instantiated_to_intlist(L).
```

```prolog
sumlist([],0).
sumlist([X|R],S) :- sumlist(R,PS), S is PS+X.
```

to describe the type of calls for which the program has been designed, i.e., those in which the first argument of `sumlist/2` is indeed a list of integers.

The property `instantiated_to_intlist/1` might be written as in the following (Prolog) definition:

```prolog
:- prop instantiated_to_intlist/1.
instantiated_to_intlist(X) :- nonvar(X), instantiated_to_intlist_aux(X).
instantiated_to_intlist_aux([]).
instantiated_to_intlist_aux([X|T]) :- integer(X), instantiated_to_intlist(T).
```

(Recall that the Prolog builtin `integer/1` itself implements an instantiation check, failing if called with a variable as the argument.)

The property `compatible_with_intlist/1` might in turn be written as follows (also in Prolog):

```prolog
:- prop compatible_with_intlist/1.
compatible_with_intlist(X) :- var(X).
compatible_with_intlist(X) :- nonvar(X), compatible_with_intlist_aux(X).
compatible_with_intlist_aux([]).
compatible_with_intlist_aux([X|T]) :- int_compat(X), compatible_with_intlist(T).
```

```
int_compat(X) :- var(X).
int_compat(X) :- nonvar(X), integer(X).
```

Note that these predicates meet the criteria for being properties and thus the `prop/1` declaration is correct.

Ensuring that a property meets the criteria for “not affecting the computation” can sometimes make its coding somewhat tedious. In some ways, one would like to be able to write simply:

```prolog
intlist([]).
intlist([X|R]) :- int(X), intlist(R).
```

(Incidentally, note that the above definition, provided that it suits the requirements for being a property and that `int/1` is a regular type, meets the criteria for being a regular type. Thus, it could be declared `:- regtype intlist/1.`)

But note that (independently of the definition of `int/1`) the definition above is not the correct instantiation check, since it would succeed for a call such as `intlist(X)`. In fact, it is not strictly correct as a compatibility property either, because, while it would fail or succeed
as expected, it would perform instantiations (e.g., if called with \texttt{intlist(X)} it would bind \texttt{X} to \texttt{[]}). In practice, it is convenient to provide some run-time support to aid in this task.

The run-time support of the Ciao system (see \texttt{(undefined)}} \cite{Run-time checking of assertions}, \page{\texttt{(undefined)}}) ensures that the execution of properties is performed in such a way that properties written as above can be used directly as instantiation checks. Thus, writing:

\begin{verbatim}
:- calls sumlist(L,N) : intlist(L).
\end{verbatim}

has the desired effect. Also, the same properties can often be used as compatibility checks by writing them in the assertions as \texttt{compat(Property)} (\texttt{basic_props:compat/1}). Thus, writing:

\begin{verbatim}
:- success string_concat(A,B,C) => ( compat(intlist(A)),
                compat(intlist(B)),
                compat(intlist(C)))
\end{verbatim}

also has the desired effect.

As a general rule, the properties that can be used directly for checking for compatibility should be \textit{downwards closed}, i.e., once they hold they will keep on holding in every state accessible in forwards execution. There are certain predicates which are inherently \textit{instantiation} checks and should not be used as \textit{compatibility} properties nor appear in the definition of a property that is to be used with \texttt{compat}. Examples of such predicates (for Prolog) are \texttt{==, ground, nonvar, integer, atom, >}, etc. as they require a certain instantiation degree of their arguments in order to succeed.

In contrast with properties of execution states, \textit{properties of computations} refer to the entire execution of the call(s) that the assertion relates to. One such property is, for example, \texttt{not_fail/1} (note that although it has been used as in \texttt{:- comp append(Xs,Ys,Zs) + not_fail}, it is in fact read as \texttt{not_fail(append(Xs,Ys,Zs))}; see \texttt{assertions_props:complex_goal_property/1}). For this property, which should be interpreted as “execution of the predicate either succeeds at least once or loops,” we can use the following predicate \texttt{not_fail/1} for run-time checking:

\begin{verbatim}
not_fail(Goal):-
    if( call(Goal),
        true, \%\% then
        warning(Goal) ). \%\% else
\end{verbatim}

where the \texttt{warning/1} (library) predicate simply prints a warning message.

In this simple case, implementation of the predicate is not very difficult using the (non-standard) \texttt{if/3} builtin predicate present in many Prolog systems.

However, it is not so easy to code predicates which check other properties of the computation and we may in general need to program a meta-interpreter for this purpose.
55.2 Usage and interface (regtypes)

- Library usage:
  
  `:- use_package(regtypes).`

  or

  `:- module(...,...,[regtypes]).`

- New operators defined:
  
  `regtype/1 [1150.fx], regtype/2 [1150.xfx].`

- New declarations defined:
  
  `regtype/1, regtype/2.`

- Other modules used:
  
  `System library modules:
  
  assertions/assertions_props.`

55.3 Documentation on new declarations (regtypes)

**DECLARATION**

`regtype/1:`

This assertion is similar to a pred assertion but it flags that the predicate being documented is also a “regular type.” This allows for example checking whether it is in the class of types supported by the type checking and inference modules. Currently, types are properties whose definitions are regular programs.

A regular program is defined by a set of clauses, each of the form:

```
  p(x, v_1, ..., v_n) :- body_1, ..., body_k.
```

where:

1. `x` is a term whose variables (which are called *term variables*) are unique, i.e., it is not allowed to introduce equality constraints between the variables of `x`. For example, `p(f(X, Y)) :- ...` is valid, but `p(f(X, X)) :- ...` is not.
2. in all clauses defining `p/n+1` the terms `x` do not unify except maybe for one single clause in which `x` is a variable.
3. `n >= 0` and `p/n` is a *parametric type functor* (whereas the predicate defined by the clauses is `p/n+1`).
4. `v_1, ..., v_n` are unique variables, which are called *parametric variables*.
5. Each `body_i` is of the form:
   1. `t(z)` where `z` is one of the *term variables* and `t` is a *regular type expression*;
   2. `q(y, t_1, ..., t_m)` where `m >= 0`, `q/m` is a *parametric type functor*, not in the set of functors `/2, ^/2, ./3.

   `t_1, ..., t_m` are *regular type expressions*, and `y` is a *term variable*.
6. Each term variable occurs at most once in the clause’s body (and should be as the first argument of a literal).

A *regular type expression* is either a parametric variable or a parametric type functor applied to some of the parametric variables (but regular type abstractions might also be used in some cases, see *(undefined)* [Meta-properties], page *(undefined)*).

A parametric type functor is a regular type, defined by a regular program, or a basic type. Basic types are defined in Chapter 15 [Basic data types and properties], page 79.
The set of types is thus a well defined subset of the set of properties. Note that types can be used to describe characteristics of arguments in assertions and they can also be executed (called) as any other predicates.

**Usage:** \(-\, \text{regtype(AssertionBody)}.\)

- The following properties should hold at call time:
  
  \textbf{AssertionBody} is an assertion body. \hspace{1cm} (assertions\_props:assrt\_body/1)

\textbf{regtype/2:} \hspace{5cm} \textbf{DECLARATION}

This assertion is similar to a \texttt{regtype/1} assertion but it is explicitly qualified. Non-qualified \texttt{regtype/1} assertions are assumed the qualifier \texttt{check}. Note that checking regular type definitions should be done with the \texttt{ciaopp} preprocessor.

**Usage:** \(-\, \text{regtype(AssertionStatus, AssertionBody)}.\)

- The following properties should hold at call time:
  
  \textbf{AssertionStatus} is an acceptable status for an assertion. \hspace{1cm} (assertions\_props:assrt\_status/1)
  
  \textbf{AssertionBody} is an assertion body. \hspace{1cm} (assertions\_props:assrt\_body/1)
56 Properties which are native to analyzers

Author(s): Francisco Bueno, Manuel Hermenegildo, Pedro Lopez.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

This library contains a set of properties which are natively understood by the different program analyzers of ciaopp. They are used by ciaopp on output and they can also be used as properties in assertions.

56.1 Usage and interface (native_props)

- Library usage:
  ```prolog
  :- use_module(library('assertions/native_props'))
  ```
  or also as a package:
  ```prolog
  :- use_package(nativeprops).
  ```
  Note the different names of the library and the package.

- Exports:
  - Properties:
    - covered/2, linear/1, mshare/1, nonground/1, fails/1, not_fails/1, possibly_fails/1, covered/1, not_covered/1, is_det/1, non_det/1, possibly_nondet/1, mut_exclusive/1, not_mut_exclusive/1, size_lb/2, size_ub/2, steps_lb/2, steps_ub/2, steps/2, finite_solutions/1, terminates/1.

- Other modules used:
  - System library modules:
    - andprolog/andprolog_rt, terms_check, terms_vars, sort, lists.

56.2 Documentation on exports (native.props)

**covered/2:**

PROPERTY

```prolog
covered(X, Y)
```

All variables occurring in \(X\) occur also in \(Y\).

Usage: covered(X, Y)

- Description: \(X\) is covered by \(Y\).
- The following properties hold globally:

  This predicate is understood natively by CiaoPP.  

(basic_props:native/1)

**linear/1:**

PROPERTY

```prolog
linear(X)
```

\(X\) is bound to a term which is linear, i.e., if it contains any variables, such variables appear only once in the term. For example, \([1, 2, 3]\) and \(f(A, B)\) are linear terms, while \(f(A, A)\) is not.

Usage: linear(X)

- Description: \(X\) is instantiated to a linear term.
- The following properties hold globally:

  This predicate is understood natively by CiaoPP.  

(basic_props:native/1)
**mshare/1:**

```
mshare(X)
```

**PROPERTY**

X contains all sharing sets [JL88,MH89] which specify the possible variable occurrences in the terms to which the variables involved in the clause may be bound. Sharing sets are a compact way of representing groundness of variables and dependencies between variables. This representation is however generally difficult to read for humans. For this reason, this information is often translated to `ground/1`, `indep/1` and `indep/2` properties, which are easier to read.

**Usage:** `mshare(X)`

- **Description:** The sharing pattern is X.
- **The following properties should hold globally:**
  This predicate is understood natively by CiaoPP as `sharing(X)`. (basic_props:native/2)

**nonground/1:**

```
nonground(X)
```

**PROPERTY**

**Usage:** `nonground(X)`

- **Description:** X is not ground.
- **The following properties should hold globally:**
  This predicate is understood natively by CiaoPP as `not_ground(X)`. (basic_props:native/2)

**fails/1:**

```
fails(X)
```

**PROPERTY**

Calls of the form X fail.

**Usage:** `fails(X)`

- **Description:** Calls of the form X fail.
- **The following properties hold globally:**
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

**not_fails/1:**

```
otin_fails(X)
```

**PROPERTY**

Calls of the form X produce at least one solution, or not terminate [DLGH97].

**Usage:** `not_fails(X)`

- **Description:** All the calls of the form X do not fail.
- **The following properties hold globally:**
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

**possibly_fails/1:**

```
possibly_fails(X)
```

**PROPERTY**

Non-failure is not ensured for any call of the form X [DLGH97]. In other words, nothing can be ensured about non-failure nor termination of such calls.

**Usage:** `possibly_fails(X)`

- **Description:** Non-failure is not ensured for calls of the form X.
Chapter 56: Properties which are native to analyzers

**PROPERTY covered/1:**

covered(X)

For any call of the form X there is at least one clause whose test succeeds (i.e. all the calls of the form X are covered.) [DLGH97].

**Usage:** covered(X)

- **Description:** All the calls of the form X are covered.

**PROPERTY not_covered/1:**

not_covered(X)

There is some call of the form X for which there is not any clause whose test succeeds [DLGH97].

**Usage:** not_covered(X)

- **Description:** Not all of the calls of the form X are covered.

**PROPERTY is_det/1:**

is_det(X)

All calls of the form X are deterministic, i.e. produce at most one solution, or not terminate.

**Usage:** is_det(X)

- **Description:** All calls of the form X are deterministic.

**PROPERTY non_det/1:**

non_det(X)

All calls of the form X are not deterministic, i.e., produce several solutions.

**Usage:** non_det(X)

- **Description:** All calls of the form X are not deterministic.

**PROPERTY possibly_nondet/1:**

possibly_nondet(X)

Non-determinism is not ensured for all calls of the form X. In other words, nothing can be ensured about determinacy nor termination of such calls.

**Usage:** possibly_nondet(X)

- **Description:** Non-determinism is not ensured for calls of the form X.

**PROPERTY mut_exclusive/1:**

mut_exclusive(X)

For any call of the form X at most one clause succeeds, i.e. clauses are pairwise exclusive.

**Usage:** mut_exclusive(X)

- **Description:** For any call of the form X at most one clause succeeds.
not_mut_exclusive/1: \* PROPERTY \\
not_mut_exclusive(X) 
Not for all calls of the form X at most one clause succeeds. I.e. clauses are not disjoint for some call. 
Usage: not_mut_exclusive(X) 
\- Description: Not for all calls of the form X at most one clause succeeds.

size_lb/2: \* PROPERTY 
size_lb(X, Y) 
The minimum size of the terms to which the argument Y is bound to is given by the expression Y. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93]. 
Usage: size_lb(X, Y) 
\- Description: Y is a lower bound on the size of argument X.

size_ub/2: \* PROPERTY 
size_ub(X, Y) 
The maximum size of the terms to which the argument Y is bound to is given by the expression Y. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93]. 
Usage: size_ub(X, Y) 
\- Description: Y is a upper bound on the size of argument X.

steps_lb/2: \* PROPERTY 
steps_lb(X, Y) 
The minimum computation time (in resolution steps) spent by any call of the form X is given by the expression Y [DLGHL97,LGHD96] 
Usage: steps_lb(X, Y) 
\- Description: Y is a lower bound on the cost of any call of the form X.

steps_ub/2: \* PROPERTY 
steps_ub(X, Y) 
The maximum computation time (in resolution steps) spent by any call of the form X is given by the expression Y [DL93,LGHD96] 
Usage: steps_ub(X, Y) 
\- Description: Y is a upper bound on the cost of any call of the form X.

steps/2: \* PROPERTY 
steps(X, Y) 
The time (in resolution steps) spent by any call of the form X is given by the expression Y 
Usage: steps(X, Y) 
\- Description: Y is the cost (number of resolution steps) of any call of the form X.
finite_solutions/1:  
\textit{PROPERTY}  
finite_solutions(X)  
Calls of the form \(X\) produce a finite number of solutions \cite{DLGH97}.  
\textit{Usage: finite_solutions(X)}  
\begin{itemize}  
\item \textit{Description:} All the calls of the form \(X\) have a finite number of solutions.  
\end{itemize}

terminates/1:  
\textit{PROPERTY}  
terminates(X)  
Calls of the form \(X\) always terminate \cite{DLGH97}.  
\textit{Usage: terminates(X)}  
\begin{itemize}  
\item \textit{Description:} All the calls of the form \(X\) terminate.  
\end{itemize}

indep/1:  
\textit{PROPERTY}  
\textit{Usage: indep(X)}  
\begin{itemize}  
\item \textit{Description:} The variables in pairs in \(X\) are pairwise independent.  
\item The following properties hold globally:  
\begin{itemize}  
\item This predicate is understood natively by CiaoPP as \texttt{indep(X)}.  
\item \texttt{basic_props:native/2}  
\end{itemize}  
\end{itemize}

indep/2:  
\textit{PROPERTY}  
\textit{Usage: indep(X, Y)}  
\begin{itemize}  
\item \textit{Description:} \(X\) and \(Y\) do not have variables in common.  
\item The following properties hold globally:  
\begin{itemize}  
\item This predicate is understood natively by CiaoPP as \texttt{indep([X,Y])}.  
\item \texttt{basic_props:native/2}  
\end{itemize}  
\end{itemize}

instance/2:  
\texttt{(UNDOC_REEXPORT)}  
Imported from terms_check (see the corresponding documentation for details).
57 ISO-Prolog modes

Author(s): Daniel Cabeza, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#292 (2004/2/16, 14:52:52 CET)

This file defines the “modes” used in the documentation of the ISO-Prolog standard. See also Chapter 58 [Classical Prolog modes], page 297 for an alternative set of modes.

57.1 Usage and interface (isomodes)

- Library usage:
  :- use_package([assertions,isomodes]).
- New operators defined:
  ?/1 [200,fy], @/1 [200,fy].
- New modes defined:
  +/-1, @/1, -/1, ?/1, */1, +/-2, @/2, -/2, ?/2, */2.

57.2 Documentation on new modes (isomodes)

+/1:
  Usage: + A
  - The following properties are added at call time:
    A is currently a term which is not a free variable. (term_typing:nonvar/1)

@/1:
  Usage: @ A
  - The following properties are added globally:
    A is not further instantiated. (basic_props:not_further_inst/2)

-/1:
  Usage: - A
  - The following properties are added at call time:
    A is a free variable. (term_typing:var/1)

?/1:
  Unspecified argument.

*/1:
  Unspecified argument.
+/2:
Usage: A + X
- The following properties are added at call time:
  undefined:call(X,A) (undefined property)

@/2:
Usage: @(A, X)
- The following properties are added at call time:
  undefined:call(X,A) (undefined property)
- The following properties are added upon exit:
  undefined:call(X,A) (undefined property)
- The following properties are added globally:
  A is not further instantiated. (basic_props:not_further_inst/2)

-/2:
Usage: A - X
- The following properties are added at call time:
  A is a free variable. (term_typing:var/1)
- The following properties are added upon exit:
  undefined:call(X,A) (undefined property)

?/2:
Usage: ?(A, X)
- Call and exit are compatible with:
  undefined:call(X,A) (undefined property)
- The following properties are added upon exit:
  undefined:call(X,A) (undefined property)

*/2:
Usage: A * X
- Call and exit are compatible with:
  undefined:call(X,A) (undefined property)
58 Classical Prolog modes

Author(s): Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 0.8#43 (1999/3/6, 18:39:38 CET)

This file defines a number of very simple “modes” which are frequently useful in programs. These correspond to the modes used in classical Prolog texts with some simple additions. Note that some of these modes use the same symbol as one of the ISO-modes (see Chapter 57 [ISO-Prolog modes], page 295) but with subtly different meaning.

58.1 Usage and interface (basicmodes)

- Library usage:
  ```prolog
  :- use_package([assertions,basicmodes]).
  ```

- New operators defined:
  ```prolog
  ?/1 [500,fx], @/1 [500,fx].
  ```

- New modes defined:
  ```prolog
  +/-1, -/-1, ?/1, @/1, in/1, out/1, go/1, +/-2, -/-2, ?/2, @/2, in/2, out/2, go/2.
  ```

58.2 Documentation on new modes (basicmodes)

+/1: MODE

  Input value in argument.
  Usage: + A
  - *The following properties are added at call time:*
    A is currently a term which is not a free variable. (term_typing:nonvar/1)

-/1: MODE

  No input value in argument.
  Usage: - A
  - *The following properties are added at call time:*
    A is a free variable. (term_typing:var/1)

?/1: MODE

  Unspecified argument.

@/1: MODE

  No output value in argument.
  Usage: @ A
  - *The following properties are added globally:*
    A is not further instantiated. (basic_props:not_further_inst/2)
**in/1:**  
Input argument.  
Usage: `in(A)`  
- The following properties are added at call time:  
  A is currently ground (it contains no variables). (term_typing:ground/1)  
- The following properties are added upon exit:  
  A is currently ground (it contains no variables). (term_typing:ground/1)

**out/1:**  
Output argument.  
Usage: `out(A)`  
- The following properties are added at call time:  
  A is a free variable. (term_typing:var/1)  
- The following properties are added upon exit:  
  A is currently ground (it contains no variables). (term_typing:ground/1)

**go/1:**  
Ground output (input/output argument).  
Usage: `go(A)`  
- The following properties are added upon exit:  
  A is currently ground (it contains no variables). (term_typing:ground/1)

**+/2:**  
Usage: `A + X`  
- Call and exit are compatible with:  
  undefined:call(X,A) (undefined property)  
- The following properties are added at call time:  
  A is currently a term which is not a free variable. (term_typing:nonvar/1)

**-/2:**  
Usage: `A - X`  
- Call and exit are compatible with:  
  undefined:call(X,A) (undefined property)  
- The following properties are added at call time:  
  A is a free variable. (term_typing:var/1)

**?/2:**  
Usage: `?(A, X)`  
- Call and exit are compatible with:  
  undefined:call(X,A) (undefined property)
MODE \( @/2 \):

Usage: \( @(A, X) \)

- Call and exit are compatible with:
  
  \texttt{undefined:call(X,A)}

  (undefined property)

- The following properties are added globally:
  
  \( A \) is not further instantiated. 

  (\texttt{basic_props:not\_further\_inst/2})

MODE \( in/2 \):

Usage: \( in(A, X) \)

- Call and exit are compatible with:
  
  \texttt{undefined:call(X,A)}

  (undefined property)

- The following properties are added at call time:
  
  \( A \) is currently ground (it contains no variables).

  (\texttt{term\_typing:ground/1})

- The following properties are added upon exit:
  
  \( A \) is currently ground (it contains no variables).

  (\texttt{term\_typing:ground/1})

MODE \( out/2 \):

Usage: \( out(A, X) \)

- Call and exit are compatible with:
  
  \texttt{undefined:call(X,A)}

  (undefined property)

- The following properties are added at call time:
  
  \( A \) is a free variable.

  (\texttt{term\_typing:var/1})

- The following properties are added upon exit:
  
  \( A \) is currently ground (it contains no variables).

  (\texttt{term\_typing:ground/1})

MODE \( go/2 \):

Usage: \( go(A, X) \)

- Call and exit are compatible with:
  
  \texttt{undefined:call(X,A)}

  (undefined property)

- The following properties are added upon exit:
  
  \( A \) is currently ground (it contains no variables).

  (\texttt{term\_typing:ground/1})
The Ciao Prolog System
PART VI - Ciao Prolog library miscellanea

Author(s): The CLIP Group.

This part documents several Ciao libraries which provide different useful additional functionalities. Such functionalities include performing operating system calls, gathering statistics from the Prolog engine, file and file name manipulation, error and exception handling, fast reading and writing of terms (marshalling and unmarshalling), file locking, program reporting messages, pretty-printing programs and assertions, a browser of the system libraries, additional expansion utilities, concurrent aggregates, graph visualization, etc.
Chapter 59: Structured stream handling


59.1 Usage and interface (streams)

- Library usage:
  ```prolog
  :- use_module(library(streams)).
  ```
- Exports:
  - Predicates:
    - open_null_stream/1, open_input/2, close_input/1, open_output/2, close_output/1.

59.2 Documentation on exports (streams)

open_null_stream/1: No further documentation available for this predicate.

open_input/2: No further documentation available for this predicate.

close_input/1: No further documentation available for this predicate.

open_output/2: No further documentation available for this predicate.

close_output/1: No further documentation available for this predicate.
60 Dictionaries

Author(s): The CLIP Group.
Version: 1.11#116 (2003/12/22, 18:52:5 CET)

This module provides predicates for implementing dictionaries. Such dictionaries are currently implemented as ordered binary trees of key-value pairs.

60.1 Usage and interface (dict)

- Library usage:
  ```prolog
  :- use_module(library(dict)).
  ```

- Exports:
  - Predicates:
    - `dictionary/5`, `dic_node/2`, `dic_lookup/3`, `dic_lookup/4`, `dic_get/3`, `dic_replace/4`.
  - Properties:
    - `dictionary/1`.

60.2 Documentation on exports (dict)

dictionary/1:  
**PROPERTY**

  **Usage:** `dictionary(D)`
  
  **Description:** `D` is a dictionary.

dictionary/5:  
**PREDICATE**

  **Usage:** `dictionary(D, K, V, L, R)`
  
  **Description:** The dictionary node `D` has key `K`, value `V`, left child `L`, and right child `R`.

dic_node/2:  
**PREDICATE**

  **Usage:** `dic_node(D, N)`
  
  **Description:** `N` is a sub-dictionary of `D`.
  
  **Calls should, and exit will be compatible with:**
  
  `D` is a dictionary.
  
  `N` is a dictionary.

  (dict:dictionary/1)

dic_lookup/3:  
**PREDICATE**

  **Usage:** `dic_lookup(D, K, V)`
  
  **Description:** `D` contains value `V` at key `K`. If it was not already in `D` it is added.
  
  **Calls should, and exit will be compatible with:**
  
  `D` is a dictionary.

  (dict:dictionary/1)
dic_lookup/4:
Usage: dic_lookup(D, K, V, 0)
- Description: Same as dic_lookup(D,K,V). 0 indicates if it was already in D (old) or not (new).
- Calls should, and exit will be compatible with:
  D is a dictionary.

DIC GET/3:
Usage: dic_get(D, K, V)
- Description: D contains value V at key K. Fails if it is not already in D.
- Calls should, and exit will be compatible with:
  D is a dictionary.

Dic replace/4:
Usage: dic_replace(D, K, V, D1)
- Description: D and D1 are identical except for the element at key K, which in D1 contains value V, whatever has (or whether it is) in D.
- Calls should, and exit will be compatible with:
  D is a dictionary.
  D1 is a dictionary.
Chapter 61: String processing

Author(s): Daniel Cabeza.
Version: 0.4#5 (1998/2/24)

This module provides predicates for doing input/output with strings (character code lists) and for including in grammars defining strings.

61.1 Usage and interface (strings)

- Library usage:
  :- use_module(library(strings)).
- Exports:
  - Predicates:
    get_line/2, get_line/1, write_string/2, write_string/1, whitespace/2, whitespace0/2, string/3.

61.2 Documentation on exports (strings)

get_line/2:

get_line(Stream, Line)

Reads from Stream a line of text and unifies Line with it. The end of the line can have UNIX [10] or MS-DOS [13 10] termination, which is not included in Line. At EOF, the term end_of_file is returned.

Usage:
  - Calls should, and exit will be compatible with:
    strings:line(Line) (strings:line/1)
  - The following properties should hold at call time:
    Stream is an open stream. (streams_basic:stream/1)
  - The following properties hold upon exit:
    strings:line(Line) (strings:line/1)

get_line/1:

get_line(Line)

Behaves like current_input(S), get_line(S,Line).

Usage:
  - Calls should, and exit will be compatible with:
    strings:line(Line) (strings:line/1)
  - The following properties hold upon exit:
    strings:line(Line) (strings:line/1)
**write_string/2:**

\[\text{write_string(Stream, String)}\]

Writes \(\text{String}\) onto \(\text{Stream}\).

**Usage:**

- *The following properties should hold at call time:*
  
  \(\text{Stream}\) is an open stream. \(\text{(streams_basic:stream/1)}\)
  
  \(\text{String}\) is a string (a list of character codes). \(\text{(basic_props:string/1)}\)

**write_string/1:**

\[\text{write_string(String)}\]

Behaves like \(\text{current_input(S)}, \text{write_string(S, String)}\).

**Usage:**

- *The following properties should hold at call time:*
  
  \(\text{String}\) is a string (a list of character codes). \(\text{(basic_props:string/1)}\)

**whitespace/2:**

\[\text{whitespace(String, Rest)}\]

In a grammar rule, as \(\text{whitespace/0}\), represents whitespace (a positive number of space (32), tab (9), newline (10) or return (13) characters). Thus, \(\text{Rest}\) is a proper suffix of \(\text{String}\) with one or more whitespace characters removed. An example of use would be:

\[\text{attrs([])} \rightarrow ""\]
\[\text{attrs([N|Ns])} \rightarrow\]
\[\text{whitespace, attr(N), attrs(Ns).}\]

**Usage:**

- *Calls should, and exit will be compatible with:*
  
  \(\text{Rest}\) is a string (a list of character codes). \(\text{(basic_props:string/1)}\)

**whitespace0/2:**

\[\text{whitespace0(String, Rest)}\]

In a grammar rule, as \(\text{whitespace0/0}\), represents possible whitespace (any number of space (32), tab (9), newline (10) or return (13) characters). Thus, \(\text{Rest}\) is \(\text{String}\) or a proper suffix of \(\text{String}\) with one or more whitespace characters removed. An example of use would be:

\[\text{assignment(N,V)} \rightarrow\]
\[\text{variable_name(N), whitespace0, "}", whitespace0, value(V).}\]

**Usage:**
– **Calls should, and exit will be compatible with:**
  *Rest* is a string (a list of character codes).  
  (basic_props:string/1)

– **The following properties should hold at call time:**
  *String* is a string (a list of character codes).  
  (basic_props:string/1)

– **The following properties hold upon exit:**
  *Rest* is a string (a list of character codes).  
  (basic_props:string/1)

---

**string/3:**

PREDICATE

`string(String, Head, Tail)`

In a grammar rule, as `string/1`, represents literally *String*. An example of use would be:

```prolog
double(A) -->
  string(A),
  string(A).
```

**Usage:**

– **Calls should, and exit will be compatible with:**
  *String* is a string (a list of character codes).  
  (basic_props:string/1)
  
  *Head* is a string (a list of character codes).  
  (basic_props:string/1)
  
  *Tail* is a string (a list of character codes).  
  (basic_props:string/1)

– **The following properties hold upon exit:**
  *String* is a string (a list of character codes).  
  (basic_props:string/1)
  
  *Head* is a string (a list of character codes).  
  (basic_props:string/1)
  
  *Tail* is a string (a list of character codes).  
  (basic_props:string/1)

---

**61.3 Documentation on internals (strings)**

**line/1:**

PROPERTY

A property, defined as follows:

```prolog
line(L) :-
  string(L).
  line(end_of_file).
```
Chapter 62: Printing status and error messages

Author(s): The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#233 (2004/5/27, 17:43:49 CEST)

This is a very simple library for printing status and error messages to the console.

62.1 Usage and interface (messages)

- Library usage:
  :- use_module(library(messages)).

- Exports:
  - Predicates:
    error_message/1, error_message/2, error_message/3, warning_message/1, warning_message/2, warning_message/3, note_message/1, note_message/2, note_message/3, simple_message/1, simple_message/2, optional_message/2, optional_message/3, debug_message/1, debug_message/2, debug_goal/2, debug_goal/3.
  - Multifiles:
    issue_debug_messages/1.

- Other modules used:
  - System library modules:
    format, lists, filenames, strings, write.

62.2 Documentation on exports (messages)

error_message/1:
Meta-predicate with arguments: error_message(addmodule).
Usage: error_message(Text)
  - Description: The text provided in Text is printed as an ERROR message.
  - The following properties should hold at call time:
    Text is a string (a list of character codes).

error_message/2:
Meta-predicate with arguments: error_message(?,addmodule).
Usage: error_message(Text, ArgList)
  - Description: The text provided in Text is printed as an ERROR message, using the arguments in ArgList to interpret any variable-related formatting commands embedded in Text.
  - The following properties should hold at call time:
    Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2.
    ArgList is a list.
error_message/3:  
Meta-predicate with arguments: error_message(?,?,addmodule).

Usage: error_message(Lc, Text, ArgList)

- Description: The text provided in Text is printed as an ERROR message, using the arguments in ArgList to interpret any variable-related formatting commands embedded in Text, and reporting error location Lc (file and line numbers).
- The following properties should hold at call time:
  Identifies a program source line. (messages:location/1)
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)
  ArgList is a list. (basic_props:list/1)

warning_message/1:  
Meta-predicate with arguments: warning_message(addmodule).

Usage: warning_message(Text)

- Description: The text provided in Text is printed as a WARNING message.
- The following properties should hold at call time:
  Text is a string (a list of character codes). (basic_props:string/1)

warning_message/2:  
Meta-predicate with arguments: warning_message(?,addmodule).

Usage: warning_message(Text, ArgList)

- Description: The text provided in Text is printed as a WARNING message, using the arguments in ArgList to interpret any variable-related formatting commands embedded in Text.
- The following properties should hold at call time:
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)
  ArgList is a list. (basic_props:list/1)

warning_message/3:  
Meta-predicate with arguments: warning_message(?,?,addmodule).

Usage: warning_message(Lc, Text, ArgList)

- Description: The text provided in Text is printed as a WARNING message, using the arguments in ArgList to interpret any variable-related formatting commands embedded in Text, and reporting error location Lc (file and line numbers).
- The following properties should hold at call time:
  Identifies a program source line. (messages:location/1)
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)
  ArgList is a list. (basic_props:list/1)
**note_message/1:**  
*Meta-predicate* with arguments: `note_message(addmodule)`.  
Usage: `note_message(Text)`  
- *Description:* The text provided in `Text` is printed as a NOTE.  
- *The following properties should hold at call time:*  
  Text is a string (a list of character codes).  

**note_message/2:**  
*Meta-predicate* with arguments: `note_message(?, addmodule)`.  
Usage: `note_message(Text, ArgList)`  
- *Description:* The text provided in `Text` is printed as a NOTE, using the arguments in `ArgList` to interpret any variable-related formatting commands embedded in `Text`.  
- *The following properties should hold at call time:*  
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`.  
  ArgList is a list.  

**note_message/3:**  
*Meta-predicate* with arguments: `note_message(?, ?, addmodule)`.  
Usage: `note_message(Lc, Text, ArgList)`  
- *Description:* The text provided in `Text` is printed as a NOTE, using the arguments in `ArgList` to interpret any variable-related formatting commands embedded in `Text`, and reporting error location `Lc` (file and line numbers).  
- *The following properties should hold at call time:*  
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`.  
  ArgList is a list.  

**simple_message/1:**  
Usage: `simple_message(Text)`  
- *Description:* The text provided in `Text` is printed.  
- *The following properties should hold at call time:*  
  Text is a string (a list of character codes).  

**simple_message/2:**  
Usage: `simple_message(Text, ArgList)`  
- *Description:* The text provided in `Text` is printed as a message, using the arguments in `ArgList`.  
- *The following properties should hold at call time:*  
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`.  
  ArgList is a list.
optional_message/2:
Usage: optional_message(Text, Opts)
Description: The text provided in Text is printed as a message, but only if the atom -v is a member of Opts. These predicates are meant to be used for optional messages, which are only to be printed when verbose output is requested explicitly.
The following properties should hold at call time:
Text is a string (a list of character codes). (basic_props:string/1)
Opts is a list of atms. (basic_props:list/2)

optional_message/3:
Usage: optional_message(Text, ArgList, Opts)
Description: The text provided in Text is printed as a message, using the arguments in ArgList, but only if the atom -v is a member of Opts. These predicates are meant to be used for optional messages, which are only to be printed when verbose output is requested explicitly.
The following properties should hold at call time:
Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)
ArgList is a list. (basic_props:list/1)
Opts is a list of atms. (basic_props:list/2)

debug_message/1:
Meta-predicate with arguments: debug_message(addmodule).
Usage: debug_message(Text)
Description: The text provided in Text is printed as a debugging message. These messages are turned on by defining a fact of issue_debug_messages/1 with the module name as argument.
The following properties should hold at call time:
Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)

debug_message/2:
Meta-predicate with arguments: debug_message(?,addmodule).
Usage: debug_message(Text, ArgList)
Description: The text provided in Text is printed as a debugging message, using the arguments in ArgList to interpret any variable-related formatting commands embedded in Text. These messages are turned on by defining a fact of issue_debug_messages/1 which the module name as argument.
The following properties should hold at call time:
Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)
ArgList is a list. (basic_props:list/1)
debug_goal/2:
Meta-predicate with arguments: debug_goal(goal,addmodule).
Usage: debug_goal(Goal, Text)
   – Description: Goal is called. The text provided in Text is then printed as a debugging message. The whole process (including running Goal) is turned on by defining a fact of issue_debug_messages/1 with the module name as argument.

debug_goal/3:
Meta-predicate with arguments: debug_goal(goal,?,?,addmodule).
Usage: debug_goal(Goal, Text, ArgList)
   – Description: Goal is called. The text provided in Text is then printed as a debugging message, using the arguments in ArgList to interpret any variable-related formatting commands embedded in Text. Note that the variables in ArgList can be computed by Goal. The whole process (including running Goal) is turned on by defining a fact of issue_debug_messages/1 with the module name as argument.

62.3 Documentation on multifiles (messages)

issue_debug_messages/1:
The predicate is multile.
The predicate is of type data.
Usage: issue_debug_messages(Module)
   – Description: Printing of debugging messages is enabled for module Module.
   – The following properties should hold upon exit:
     Module is currently instantiated to an atom. (term_typing:atom/1)

62.4 Documentation on internals (messages)

location/1:
Usage:
   – Description: Identifies a program source line.

62.5 Known bugs and planned improvements (messages)
- Debug message switching should really be done with an expansion, for performance.
63 Accessing and redirecting the stream aliases

Author(s): Manuel Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#52 (2001/1/26, 15:34:13 CET)

This library allows the redefinition of the files to which the special streams user_input, user_output, and user_error point to. On startup they point to the standard input, standard output, and standard error, in Unix style (Windows users may find that standard error stream does not work properly). Changing the file pointed to is useful for, e.g., redirecting the place to which the Prolog’s standard error stream goes from within Prolog (e.g., to start a log file).

63.1 Usage and interface (io_alias_redirection)

- Library usage:
  := use_module(library(io_alias_redirection)).
- Exports:
  - Predicates:
    set_stream/3, get_stream/2.

63.2 Documentation on exports (io_alias_redirection)

**set_stream/3:**
Usage: set_stream(StreamAlias, NewStream, OldStream)
- Description: Associate StreamAlias with an open stream NewStream. Returns in OldStream the stream previously associated with the alias. The mode of NewStream must match the intended use of StreamAlias.
- The following properties should hold at call time:
  StreamAlias is currently a term which is not a free variable. (term_typing:nonvar/1)
  NewStream is currently a term which is not a free variable. (term_typing:nonvar/1)
  StreamAlias is the alias of an open stream, i.e., an atom which represents a stream at Prolog level. (streams_basic:stream_alias/1)
  NewStream is an open stream. (streams_basic:stream/1)
  OldStream is an open stream. (streams_basic:stream/1)

**get_stream/2:**
Usage: get_stream(StreamAlias, Stream)
- Description: Return in Stream the stream associated with StreamAlias.
- The following properties should hold at call time:
  StreamAlias is currently a term which is not a free variable. (term_typing:nonvar/1)
  StreamAlias is the alias of an open stream, i.e., an atom which represents a stream at Prolog level. (streams_basic:stream_alias/1)
  Stream is an open stream. (streams_basic:stream/1)
Chapter 64: Atom to term conversion

Author(s): Francisco Bueno, Daniel Cabeza.

This module implements the predicates involved in the atom to term conversion.

64.1 Usage and interface (atom2term)

- **Library usage:**
  ```prolog```
  ```
  :- use_module(library(atom2term)).
  ```

- **Exports:**
  - **Predicates:**
    - atom2term/2,
    - string2term/2,
    - parse_term/3.

64.2 Documentation on exports (atom2term)

**atom2term/2:**

- **Usage:**
  ```prolog```
  ```
  atom2term(Atom, Term).
  ```

- **Description:** Convert an atom into a term. Atom is an atom, but must have term syntax. Term is a term resulting from parsing Atom char by char.

- **The following properties should hold at call time:**
  - Atom is currently a term which is not a free variable.  
  - Term is a free variable.

**string2term/2:**

- **Usage:**
  ```prolog```
  ```
  string2term(String, Term).
  ```

- **Description:** Same as atom2term/2 but first argument is a string (containing a term).

- **The following properties should hold at call time:**
  - String is currently a term which is not a free variable.
  - Term is a free variable.

**parse_term/3:**

- **Usage:**
  ```prolog```
  ```
  parse_term(String, Term, Dummy).
  ```

- **Description:** String is parsed into Term upto Dummy (which is the non-parsed rest of the list).

- **The following properties should hold at call time:**
  - String is currently a term which is not a free variable.
  - Term is a free variable.

64.3 Known bugs and planned improvements (atom2term)

- This is just a quick hack written mainly for parsing daVinci’s messages. There should be a call to the standard reader to do this!
65 ctrlcclean (library)

Version: 0.4#5 (1998/2/24)

65.1 Usage and interface (ctrlcclean)

- Library usage:
  ```prolog```
  :- use_module(library(ctrlcclean)).
  ```prolog```

- Exports:
  - Predicates:
    ctrlc_clean/1, delete_on_ctrlc/2, ctrlcclean/0.

- Other modules used:
  - System library modules:
    system.

65.2 Documentation on exports (ctrlcclean)

ctrlc_clean/1:
No further documentation available for this predicate.

Meta-predicate with arguments: ctrlc_clean(goal).

delete_on_ctrlc/2:
No further documentation available for this predicate.

ctrlcclean/0:
No further documentation available for this predicate.
Chapter 66: errhandle (library)


66 errhandle (library)

66.1 Usage and interface (errhandle)

- Library usage:
  :- use_module(library(errhandle)).
- Exports:
  - Predicates:
    error_protect/1, handle_error/2.

66.2 Documentation on exports (errhandle)

error_protect/1:
  No further documentation available for this predicate.
  Meta-predicate with arguments: error_protect(goal).

handle_error/2:
  No further documentation available for this predicate.
67 Fast reading and writing of terms

Author(s): Daniel Cabeza, Oscar Portela Arjona.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#16 (2000/8/29, 13:44:18 CEST)

This library provides predicates to support reading / writing of terms on a format designed to be handled on read faster than standard representation.

67.1 Usage and interface (fastrw)

- Library usage:

```prolog
:- use_module(library(fastrw)).
```

- Exports:

  - Predicates:
    fast_read/1, fast_write/1, fast_read/2, fast_write/2,
    fast_write_to_string/3.

- Other modules used:

  - System library modules:
    dict.

67.2 Documentation on exports (fastrw)

**fast_read/1:**

```prolog
fast_read(Term)
```

The next term is read from current standard input and is unified with `Term`. The syntax of the term must agree with fast_read / fast_write format. If the end of the input has been reached, `Term` is unified with the term 'end_of_file'. Further calls to `fast_read/1` will then cause an error.

**fast_write/1:**

```prolog
fast_write(Term)
```

Output `Term` in a way that `fast_read/1` and `fast_read/2` will be able to read it back.

**fast_read/2:**

```prolog
fast_read(Stream, Term)
```

The next term is read from `Stream` and unified with `Term`. The syntax of the term must agree with fast_read / fast_write format. If the end of the input has been reached, `Term` is unified with the term 'end_of_file'. Further calls to `fast_read/2` will then cause an error.

**fast_write/2:**

```prolog
fast_write(Stream, Term)
```

Output `Term` to `Stream` in a way that `fast_read/1` and `fast_read/2` will be able to read it back.
fast_write_to_string/3:
No further documentation available for this predicate.

67.3 Known bugs and planned improvements (fastrw)

- Both fast_read/2 and fast_write/2 simply set the current output/input and call fast_read/1 and fast_write/1. Therefore, in the event an error happens during its execution, the current input / output streams may be left pointing to the Stream
68 File name manipulation

Author(s): Daniel Cabeza, Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.3#51 (1999/9/9, 16:28:44 MEST)
This library provides some small utilities to handle file name syntax.

68.1 Usage and interface (filenames)

- Library usage:
  :- use_module(library(filenames)).
- Exports:
  - Predicates:
    - no_path_file_name/2, file_name_extension/3, basename/2, extension/2.
- Other modules used:
  - System library modules:
    lists.

68.2 Documentation on exports (filenames)

no_path_file_name/2:
This predicate will extract the last item (usually the file name) from a given path.
The first argument must be instantiated to a string or atom. Whenever the first argument
is an atom, the second argument will be an atom. Whenever the first argument is a string,
the second argument will be a string.
This predicate will fail under any of the following conditions:
  - First argument is not an atom, nor a string.
  - Second argument is not the last given path item (given path is the first argument).
Those are the most usual usages of no_path_file_name/2:
?– no_path_file_name("/home/nexusV/somefile.txt",K).

K = "somefile.txt" ?
yes

?– no_path_file_name('/home/nexusV/somefile.txt',K).

K = 'somefile.txt' ?
yes

Usage: no_path_file_name(Path, FileName)
- Description: FileName is the file corresponding to the given Path.
- Call and exit should be compatible with:
  Path is an atom or a string
  FileName is an atom or a string
**file_name_extension/3:**

This predicate may be used in two ways:

- To create a file name from its components: name and extension. For instance:
  ```prolog
  ?- file_name_extension(File, mywork, '.txt').
  File = 'mywork.txt' ?
  yes
  ?-
  ```

- To split a file name into its name and extension. For instance:
  ```prolog
  ?- file_name_extension('mywork.txt', A, B).
  A = mywork,
  B = '.txt' ?
  yes
  ?-
  ```

Any other usage of `file_name_extension/3` will cause the predicate to fail. Notice that valid arguments are accepted both as atoms or strings.

**Usage:** `file_name_extension(File, BaseName, Extension)`

  - **Description:** Splits a `FileName` into its `BaseName` and `Extension`.
  - **Call and exit should be compatible with:**
    - `FileName` is an atom or a string (`filenames:atom_or_str/1`)
    - `BaseName` is an atom or a string (`filenames:atom_or_str/1`)
    - `Extension` is an atom or a string (`filenames:atom_or_str/1`)

**basename/2:**

`basename(FileName, BaseName)`

`BaseName` is `FileName` without extension. Equivalent to `file_name_extension(FileName, BaseName, '_')`. Useful to extract the base name of a file using functional syntax.

**extension/2:**

`extension(FileName, Extension)`

`Extension` is the extension (suffix) of `FileName`. Equivalent to `file_name_extension(FileName, '_', Extension)`. Useful to extract the extension of a file using functional syntax.
69 Symbolic filenames

Author(s): Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#9 (2002/5/28, 19:11:29 CEST)

This module provides a predicate for file opening which can use any term as an alias for the filename (i.e., symbolic filenames) instead of the usual constants which are file system path names of the actual files.

The correspondence between an alias and the actual file path is done dynamically, without having to recompile the program. It is possible to define the correspondence via facts for file_alias/2 in a file declared with multifile:alias_file/1 in the program: those facts will be dynamically loaded when running the program. Alternatively, the correspondence can be defined via shell environment variables, by defining the value of a variable by the (symbolic) name of the file to be the path of the actual file.

69.1 Usage and interface (symfnames)

- Library usage:
  
  \[- use_module(library(symfnames)). \]

- Exports:
  - Predicates:
    - open/3.
  - Multi-files:
    - alias_file/1, file_alias/2.
- Other modules used:
  - System library modules:
    - read, system.

69.2 Documentation on exports (symfnames)

**open/3:**

open(File, Mode, Stream)

Open File with mode Mode and return in Stream the stream associated with the file. It is like streams_basic:open/3, but File is considered a symbolic name: either defined by user:file_alias/2 or as an environment variable. Predicate user:file_alias/2 is inspected before the environment variables.

**Usage:**

- Calls should, and exit will be compatible with:
  
  Stream is an open stream. (streams_basic:stream/1)

- The following properties should hold at call time:
  
  File is any term. (basic_props:term/1)
  Mode is an opening mode (‘read’, ‘write’ or ‘append’). (streams_basic:io_mode/1)

- The following properties hold upon exit:
  
  Stream is an open stream. (streams_basic:stream/1)
69.3 Documentation on multfiles (symfnames)

**alias_file/1:**

```prolog
alias_file(File)
```

*Declarations*

Declares `File` to be a file defining symbolic names via `file_alias/2`. Anything else in `File` is simply ignored.

The predicate is *multifile*.

**file_alias/2:**

```prolog
file_alias(Alias, File)
```

*Declarations*

Declares `Alias` as a symbolic name for `File`, the real name of an actual file (or directory).

The predicate is *multifile*.

The predicate is of type *data*.

69.4 Other information (symfnames)

The example discussed here is included in the distribution files. There is a main application file which uses module `mm`. This module reads a line from a file; the main predicate in the main file then prints this line. The important thing is that the file read is named by a symbolic name "file". The main application file declares another file where the symbolic names are assigned actual file names:

```prolog
:- use_module(mm).

:- multifile alias_file/1.
alias_file(myfiles).

main :- p(X), display(X), nl.
```

Now, the file `myfiles.pl` can be used to change the file you want to read from without having to recompile the application. The current assignment is:

```prolog
%:- use_package([]).
file_alias(file,'mm.pl').
```

so the execution of the application will show the first line of `mm.pl`. However, you can change to:

```prolog
file_alias(file,'main.pl').
```

and then execution of the same executable will show the first line of `main.pl`. 
70 File I/O utilities

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#93 (2003/12/21, 2:30:33 CET)
This module implements the file I/O utilities.

70.1 Usage and interface (file_utils)

- Library usage:
  :- use_module(library(file_utils)).
- Exports:
  - Predicates:
    file_terms/2, copy_stdout/1, file_to_string/2, stream_to_string/2.
- Other modules used:
  - System library modules:
    read, streams.

70.2 Documentation on exports (file_utils)

file_terms/2: PREDICATE

Usage 1: file_terms(File, Terms)
  - Description: Transform a file File to/from a list of terms Terms.
  - The following properties should hold upon exit:
    File is a source name. (streams_basic:sourcename/1)
    Terms is a list. (basic_props:list/1)
  - The following properties should hold globally:
    File is not further instantiated. (basic_props:not_further_inst/2)

Usage 2: file_terms(File, Terms)
  - Description: Unifies Terms with the list of all terms in File.
  - The following properties should hold at call time:
    File is a source name. (streams_basic:sourcename/1)
    Terms is a free variable. (term_typing:var/1)
  - The following properties should hold upon exit:
    File is a source name. (streams_basic:sourcename/1)
    Terms is a list. (basic_props:list/1)

Usage 3: file_terms(File, Terms)
  - Description: Writes the terms in list Terms (including the ending '.') onto file File.
  - The following properties should hold at call time:
    File is a source name. (streams_basic:sourcename/1)
    Terms is a list. (basic_props:list/1)
The following properties should hold upon exit:
File is a source name. (streams_basic:sourcename/1)
Terms is a list. (basic_props:list/1)

**copy_stdout/1:**

Usage: copy_stdout(File)

- Description: Copies file File to standard output.
- The following properties should hold at call time:
  File is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties should hold upon exit:
  File is a source name. (streams_basic:sourcename/1)

**file_to_string/2:**

Usage: file_to_string(FileName, String)

- Description: Reads all the characters from the file FileName and returns them in String.
- Call and exit should be compatible with:
  FileName is a source name. (streams_basic:sourcename/1)
  String is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold at call time:
  FileName is currently a term which is not a free variable. (term_typing:nonvar/1)
  String is a free variable. (term_typing:var/1)

**stream_to_string/2:**

Usage: stream_to_string(Stream, String)

- Description: Reads all the characters from Stream and returns them in String.
- Call and exit should be compatible with:
  Stream is an open stream. (streams_basic:stream/1)
  String is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold at call time:
  Stream is currently a term which is not a free variable. (term_typing:nonvar/1)
  String is a free variable. (term_typing:var/1)
71 File locks

Author(s): J. Gomez, D. Cabeza, M. Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#159 (2001/11/27, 11:58:24 CET)

This module implements file locks: the ability to lock a file so that other processes cannot access it until the file is unlocked. It is, however, not working. The predicates do nothing. Proper implementation is planned for a near future.

71.1 Usage and interface (file_locks)

- Library usage:
  
  ```prolog
  :- use_module(library(file_locks)).
  ```

- Exports:
  - Predicates:
    - lock_file/3, unlock_file/2.

71.2 Documentation on exports (file_locks)

**lock_file/3:**

**Predicate**

**Usage:** lock_file(File, LockType, Result)

- **Description:** Tries to lock File with LockType and returns the result (either true or false) in Result.

- **The following properties should hold at call time:**
  - File is an atom.
  - LockType is an atom.
  - Result is an atom.

  ```prolog
  (basic_props:atm/1)
  ```

**unlock_file/2:**

**Predicate**

**Usage:** unlock_file(File, Result)

- **Description:** Tries to unlock File the result (either true or false) in Result.

- **The following properties should hold at call time:**
  - File is an atom.
  - Result is an atom.

  ```prolog
  (basic_props:atm/1)
  ```

71.3 Known bugs and planned improvements (file_locks)

- No doing anything helpful.
Chapter 72: Term manipulation utilities

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#94 (2003/12/21, 18:44:37 CET)
This module implements some utils to do term manipulation.

72.1 Usage and interface (terms)

- Library usage:
  ```prolog```
  :- use_module(library(terms)).
  ```prolog```
- Exports:
  - Predicates:
    - copy_args/3, arg/2, atom_concat/2.

72.2 Documentation on exports (terms)

copy_args/3: PREDICATE
Usage: copy_args(N, Term, Copy)
  - Description: Term and Copy have the same first N arguments.
  - The following properties should hold at call time:
    N is a non-negative integer. (basic_props:nnegint/1)

arg/2: PREDICATE
Usage: arg(Term, Arg)
  - Description: Arg is an argument of Term. Gives each of the arguments on backtracking.

atom_concat/2: PREDICATE
atom_concat(Atoms, Atm)
Atm is the atom resulting from concatenating all atoms in the list Atoms in the order in which they appear.
73 Term checking utilities

Author(s): The CLIP Group.
Version: 1.11#95 (2003/12/21, 18:52:2 CET)

This module implements the term checking utilities.

73.1 Usage and interface (terms_check)

- Library usage:
  
  ```prolog```
  ```
  :- use_module(library(terms_check)).
  ```
  ```
  ```

- Exports:
  
  - Predicates:
    `ask/2`, `variant/2`, `most_general_instance/3`, `most_specific_generalization/3`.
  
- Properties:
  
  `instance/2`.

73.2 Documentation on exports (terms_check)

ask/2:  

```prolog```

```prolog```

ask(Term1, Term2)  

Term1 and Term2 unify without producing bindings for the variables of Term1. I.e., 

`instance(Term1,Term2)` holds.


instance/2:  

```prolog```

```prolog```

instance(Term1, Term2)  

Term1 is an instance of Term2.  

Usage: instance(A, B)  

- **The following properties hold globally:**  
  
  This predicate is understood natively by CiaoPP.  

  `basic_props:native/1`  


variant/2:  

```prolog```

```prolog```

variant(Term1, Term2)  

Term1 and Term2 are identical up to renaming.


most_general_instance/3:  

```prolog```

```prolog```

most_general_instance(Term1, Term2, Term)  

Term satisfies `instance(Term,Term1)` and `instance(Term,Term2)` and there is no term 
more general than Term (modulo variants) that satisfies it.
most_specific_generalization/3:

\[
\text{most_specific_generalization}(\text{Term}_1, \text{Term}_2, \text{Term})
\]

\(\text{Term}\) satisfies \text{instance}(\text{Term}_1,\text{Term}) and \text{instance}(\text{Term}_2,\text{Term}) and there is no term less general than \(\text{Term}\) (modulo variants) that satisfies it.

### 73.3 Other information (terms_check)

Currently, \text{ask}/2 and \text{instance}/2 are exactly the same. However, \text{ask}/2 is more general, since it is also applicable to constraint domains (although not yet implemented): for the particular case of Herbrand terms, it is just \text{instance}/2 (which is the only ask check currently implemented).
74 Sets of variables in terms

Author(s): The CLIP Group.

Version: 1.11#96 (2003/12/21, 18:58:37 CET)

This module implements predicates to handle sets of variables in terms.

74.1 Usage and interface (terms_vars)

- Library usage:
  
  :- use_module(library(terms_vars)).

- Exports:
  - Predicates:
    varset/2, varsbag/3, varset_in_args/2.

- Other modules used:
  - System library modules:
    idlists, sort.

74.2 Documentation on exports (terms_vars)

varset/2:  

varset(Term, Xs)

Xs is the sorted list of all the variables in Term.

varsbag/3:

varsbag(Term, Vs, Xs)

Vs is the list of all the variables in Term ordered as they appear in Term right-to-left depth-first (including duplicates) plus Xs.

varset_in_args/2:  

Usage: varset_in_args(T, LL)

- Description: Each list of LL contains the variables of an argument of T, for each argument, and in left to right order.
- The following properties should hold at call time:
  T is currently a term which is not a free variable.  (term_typing:nonvar/1)
- The following properties should hold upon exit:
  LL is a list of list(var)s.  (basic.props:list/2)
75 A simple pretty-printer for Ciao programs

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#126 (2003/12/30, 21:52:16 CET)
This library module writes out to standard output a clause or a list of clauses.

75.1 Usage and interface (pretty_print)

- Library usage:
  ```prolog
  :- use_module(library(pretty_print)).
  ```
- Exports:
  - Predicates:
    - pretty_print/2, pretty_print/3.
- Other modules used:
  - System library modules:
    - operators, vndict, write.

75.2 Documentation on exports (pretty_print)

**pretty_print/2:**

**Usage:** pretty_print(Cls, Flags)

- **Description:** Prints each clause in the list Cls after numbering its variables.
- **The following properties should hold at call time:**
  ```prolog
  pretty_print:clauses(Cls) (pretty_print:clauses/1)
  Flags is a list of flags. (basic_props:list/2)
  ```

**pretty_print/3:**

**Usage:** pretty_print(Cls, Flags, Ds)

- **Description:** Prints each clause in the list Cls after using the corresponding variable names dictionary in Ds to name its variables.
- **The following properties should hold at call time:**
  ```prolog
  pretty_print:clauses(Cls) (pretty_print:clauses/1)
  Flags is a list of flags. (basic_props:list/2)
  Ds is a dictionary of variable names. (vndict:varnamedict/1)
  ```
75.3 Documentation on internals (pretty_print)

**clauses/1:**
A regular type, defined as follows:

```prolog
clauses([]).
clauses([_1|_2]) :-
    clause(_1),
    clauses(_2).
clauses(_1) :-
    clause(_1).
```

**clause/1:**
A regular type, defined as follows:

```prolog
clause(_1) :-
    clterm(_1).
clause((_1,_2)) :-
    clterm(_1),
    term(_2).
```

**clterm/1:**
A regular type, defined as follows:

```prolog
clterm(clause(_1,_2)) :-
    callable(_1),
    body(_2).
clterm(directive(_1)) :-
    body(_1).
clterm((_1:-_2)) :-
    callable(_1),
    body(_2).
clterm(_1) :-
    callable(_1).
```

**body/1:**
A well formed body, including ege expressions and &-concurrent expressions. The atomic goals may or may not have a key in the form `^(goal:any)`, and may or may not be module qualified, but if they are it has to be in the form `^(moddesc:goal):any`.

**Usage:** body(X)
- **Description:** X is a printable body.
flag/1:
A keyword ask/1 flags whether to output asks or whens and nl/1 whether to separate clauses with a blank line or not.

Usage: flag(X)
  – Description: X is a flag for the pretty-printer.
Chapter 76: Pretty-printing assertions

76 Pretty-printing assertions

Author(s): Francisco Bueno Carrillo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

This module defines some predicates which are useful for writing assertions in a readable form.

76.1 Usage and interface (assert_write)

- Library usage:
  :- use_module(library(assert_write)).
- Exports:
  - Predicates:
    write assertion/6, write assertion as comment/6.
- Other modules used:
  - System library modules:
    format, assertions/assrt_lib, messages, assertions/assertions_props.

76.2 Documentation on exports (assert_write)

write assertion/6: PREDICATE

Usage: write assertion(Goal, Status, Type, Body, Dict, Flag)
– Description: Writes the (normalized) assertion to current output.
– Call and exit should be compatible with:
  Status is an acceptable status for an assertion.               (assertions_props:assrt_status/1)
  Type is an admissible kind of assertion.                 (assertions_props:assrt_type/1)
  Body is a normalized assertion body.                        (assertions_props:nabody/1)
  Dict is a dictionary of variable names.                     (assertions_props:dictionary/1)
  Flag is status or nostatus.                                 (asserts write:status_flag/1)

write assertion as comment/6: PREDICATE

Usage: write assertion as comment(Goal, Status, Type, Body, Dict, Flag)
– Description: Writes the (normalized) assertion to current output as a Prolog comment.
– Call and exit should be compatible with:
  Status is an acceptable status for an assertion.               (assertions_props:assrt_status/1)
  Type is an admissible kind of assertion.                 (assertions_props:assrt_type/1)
  Body is a normalized assertion body.                        (assertions_props:nabody/1)
  Dict is a dictionary of variable names.                     (assertions_props:dictionary/1)
  Flag is status or nostatus.                                 (asserts write:status_flag/1)
Chapter 77: The Ciao library browser

Author(s): Angel Fernandez Pineda.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.7#21 (2000/9/26, 13:37:17 CEST)

Librowser provides a set of predicates which enables the user to interactively find
Ciao/Prolog libraries and/or any predicate exported by them.

This is a simple example:

?- apropos('*find*').
persdbrt_sql: dbfindall/4
persdbrtsql: dbfindall/4
conc_aggregates: findall/3
linda: rd_findall/3
vndict: find_name/4
internals: $find_file/8
aggregates: findall/4,findall/3

yes
?- Librowser is specially useful when using inside GNU Emacs, just place the cursor over a
librowser response and press C-cTAB in order to get help on the related predicate. Refer to the
"Using Ciao inside GNU Emacs" chapter for further information.

77.1 Usage and interface (librowser)

- Library usage:
  It is not necessary to use this library at user programs. It was designed to be used at the
  Ciao toplevel shell: ciaosh. In order to do so, just make use of use_module/1 as follows:
  use_module(library(librowser)).

  Then, the library interface must be read. This is automatically done when calling any
  predicate at librowser, and the entire process will take a little moment. So, you should want
  to perform such a process after loading the Ciao toplevel:
  Ciao 0.9 #75: Fri Apr 30 19:04:24 MEST 1999
  ?- use_module(library(librowser)).

  yes
  ?- update.

  Whether you want this process to be automatically performed when loading ciaosh, you
  may include those lines in your .ciaorc personal initialization file.

- Exports:
  - Predicates:
    update/0, browse/2, where/1, describe/1, system_lib/1, apropos/1.

- Other modules used:
  - System library modules:
    filenames, read, fastrw, system, streams, patterns, lists.
77.2 Documentation on exports (librowser)

update/0:

This predicate will scan the Ciao system libraries for predicate definitions. This may be done once time before calling any other predicate at this library.
update/0 will also be automatically called (once) when calling any other predicate at librowser.
Usage:
- Description: Creates an internal database of modules at Ciao system libraries.

browse/2:

This predicate is fully reversible, and is provided to inspect concrete predicate specifications. For example:
?- browse(M, findall/A).
A = 3,
M = conc_aggregates ? ;
A = 4,
M = aggregates ? ;
A = 3,
M = aggregates ? ;
no
?- Usage: browse(Module, Spec)
- Description: Associates the given Spec predicate specification with the Module which exports it.
- The following properties should hold at call time:
  Module is a module name (an atom)  (librowser:module_name/1)
  Spec is a Functor/Arity predicate specification  (librowser:pred_spec/1)

where/1:

This predicate will print at the screen the module needed in order to import a given predicate specification. For example:
?- where(findall/A).
findall/3 exported at module conc_aggregates
findall/4 exported at module aggregates
findall/3 exported at module aggregates

yes
?- Usage: where(Spec)
- Description: Display what module to load in order to import the given Spec.
- The following properties should hold at call time:
  Spec is a Functor/Arity predicate specification  (librowser:pred_spec/1)
describe/1: PREDICATE
This one is used to find out which predicates were exported by a given module. Very useful when you know the library, but not the concrete predicate. For example:

?- describe(librowser).
Predicates at library librowser:

\[ \text{apropos/1} \]
\[ \text{system_lib/1} \]
\[ \text{describe/1} \]
\[ \text{where/1} \]
\[ \text{browse/2} \]
\[ \text{update/0} \]

yes
?- 

Usage: describe(Module)
- Description: Display a list of exported predicates at the given Module
- The following properties should hold at call time:
  Module is a module name (an atom) \((\text{librowser:module_name/1})\)

system_lib/1: PREDICATE
It retrieves on backtracking all Ciao system libraries stored in the internal database. Certainly, those which were scanned at update/0 calling.

Usage: system_lib(Module)
- Description: Module variable will be successively instantiated to the system libraries stored in the internal database.
- The following properties should hold at call time:
  Module is a module name (an atom) \((\text{librowser:module_name/1})\)

apropos/1: PREDICATE
This tool makes use of regular expresions in order to find predicate specifications. It is very useful whether you can’t remember the full name of a predicate. Regular expresions take the same format as described in library patterns. Example:

?- apropos('atom_*').

\[ \text{terms: atom_concat/2} \]
\[ \text{concurrency: atom_lock_state/2} \]
\[ \text{atomic_basic: atom_concat/3,atom_length/2,atom_codes/2} \]
\[ \text{iso_byte_char: atom_chars/2} \]

yes
?- 

Usage: apropos(RegSpec)
- Description: This will search any predicate specification Spec which matches the given RegSpec incomplete predicate specification.
- The following properties should hold at call time:
  RegSpec is a Pattern/Arity specification. \((\text{librowser:apropos_spec/1})\)
77.3 Documentation on internals (librowser)

`apropos_spec/1:`
Defined as:

```
apropos_spec(_1).
apropos_spec(Pattern/Arity) :-
    pattern(Pattern),
    int(Arity).
```

Usage: `apropos_spec(S)`

- Description: `S` is a Pattern/Arity specification.
Chapter 78: Code translation utilities

Author(s): Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#16 (2003/4/8, 4:35:48 CEST)

This library offers a general way to perform clause body expansions. Goal, fact and spec translation predicates are automatically called when needed, while this utility navigates through the meta-argument specification of the body itself. All predicates within this library must be called at second-pass expansions, since it uses information stored at c_itf library.

78.1 Usage and interface (expansion_tools)

- Library usage:
  This library is provided as a tool for those modules which performs source-to-source code translation, usually known as code expanders. It may be loaded as other modules using a use_module/1. Nothing special needs to be done.

- Exports:
  - Predicates:
    imports_meta_pred/3, body_expander/6, arg_expander/6.
  - Other modules used:
    System library modules:
    compiler/c_itf.

78.2 Documentation on exports (expansion_tools)

imports_meta_pred/3:
Macro provided in order to know meta-predicate specifications accessible from a module.
Usage: imports_meta_pred(Module, MetaSpec, AccessibleAt)
- Description: Tells whether MetaSpec meta-predicate specification is accessible from Module. AccessibleAt will be binded to '-' whether meta-predicate is a builtin one. If not, it will be unified with the module which defines the meta-predicate.
- The following properties should hold at call time:
  Module is an atom. (basic_props:atm/1)
  MetaSpec is any term. (basic_props:term/1)
  AccessibleAt is a free variable. (term_typing:var/1)

body_expander/6:
This predicate is the main translation tool. It navigates through a clause body, when a single goal appears, user-code is called in order to perform a translation. Whether user-code fails to translate the involved goal, it remains the same. Regardless that goal is translated or not, an argument expansion will be performed over all goals if applicable (see arg_expander/6 predicate).
Variable (unknown at compile time) goals will also be attempt to translate.
Meta-predicate with arguments: body_expander(pred(3),pred(3),pred(3),?,?,?).

Usage: body_expander(GoalTrans, FactTrans, SpecTrans, Module, Body, ExpandedBody)

- Description: Translates Body to ExpandedBody by the usage of user-defined translators GoalTrans, FactTrans and SpecTrans. The module where the original body appears must be unified with Module argument.

- The following properties should hold at call time:
  GoalTrans is a user-defined predicate which performs goal meta-type translation (expansion_tools:goal_expander/1)
  FactTrans is a user-defined predicate which performs fact meta-type translation (expansion_tools:fact_expander/1)
  SpecTrans is a user-defined predicate which performs spec meta-type translation (expansion_tools:spec_expander/1)

Module is an atom.  
Body is currently a term which is not a free variable.  
ExpandedBody is a free variable.

arg_expander/6:

This predicate is an auxiliary translation tool, which is used by body_expander/6 predicate. It remains exported as a macro. The predicate navigates through the meta-argument specification of a goal. Whether a goal, fact or spec argument appears, user-code is called in order to perform a translation. Whether user-code fails to translate the involved argument, it remains the same. Builtins as ';/2 or ';/2 are treated as meta-predicates defining goal meta-arguments. When a goal meta-argument is located, body_expander/6 will be called in order to navigate through it. Notice that a goal meta-argument may be unified with another goal defining another meta-argument, so navigation is required. If arguments are not known to arg_expander/6, translation will not occur. This is possible whether goal or qualifying module are variables.

Meta-predicate with arguments: arg_expander(pred(3),pred(3),pred(3),?,?,?).

Usage: arg_expander(GoalTrans, FactTrans, SpecTrans, Module, Goal, ExpandedGoal)

- Description: Translates Goal to ExpandedGoal by applying user-defined translators (GoalTrans, FactTrans and SpecTrans) to each meta-argument present at such goal. The module where the original goal appears must be unified with Module argument.

- The following properties should hold at call time:
  GoalTrans is a user-defined predicate which performs goal meta-type translation (expansion_tools:goal_expander/1)
  FactTrans is a user-defined predicate which performs fact meta-type translation (expansion_tools:fact_expander/1)
  SpecTrans is a user-defined predicate which performs spec meta-type translation (expansion_tools:spec_expander/1)

Module is an atom.  
Goal is currently a term which is not a free variable.  
ExpandedBody is a free variable.
78.3 Documentation on internals (expansion_tools)

expander_pred/1:
Usage: expander_pred(Pred)
  
  Description: Pred is a user-defined predicate used to perform code translations. First argument will be binded to the corresponding term to be translated. Second argument must be binded to the corresponding translation. Third argument will be binded to the current module were first argument appears. Additional arguments will be user-defined.

78.4 Known bugs and planned improvements (expansion_tools)

- pred(N) meta-arguments are not supported at this moment.
79 Low-level concurrency/multithreading primitives

Author(s): Manuel Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#138 (2001/11/8, 19:50:32 CET)

This module provides basic mechanisms for using concurrency and implementing multi-goal applications. It provides a means for arbitrary goals to be specified to be run in a separate stack set; in that case, they are assigned a goal identifier with which further accesses (e.g., asking for more solutions) to the goal can be made. Additionally, in some architectures, these goals can be assigned an O.S. thread, separate from the one which made the initial call, thus providing concurrency and, in multiprocessors, parallelism capabilities.

As for now, the memory space of the threads (c.f., stack sets) is separate in the sense that goals are copied to the new stack set, and bindings of variables are not seen among stack sets which allows forward and backward execution to proceed independently in each stack set, at the cost of the initial goal copy. However, the program space (including, specially, the concurrent predicates) are shared and seen by all the goals and threads, and should be used as the primary means of communication and synchronization. Higher level libraries can be built using these basic blocks.

Additionally, a small set of lock primitives are provided. Locks are associated with atom names. Whereas the concurrent database facilities are enough to implement locks, semaphores, messages, etc., the predicates implementing atom-based locks are faster than the ones accessing the concurrent database (but they are less powerful).

79.1 Usage and interface (concurrency)

- **Library usage:**
  ```prolog```
  ```:- use_module(library(concurrency)).```  
- **Exports:**
  - **Predicates:**
    - `eng_call/4`, `eng_call/3`, `eng_backtrack/2`, `eng_cut/1`, `eng_release/1`, `eng_wait/1`, `eng_kill/1`, `eng_killothers/0`, `eng_self/1`, `goal_id/1`, `eng_goal_id/1`, `eng_status/0`, `lock_atom/1`, `unlock_atom/1`, `atom_lock_state/2`, `concurrent/1`.
- **Other modules used:**
  - **System library modules:**
    - `prolog_sys`.

79.2 Documentation on exports (concurrency)

`eng_call/4`:  
**Meta-predicate** with arguments: `eng_call(goal,?,?,?)`.  
**Usage:** `eng_call(Goal, EngineCreation, ThreadCreation, GoalId)`  
- **Description:** Calls Goal in a new engine (stack set), possibly using a new thread, and returns a GoalId to designate this new goal henceforth. EngineCreation can be either `wait` or `create`; the distinction is not yet meaningful. ThreadCreation can be one of `self`, `wait`, or `create`. In the first case the creating thread is used...
to execute Goal, and thus it has to wait until its first result or failure. The call will fail if Goal fails, and succeed otherwise. However, the call will always succeed when a remote thread is started. The space and identifiers reclaimed for the thread must be explicitly deallocated by calling eng_release/1. GoalIds are unique in each execution of a Ciao Prolog program.

The following properties should hold at call time:

- Goal is currently a term which is not a free variable. (term_typing:nonvar/1)
- EngineCreation is currently a term which is not a free variable. (term_typing:nonvar/1)
- ThreadCreation is currently a term which is not a free variable. (term_typing:nonvar/1)
- GoalId is a free variable. (term_typing:var/1)
- Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- EngineCreation is an atom. (basic_props:atm/1)
- ThreadCreation is an atom. (basic_props:atm/1)
- GoalId is an integer. (basic_props:int/1)

**PREDICATE**

**eng_call/3:**

Meta-predicate with arguments: eng_call(goal, ?, ?).

Usage: eng_call(Goal, EngineCreation, ThreadCreation)

- Description: Similar to eng_call/4, but the thread (if created) and stack areas are automatically released upon success or failure of the goal. No GoalId is provided for further interaction with the goal.

The following properties should hold at call time:

- Goal is currently a term which is not a free variable. (term_typing:nonvar/1)
- EngineCreation is currently a term which is not a free variable. (term_typing:nonvar/1)
- ThreadCreation is currently a term which is not a free variable. (term_typing:nonvar/1)
- Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- EngineCreation is an atom. (basic_props:atm/1)
- ThreadCreation is an atom. (basic_props:atm/1)

**PREDICATE**

**eng_backtrack/2:**

Usage: eng_backtrack(GoalId, ThreadCreation)

- Description: Performs backtracking on the goal designed by GoalId. A new thread can be used to perform backtracking, according to ThreadCreation (same as in eng_call/4). Fails if the goal is backtracked over by the local thread, and there are no more solutions. Always succeeds if executed by a remote thread. The engine is not automatically released upon failure: eng_release/1 must be called to that end.

The following properties should hold at call time:

- GoalId is currently a term which is not a free variable. (term_typing:nonvar/1)
- ThreadCreation is currently a term which is not a free variable. (term_typing:nonvar/1)
**eng_cut/1:**

Usage: `eng_cut(GoalId)`

- **Description:** Performs a *cut* in the execution of the goal `GoalId`. The next call to `eng_backtrack/2` will therefore backtrack all the way and fail.
- **The following properties should hold at call time:**
  - `GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `GoalId` is an integer. (basic_props:int/1)

**eng_release/1:**

Usage: `eng_release(GoalId)`

- **Description:** Cleans up and releases the engine executing the goal designed by `GoalId`. The engine must be idle, i.e., currently not executing any goal. `eng_wait/1` can be used to ensure this.
- **The following properties should hold at call time:**
  - `GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `GoalId` is an integer. (basic_props:int/1)

**eng_wait/1:**

Usage: `eng_wait(GoalId)`

- **Description:** Waits for the engine executing the goal denoted by `GoalId` to finish the computation (i.e., it has finished searching for a solution, either with success or failure).
- **The following properties should hold at call time:**
  - `GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `GoalId` is an integer. (basic_props:int/1)

**eng_kill/1:**

Usage: `eng_kill(GoalId)`

- **Description:** Kills the thread executing `GoalId` (if any), and frees the memory used up by the stack set. Usually one should wait (`eng_wait/1`) for a goal, and then release it, but killing the thread explicitly allows recovering from error states. A goal cannot kill itself. This feature should be used with caution, because there are situations where killing a thread might render the system in an unstable state. Threads should cooperate in their killing, but if the killed thread is blocked in a I/O operation, or inside an internal critical region, this cooperation is not possible and the system, although stopped, might very well end up in an inconsistent state.
- **The following properties should hold at call time:**
  - `GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `GoalId` is an integer. (basic_props:int/1)
eng_killothers/0:

Usage:  

- Description: Kills threads and releases stack sets of all active goals, but the one calling eng_killothers. Again, a safety measure. The same cautions as with eng_kill/1 should be taken.

eng_self/1:

Usage: eng_self(GoalId)

- Description: GoalId is unified with the identifier of the goal within which eng_self/1 is executed. eng_self/1 is deprecated, and eng_goal_id/1 should be used instead.
- The following properties should hold at call time:
  GoalId is an integer.  

goal_id/1:

Usage: goal_id(GoalId)

- Description: GoalId is unified with the identifier of the goal within which goal_id/1 is executed. goal_id/1 is deprecated, and eng_goal_id/1 should be used instead.
- The following properties should hold at call time:
  GoalId is an integer.  

eng_goal_id/1:

Usage: eng_goal_id(GoalId)

- Description: GoalId is unified with the identifier of the goal within which eng_goal_id/1 is executed.
- The following properties should hold at call time:
  GoalId is an integer.  

eng_status/0:

Usage:  

- Description: Prints to standard output the current status of the stack sets.

lock_atom/1:

Usage: lock_atom(Atom)

- Description: The semaphore associated to Atom is accessed; if its value is nonzero, it is atomically decremented and the execution of this thread proceeds. Otherwise, the goal waits until a nonzero value is reached. The semaphore is then atomically decremented and the execution of this thread proceeds.
- The following properties should hold at call time:
  Atom is currently a term which is not a free variable.  
  Atom is an atom.  

**unlock_atom/1:**

Usage: `unlock_atom(Atom)`

- **Description:** The semaphore associated to `Atom` is atomically incremented.

- **The following properties should hold at call time:**
  - `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Atom` is an atom. (basic_props:atm/1)

**atom_lock_state/2:**

Usage 1: `atom_lock_state(Atom, Value)`

- **Description:** Sets the semaphore associated to `Atom` to `Value`. This is usually done at the beginning of the execution, but can be executed at any time. If not called, semaphore associated to atoms are by default inited to 1. It should be used with caution: arbitrary use can transform programs using locks in a mess of internal relations. The change of a semaphore value in a place other than the initialization stage of a program is **not** among the allowed operations as defined by Dijkstra [Dij65,BA82].

- **The following properties should hold at call time:**
  - `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Value` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Atom` is an atom. (basic_props:atm/1)
  - `Value` is an integer. (basic_props:int/1)

Usage 2: `atom_lock_state(Atom, Value)`

- **Description:** Consults the `Value` of the semaphore associated to `Atom`. Use sparingly and mainly as a medium to check state correctness. Not among the operations on semaphore by Dijkstra.

- **The following properties should hold at call time:**
  - `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Value` is a free variable. (term_typing:var/1)
  - `Atom` is an atom. (basic_props:atm/1)
  - `Value` is an integer. (basic_props:int/1)

**concurrent/1:**

**concurrent F/A**

The predicate named `F` with arity `A` is made concurrent in the current module at runtime (useful for predicate names generated on-the-fly). This difficulties a better compile-time analysis, but in turn offers more flexibility to applications. It is also faster for some applications: if several agents have to share data in a strucuted fashion (e.g., the generator knows and wants to restrict the data generated to a set of other threads), a possibility is to use the same concurrent fact and emply a field within the fact to distinguish the receiver/sender. This can cause many threads to access and wait on the same fact, which in turns can create contention problems. It is much better to create a new concurrent fact and to use that new name as a channel to communicate the different threads. `concurrent/1` can either be given a predicate spec in the form `Name/Arity`, with `Name` and `Arity` bound, or to give a value only to `Arity`, and let the system choose a new, unused `Name` for the fact.

Usage: `concurrent Spec`

- **The following properties should hold at call time:**
  - `Spec` is currently a term which is not a free variable. (term_typing:nonvar/1)
79.3 Known bugs and planned improvements (concurrency)

- Available only for Windows 32 environments and for architectures implementing POSIX threads.
- Some implementation of threads have a limit on the total number of threads that can be created by a process. Thread creation, in this case, just hangs. A better solution is planned for the future.
- Creating many concurrent facts may fill up the atom table, causing Ciao Prolog to abort.
80 All solutions concurrent predicates

Author(s): Concurrent-safe (and incomplete) version of the aggregates predicates, based on the regular versions by Richard A. O’Keefe and David H.D. Warren. Concurrency-safeness provided by Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#97 (2003/12/21, 20:7:18 CET)

This module implements thread-safe aggregation predicates. Its use and results should be the same as those in the aggregates library, but several goals can use them concurrently without the interference and wrong results (due to implementation reasons) aggregates might lead to. This particular implementation is completely based on the one used in the aggregates library.

80.1 Usage and interface (conc_aggregates)

Library usage:
:- use_module(library(conc_aggregates)).

Exports:
- Predicates:
  findall/3.
- Other modules used:
  - System library modules:
    prolog_sys.

80.2 Documentation on exports (conc_aggregates)

findall/3:

Meta-predicate with arguments: findall(?,goal,?).
Usage: findall(Template, Generator, List)

- Description: A special case of bagof, where all free variables in the Generator are taken to be existentially quantified. Safe in concurrent applications.
- The following properties should hold at call time:
  Generator is currently a term which is not a free variable. (term_typing:nonvar/1)
- The following properties should hold upon exit:
  Template is any term. (basic_props:term/1)
  Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  Set is a list. (basic_props:list/1)

80.3 Known bugs and planned improvements (conc_aggregates)

- Thread-safe setof/3 is not yet implemented.
- Thread-safe bagof/3 is not yet implemented.
81 The socket interface

Author(s): Manuel Carro, Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#58 (2001/2/8, 11:46:41 CET)

This module defines primitives to open sockets, send, and receive data from them. This allows communicating with other processes, on the same machine or across the Internet. The reader should also consult standard bibliography on the topic for a proper use of these primitives.

81.1 Usage and interface (sockets)

- Library usage:
  :- use_module(library(sockets)).
- Exports:
  - Predicates:
    connect_to_socket/3, socket_recv/2, hostname_address/2, socket_shutdown/2, socket_recv_code/3, socket_send/2, select_socket/5, socket_accept/2, bind_socket/3, connect_to_socket_type/4.
  - Regular Types:
    socket_type/1, shutdown_type/1.
- Other modules used:
  - System library modules:
    sockets/sockets_c.

81.2 Documentation on exports (sockets)

connect_to_socket/3: PREDICATE

Usage: connect_to_socket(Host, Port, Stream)

- Description: Calls connect_to_socket_type/4 with SOCK_STREAM connection type. This is the connection type you want in order to use the write/2 and read/2 predicates (and other stream IO related predicates).
- Call and exit should be compatible with:
  Host is an atom. (basic_props:atm/1)
  Port is an integer. (basic_props:int/1)
  Stream is an open stream. (streams_basic:stream/1)
- The following properties should hold at call time:
  Host is currently a term which is not a free variable. (term_typing:nonvar/1)
  Port is currently a term which is not a free variable. (term_typing:nonvar/1)
  Stream is a free variable. (term_typing:var/1)

socket_recv/2: PREDICATE

Usage: socket_recv(Stream, String)
Description: As socket_recv_code/3, but the return code is ignored.

Call and exit should be compatible with:
- Stream is an open stream.
- String is a string (a list of character codes).

The following properties should hold at call time:
- Stream is currently a term which is not a free variable.

socket_type/1:
REGTYPE
Defines the atoms which can be used to specify the socket type recognized by connect_to_socket_type/4. Defined as follows:
- socket_type(stream).
- socket_type(dgram).
- socket_type(raw).
- socket_type(seqpacket).
- socket_type(rdm).

Usage: socket_type(T)
- Description: T is a valid socket type.

shutdown_type/1:
REGTYPE
Usage: shutdown_type(T)
- Description: T is a valid shutdown type.

hostname_address/2:
PREDICATE
Usage: hostname_address(Hostname, Address)
- Description: Address is unified with the atom representing the address (in AF_INET format) corresponding to Hostname.
- Call and exit should be compatible with:
  - Hostname is an atom. (basic_props:atm/1)
  - Address is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  - Hostname is currently a term which is not a free variable. (term_typing:nonvar/1)

socket_shutdown/2:
PREDICATE
Usage: socket_shutdown(Stream, How)
- Description: Shut down a duplex communication socket with which Stream is associated. All or part of the communication can be shutdown, depending on the value of How. The atoms read, write, or read_write should be used to denote the type of closing required.
- Call and exit should be compatible with:
  - Stream is an open stream. (streams_basic:stream/1)
  - How is a valid shutdown type. (sockets:shutdown_type/1)
- The following properties should hold at call time:
  - Stream is currently a term which is not a free variable. (term_typing:nonvar/1)
  - How is currently a term which is not a free variable. (term_typing:nonvar/1)
socket_recv_code/3:
Usage: socket_recv_code(Stream, String, Length)

- Description: Receives a String from the socket associated to Stream, and returns its Length. If Length is -1, no more data is available.
- Call and exit should be compatible with:
  Stream is an open stream.  (streams_basic:stream/1)
  String is a string (a list of character codes).  (basic_props:string/1)
  Length is an integer.  (basic_props:int/1)

- The following properties should hold at call time:
  Stream is currently a term which is not a free variable.  (term_typing:nonvar/1)

socket_send/2:
Usage: socket_send(Stream, String)

- Description: Sends String to the socket associated to Stream. The socket has to be in connected state. String is not supposed to be NULL terminated, since it is a Prolog string. If a NULL terminated string is needed at the other side, it has to be explicitly created in Prolog.
- Call and exit should be compatible with:
  Stream is an open stream.  (streams_basic:stream/1)
  String is a string (a list of character codes).  (basic_props:string/1)

- The following properties should hold at call time:
  Stream is currently a term which is not a free variable.  (term_typing:nonvar/1)
  String is currently a term which is not a free variable.  (term_typing:nonvar/1)

select_socket/5:
Usage: select_socket(Socket, NewStream, TO_ms, Streams, ReadStreams)

- Description: Wait for data available in a list of Streams and in a Socket. Streams is a list of Prolog streams which will be tested for reading. Socket is a socket (i.e., an integer denoting the O.S. port number) or a free variable. TO_ms is a number denoting a timeout. Within this timeout the Streams and the Socket are checked for the availability of data to be read. ReadStreams is the list of streams belonging to Streams which have data pending to be read. If Socket was a free variable, it is ignored, and NewStream is not checked. If Socket was instantiated to a port number and there are connections pending, a connection is accepted and connected with the Prolog stream in NewStream.
- Call and exit should be compatible with:
  Socket is an integer.  (basic_props:int/1)
  NewStream is an open stream.  (streams_basic:stream/1)
  TO_ms is an integer.  (basic_props:int/1)
  Streams is a list of streams.  (basic_props:list/2)
  ReadStreams is a list of streams.  (basic_props:list/2)

- The following properties should hold at call time:
  Socket is currently a term which is not a free variable.  (term_typing:nonvar/1)
  NewStream is a free variable.  (term_typing:var/1)
socket_accept/2:
Usage: socket_accept(Sock, Stream)
- Description: Creates a new Stream connected to Sock.
- Call and exit should be compatible with:
  Sock is an integer. (basic_props:int/1)
  Stream is an open stream. (streams_basic:stream/1)
- The following properties should hold at call time:
  Sock is currently a term which is not a free variable. (term_typing:nonvar/1)
  Stream is a free variable. (term_typing:var/1)

bind_socket/3:
Usage: bind_socket(Port, Length, Socket)
- Description: Returns an AF_INET Socket bound to Port (which may be assigned by
  the OS or defined by the caller), and listens to it (hence no listen call in this set of
  primitives). Length specifies the maximum number of pending connections.
- Call and exit should be compatible with:
  Port is an integer. (basic_props:int/1)
  Length is an integer. (basic_props:int/1)
  Socket is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Length is currently a term which is not a free variable. (term_typing:nonvar/1)
  Socket is a free variable. (term_typing:var/1)

connect_to_socket_type/4:
Usage: connect_to_socket_type(Host, Port, Type, Stream)
- Description: Returns a Stream which connects to Host. The Type of connection can
  be defined. A Stream is returned, which can be used to write/2 to, to read/2, to
  socket_send/2 to, or to socket_recv/2 from the socket.
- Call and exit should be compatible with:
  Host is currently instantiated to an atom. (term_typing:atom/1)
  Port is an integer. (basic_props:int/1)
  Type is a valid socket type. (sockets:socket_type/1)
  Stream is an open stream. (streams_basic:stream/1)
- The following properties should hold at call time:
  Host is currently a term which is not a free variable. (term_typing:nonvar/1)
  Port is currently a term which is not a free variable. (term_typing:nonvar/1)
  Type is currently a term which is not a free variable. (term_typing:nonvar/1)
  Stream is a free variable. (term_typing:var/1)
82 Sockets I/O

Author(s): Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#117 (2003/12/22, 18:58:12 CET)
This module provides two useful predicates for programming with sockets.

82.1 Usage and interface (sockets_io)

- Library usage:
  :- use_module(library(sockets_io)).
- Exports:
  - Predicates:
    serve_socket/3, safe_write/2.
- Other modules used:
  - System library modules:
    lists, file_utils, sockets/sockets.

82.2 Documentation on exports (sockets_io)

serve_socket/3: PREDICATE

Meta-predicate with arguments: serve_socket(? ,pred(1) ,pred(1)).
Usage: serve_socket(Socket, Server, Handler)
  - Description: Handles the streams associated to Socket calling Server on one request of each stream (as Server(Stream)), and Handler(Stream) if the stream is empty (connection closed).
  - The following properties should hold at call time:
    Socket is a socket id. (sockets_io:socket/1)
    Server is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
    Handler is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

safe_write/2: PREDICATE

Usage: safe_write(Stream, Term)
  - Description: Writes Term to Stream in a way that it is safe for a socket connection on Stream.
  - The following properties should hold at call time:
    Stream is an open stream. (streams_basic:stream/1)
    Term is any term. (basic_props:term/1)
83 The Ciao Make Package

Author(s): Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#98 (2003/12/21, 20:10:12 CET)

This package is used mainly in two main ways:
- When writing Makefiles for lpmake.
- When writing applications which use the make library.

In both cases, this is the package that defines the syntax and meaning of the dependency rules used.

83.1 Usage and interface (make)

- Library usage:
  - When writing Makefiles for lpmake, such makefiles start with:
    ```
    :- module(_,_,[make]).
    or
    :- make(_,_).
    (The latter uses the feature that an undefined declaration at the beginning of a file is interpreted by Ciao as a use_module/3 including as third argument a package with the same name, in this case make.)
    - When writing applications which use the make package, then it is loaded as any other package within the application.
    Note: it is often useful to use the functions package inside a Makefile (or when using the make library in other applications). If both make and functions are used, then make should appear before functions in the list of packages.
  - New operators defined:
    ```
    ::/2 [1050,xfy], <=/2 [1050,xfy], <->/2 [1050,xfy], <->/1 [1050,yf].
    - Other modules used:
      - System library modules:
        ```
        make/make_rt.
      ```

83.2 Other information (make)

83.2.1 The Dependency Rules

The package allows defining the following types of rules:

\[ \text{TargetSuffix} \Leftarrow \text{SourceSuffix} :: \text{SourceRoot} :\text{BodyLiterals}. \]

A rule of this form declares that in order to produce the file with suffix \text{TargetSuffix} from a source file with the suffix \text{SourceSuffix} and root name \text{SourceRoot} the commands in \text{BodyLiterals} must be executed. \text{BodyLiterals} is a standard Ciao Prolog clause body, i.e., a comma-separated conjunction of literals. When writing the script, \text{SourceRoot} is typically left as a variable, to be instantiated by lpmake when the script is run to the root of name of the file to be processed. This allows using the value of \text{SourceRoot} in \text{BodyLiterals}. For example, the following rule:
:- use_module(library(terms), [atom_concat/2]).

dvi <= tex :: FileRoot :-
    atom_concat(['latex ', FileRoot, '.tex'], Command),
    system(Command).

states that we can generate a file File.dvi if we have a file named File.tex and that the command to do so is latex File.tex. Thus, if this rule appears in file Makefile.pl and we issue the command lpmake paper.dvi the following occurs:

- If paper.dvi does not exist and paper.tex exists, then paper.dvi is generated from paper.tex by issuing the system command latex paper.tex.
- If paper.dvi already exists, nothing is done.
- If paper.tex does not exist, an error is reported.

Target <- :- BodyLiterals.

A rule of this form declares that in order to produce the file Target the commands in BodyLiterals must be executed. Target need not be a real file: it can also be simply the name of the rule, which is used to invoke it (as a procedure name). For example, the following rule, when the command lpmake realclean is issued, deletes temporary files in the LaTeX application:

:- use_module(library('make/system_extra')).

clean <- :-
    ls('*aux|*log|*~', Files)
    delete_files(Files).

Target <- Deps :- BodyLiterals.

A rule of this form declares that in order to produce the file Target, first targets Deps will be called (i.e., the elements of Deps are either other targets with rules defined for them, or a file or files which are already present or which can—and will be—generated from other available files using other rules). Then, the commands in BodyLiterals will be executed. Deps may be one target or a list of targets. For example, the following rule, when the command lpmake realclean is issued, cleans all the temporary files in the LaTeX application (including .dvi and .ps files). It requires that clean be executed first:

:- use_package(functions).
:- use_module(library('make/system_extra')).

realclean <- clean :-
    delete_files(~ls('*dvi|*ps')).

The following rule states that in order to meet the target view, target paper.ps must be available or generated. For example, lpmake view can be used to call the ghostview visualizer on paper.ps. Note the use of a globally defined predicate main which is called in two places in the rule, and could be used in other rules in the same file (main := paper is equivalent to the fact main(paper).—see the functions library):

:- use_package(functions).
:- use_module(library('make/system_extra')).
:- use_module(library(terms), [atom_concat/2]).

main := paper.

view <- ~atom_concat(['main', '.ps']) :-
In addition to these rules, the configuration file can define normal predicates in the usual way, or import predicates from other modules, all of which can be called from the bodies of the dependency rules. For example, the system_extra library (an extension of the system library) defines many system predicates in a form which makes them very useful inside Makefiles, specially if the functions package is used (see the examples below).

If lpmake is called without an explicit target as argument, then the first target rule in the Makefile is used. This is useful in that the first rule can be seen as the default rule.

### 83.2.2 Specifying Paths

Using the vpath/1 predicate it is possible in configuration files to define several paths in which files related to the rules can be located. In this way, not all files need to be in the same directory as the configuration file. For example:

```prolog
:- use_package(functions).

vpath := '/home/clip/Systems/ciao/lib'.
vpath := '/home/clip/Systems/ciao/library'.
vpath := '/home/clip/Systems/lpdoc/lib'.
```

### 83.2.3 Documenting Rules

It is also possible to define documentation for the rules:

```prolog
target_comment(Target) :- BodyLiterals.
```

A rule of this form allows documenting the actions related to the target. The body (BodyLiterals) will be called in two circumstances:

- If Target is called during execution of 'lpmake commands'.
- When calling 'lpmake -h'.

Using noun forms (generation of foo instead of generating foo) in comments helps this dual purpose. For example, the following rule:

```prolog
target_comment(realclean) :-
    display('Cleanup of all generated files.').
```

will produce output in the two cases pointed out above.

```prolog
dependency_comment(SourceSuffix, TargetSuffix, SourceRoot) :- BodyLiterals.
```

Same as the previous rule, but for suffix rules. See, for example, the following generic rule:

```prolog
:- use_module(library(terms),[atom_concat/2]).

dependency_comment(SSuffix,TSuffix,FileBase) :-
    display(~atom_concat([['Generation of ',FileBase,'.',TSuffix, ' from ',FileBase,'.',SSuffix]])).
```

### 83.2.4 An Example of a Makefile

The following is a simple example of a Makefile showing some basic functionality (this is MakefileExample.pl in the example_simple directory in the make library.):

```prolog
%% -------------------------------------------------------------------------
:- module(_,_,[make,functions]).
:- use_module(library('make/system_extra')).
```
:- use_module(library(lists), [append/3, list_concat/2]).
:- use_module(library(terms), [atom_concat/2]).

:- discontiguous(comment/2).

%% --------------------------------------------------------------------
%% A simple target. Defines how to produce file 'hw'.

hw <= [] :-
    writeln("Hello world", hw).

%% A comment describing this target (see below):
%% comment(hw, ['Generation of file hw']).

%% --------------------------------------------------------------------
%% A target with a dependency. 'hwhw' requires 'hw'.

hwhw <= [hw] :-
    readf(hw, Content),
    append(Content, [0'n|Content], DoubleContent),
    writeln(DoubleContent, hwhw).

comment(hwhw, ['Generation of file hwhw']).

%% --------------------------------------------------------------------
%% A simple target. Defines how to produce file 'datafile.simple'.

'datafile.simple' <= :-
    writeln("Hello world", 'datafile.simple').

comment('datafile.simple', ['Generation of file datafile.simple']).

%% --------------------------------------------------------------------
%% A dependency based on suffixes:
%% %<file>.double is generated always from <file>.simple

double <= simple :: Name :-
    readf(atom_concat([Name, '.simple']), Content),
    append(Content, [0'n|Content], DoubleContent),
    writeln(DoubleContent, atom_concat([Name, '.double'])).

%% --------------------------------------------------------------------
%% A dependency based on suffixes with a precondition.
%% %<file>.double is generated always from <file>.simple, once
%% %precond is done

boo <= :-
    display((double <= simple :: name <= precond :- body1, body2)).

%% --------------------------------------------------------------------
%% Example using library predicates
clean :-
    delete_files(~ls('*'|*.asr|*.itf|*.po')).

comment(clean,['Cleanup of temporary files']).

realclean <- clean :-
    delete_files(~ls('hw|hwhw|*simple|*double')).

comment(realclean,['Cleanup of all generated files']).

%%% Reporting progress and documenting commands:
%%% If target_comment/1 is defined it can be used to produce user-defined
%%% output when targets are processed and/or documentation on what each
%%% target does (used for example when lpmake is called with -h). Using
%%% 'generation of foo' instead of 'generating foo' in comments helps in this
%%% dual purpose.

%%% Make calls target_comment/1 for simple targets:
target_comment(Target) :-
    comment(Target,Comment),
    display(~atom_concat([~atom_concat(Comment), '\n'])).

%%% Similarly, make calls dependency_comment/3 for dependencies (only
%%% during execution, not when documenting -h).
dependency_comment(SSuffix,TSuffix,FileBase) :-
    display(~atom_concat([['Generation of ',FileBase,'.',TSuffix,
                          ' from ',FileBase,'.',SSuffix]])).

The following are a few commands that can be used on the previous file (see file
CommandsToTry in the example_simple directory in the make library):

lpmake -m MakefileExample.pl hwhw
(Generate file hwhw --needs to generate file hw first)

lpmake -m MakefileExample.pl datafile.double
(Generate file datafile.double --needs to generate file
datafile.simple first)

lpmake -m MakefileExample.pl realclean
(Cleanup)

lpmake -h -m MakefileExample.pl
(Help on general use of lpmake and commands available in MakefileExample.pl)
See also the LaTeX example in the `example_latex` directory in the `make` library.
84 Predicates Available When Using The Make Package

Author(s): Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#129 (2003/12/30, 22:8:24 CET)

This is the run-time module which implements the predicates which are provided when using the make library package in a given application. For example, they are used internally by lpmake.

84.1 Usage and interface (make_rt)

- Library usage:
  This module is loaded automatically when the make library package is used.
- Exports:
  - Predicates:
    make/1, make_option/1, verbose_message/2, call_unknown/1, dyn_load_cfg_module_into_make/1.
  - Regular Types:
    target/1.
- Other modules used:
  - System library modules:
    compiler/compiler, filenames, terms, system, messages, format.

84.2 Documentation on exports (make_rt)

make/1: PREDICATE
Usage: make(TargetList)
- Description: This is the main entry point to the make library. Makes the list of targets one by one and any needed intermediate targets as dictated by the dependency rules.
- The following properties should hold at call time:
  TargetList is a list of targets. (basicProps:list/2)

target/1: REGTYPE
Usage: target(T)
- Description: T is a Makefile target.

make_option/1: PREDICATE
The predicate is of type data.
Usage: make_option(Option)
- Description: Asserting/retracting facts of this predicate sets/clears library options. Default is no options (i.e., the predicate is undefined). The following values are supported:
make_option(’-v’). % Verbose: prints progress messages (for % debugging rules).

- The following properties should hold at call time:
  Option is an atom. (basic_props:atm/1)

**verbose_message/2:**

**Usage:** verbose_message(Text, ArgList)

- **Description:** The text provided in Text is printed as a message, using the arguments in ArgList, if make_option(’-v’) is defined. Otherwise nothing is printed.
- **The following properties should hold at call time:**
  Text is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with name/2. (format:format_control/1)
  ArgList is a list. (basic_props:list/1)

**call_unknown/1:**

call_unknown(G)

This is a local copy, to make package independent. Complication is so that flags are left as they were also upon failure.

**dyn_load_cfg_module_into_make/1:**

**Usage:** dyn_load_cfg_module_into_make(ConfigFile)

- **Description:** Used to load dynamically a module (typically, a Makefile) into the make library from the application using the library.
- **The following properties should hold at call time:**
  ConfigFile is a source name. (streams_basic:sourcename/1)
85 system_extra (library)

Author(s): M. Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#198 (2004/2/16, 17:28:46 CET)

This is a (temporary) extension to library system (which it reexports). It implements functionality that is often convenient in Makefiles. Much of this should probably end up eventually in system, but once we have worked out the best interface and, in some cases, the proper implementation (the implementations in here are in some cases just calls to Un*x shell primitives or commands).

85.1 Usage and interface (system_extra)

- Library usage:
  
  `:- use_module(library(system_extra)).`

- Exports:

  - Predicates:
    
    del_dir_if_empty/1, move_files/2, move_file/2, copy_files/2, cat/2, cat_append/2, convert_permissions/4, symbolic_link/2, symbolic_link/3, delete_files/1, delete_files_nofail/1, del_file_nofail/1, del_file_nofail/2, del_endings_nofail/2, ls/3, ls/2, filter_alist_pattern/3, do/2, set_perms/2, rea df/2, datime_string/1, datime_string/2, datime_atom/1, datime_atom/2, all_values/2, no_tr_nl/2, call_unknown/1, replace_strings_in_file/3, write/3, writef/2, add_suffix/3, add_prefix/3, writef_list/3, writef_list/2, delete_files/1, etags/2, etags/2, get_dirs/2, get_rec_dirs/2, get_rec_dirs/3, get_rec_dirs/4, lss/3, lss/4, lss/6, copy_files_rec/3, filter_dirs/3, touch/1.

- Other modules used:

  - System library modules:
    system, patterns, filenames, messages, terms, lists, sort, aggregates, sets.

85.2 Documentation on exports (system_extra)

**del_dir_if_empty/1:**

No further documentation available for this predicate.

**move_files/2:**

`move_files(Files, Dir)`

Move Files to directory Dir (note that to move only one file to a directory, `rename_file/2` can be used).

Usage:

- The following properties should hold at call time:
  
  Files is a list of atoms.
  Dir is an atom.
move_file/2:  
No further documentation available for this predicate.

copy_files/2:  
copy_files(Files, Dir)  
Copy Files to directory Dir (note that to move only one file to a directory, rename_file/2 can be used).

Usage:  
The following properties should hold at call time:  
Files is a list of atms.  
Dir is an atom.

Usage:

Usage: symbolic_link(Source, Dir)  
Description: Create a symbolic link in Dir pointing to file or directory Source (performs a copy in Windows).

symbolic_link/3:  
Usage: symbolic_link(Source, Dir, NewName)  
Description: Create a symbolic link in Dir pointing to file or directory Source and give it name NewName (performs a copy in Windows).

delete_files/1:  
No further documentation available for this predicate.

delete_files_nofail/1:  
No further documentation available for this predicate.
del_file_nofail/1:
No further documentation available for this predicate.

PREDICATE

del_file_nofail/2:
No further documentation available for this predicate.

PREDICATE

del_endings_nofail/2:
No further documentation available for this predicate.

PREDICATE

ls/3:

Usage:

– The following properties should hold at call time:
  Directory is an atom. (basic_props:atm/1)
  Pattern is a pattern to match against. (patterns:pattern/1)
  FileList is a free variable. (term_typing:var/1)

– The following properties hold upon exit:
  FileList is a list of atms. (basic_props:list/2)

PREDICATE

ls/2:

Usage:

– The following properties should hold at call time:
  Pattern is a pattern to match against. (patterns:pattern/1)
  FileList is a free variable. (term_typing:var/1)

– The following properties hold upon exit:
  FileList is a list of atms. (basic_props:list/2)

PREDICATE

filter_alist_pattern/3:

Usage:

– The following properties should hold at call time:
  UnFiltered is a list of atms. (basic_props:list/2)
  Pattern is a pattern to match against. (patterns:pattern/1)
  Filtered is a free variable. (term_typing:var/1)

– The following properties hold upon exit:
  Filtered is a list of atms. (basic_props:list/2)
-/1: 
   No further documentation available for this predicate. 
   \textit{Meta-predicate} with arguments: \texttt{-goal}.

\texttt{do/2}: 
   No further documentation available for this predicate.

\texttt{set_perms/2}: 
   No further documentation available for this predicate.

\texttt{readf/2}: 
   No further documentation available for this predicate.

\texttt{datetime_string/1}: 
   No further documentation available for this predicate.

\texttt{datetime_string/2}: 
   No further documentation available for this predicate.

\texttt{datetime_atom/1}: 
   No further documentation available for this predicate.

\texttt{datetime_atom/2}: 
   No further documentation available for this predicate.

\texttt{all_values/2}: 
   No further documentation available for this predicate. 
   \textit{Meta-predicate} with arguments: \texttt{all_values(pred(1),?).}

\texttt{no_tr_nl/2}: 
   No further documentation available for this predicate.

\texttt{call_unknown/1}: 
   No further documentation available for this predicate. 
   \textit{Meta-predicate} with arguments: \texttt{call_unknown(goal)}. 

PREDOCUMENT
replace_strings_in_file/3:
   No further documentation available for this predicate.

writef/3:
   No further documentation available for this predicate.

writef/2:
   No further documentation available for this predicate.

add_suffix/3:
   No further documentation available for this predicate.

add_prefix/3:
   No further documentation available for this predicate.

writef_list/3:
   No further documentation available for this predicate.

writef_list/2:
   No further documentation available for this predicate.

delete_fileless/1:
   No further documentation available for this predicate.

etags/2:
   No further documentation available for this predicate.

etagss/2:
   No further documentation available for this predicate.

get_dirs/2:
   No further documentation available for this predicate.

get_rec_dirs/2:
   No further documentation available for this predicate.
get_rec_dirs/3:  
No further documentation available for this predicate.

get_rec_dirs/4:  
No further documentation available for this predicate.

lss/3:  
No further documentation available for this predicate.

lss/4:  
No further documentation available for this predicate.

lss/6:  
No further documentation available for this predicate.

copy_files_rec/3:  
No further documentation available for this predicate.

filter_dirs/3:  
No further documentation available for this predicate.

touch/1:  
touch(File)  
Updates the access and modification time of File to current time.

cyg2win/3:  
Imported from system (see the corresponding documentation for details).

rename_file/2:  
Imported from system (see the corresponding documentation for details).

delete_directory/1:  
Imported from system (see the corresponding documentation for details).

delete_file/1:  
Imported from system (see the corresponding documentation for details).
chmod/3: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

chmod/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

fmode/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

modif_time0/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

modif_time/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

file_properties/6: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

file_property/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

file_exists/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

file_exists/1: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

mktemp/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

directory_files/2: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).

exec/3: (UNDOD_REEXPORT)
Imported from system (see the corresponding documentation for details).
exec/4: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

popen_mode/1: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

popen/3: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

system/2: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

system/1: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

shell/2: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

shell/1: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

shell/0: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

cd/1: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

working_directory/2: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

make_dirpath/1: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).

make_dirpath/2: (UNDOK_REEXPORT)
Imported from system (see the corresponding documentation for details).
make_directory/1:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

make_directory/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

umask/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

current_executable/1:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

current_host/1:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

get_pid/1:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

extract_paths/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

copy_file/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

del_env/1:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

set_env/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

current_env/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).

setenvstr/2:  (UNDOC_REEXPORT)
   Imported from system (see the corresponding documentation for details).
getenvstr/2: (UNDIC_REEXPORT)
    Imported from system (see the corresponding documentation for details).

datetime_struct/1: (UNDIC_REEXPORT)
    Imported from system (see the corresponding documentation for details).

datetime/9: (UNDIC_REEXPORT)
    Imported from system (see the corresponding documentation for details).

datetime/1: (UNDIC_REEXPORT)
    Imported from system (see the corresponding documentation for details).

time/1: (UNDIC_REEXPORT)
    Imported from system (see the corresponding documentation for details).

pause/1: (UNDIC_REEXPORT)
    Imported from system (see the corresponding documentation for details).
PART VII - Ciao Prolog extensions

Author(s): The CLIP Group.

The libraries documented in this part extend the Ciao language in several different ways. The extensions include:

- pure Prolog programming (well, this can be viewed more as a restriction than an extension);
- feature terms or *records* (i.e., structures with names for each field);
- parallel programming (e.g., &-Prolog style);
- functional syntax;
- higher-order library;
- global variables;
- `setarg` and `undo`;
- delaying predicate execution;
- active modules;
- breadth-first execution;
- iterative deepening-based execution;
- constraint logic programming;
- object oriented programming.
86 Pure Prolog package

Author(s): The CLIP Group.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#100 (2003/12/21, 20:16:33 CET)

This library package allows the use of pure Prolog in a Ciao module/program. It is based on the fact that if an engine module is imported explicitly then all of them have to be imported explicitly. The engine modules are:

- **engine(arithmetic)**
  Chapter 20 [Arithmetic], page 105.
- **engine(atomic_basic)**
  Chapter 19 [Basic predicates handling names of constants], page 99.
- **engine(attributes)**
  Chapter 28 [Attributed variables], page 151.
- **engine(basic_props)**
  Chapter 15 [Basic data types and properties], page 79.
- **engine(basiccontrol)**
  Chapter 13 [Control constructs/predicates], page 73.
- **engine(data_facts)**
  Chapter 25 [Fast/concurrent update of facts], page 137.
- **engine(exceptions)**
  Chapter 23 [Exception handling], page 129.
- **engine(io_aux)**
  Chapter 27 [Message printing primitives], page 147.
- **engine(io_basic)**
  Chapter 22 [Basic input/output], page 121.
- **engine(prolog_flags)**
  Chapter 24 [Changing system behaviour and various flags], page 131.
- **engine(streams_basic)**
  Chapter 21 [Basic file/stream handling], page 111.
- **engine(system_info)**
  Chapter 29 [Gathering some basic internal info], page 155.
- **engine(term_basic)**
  Chapter 17 [Basic term manipulation], page 91.
- **engine(term_compare)**
  Chapter 18 [Comparing terms], page 95.
- **engine(term_typing)**
  Chapter 16 [Extra-logical properties for typing], page 87.

Note that if any of these modules is explicitly imported in a program then the language defaults to Pure Prolog, plus the functionality added by the modules explicitly imported.

It is recommended that if you explicitly import an engine module you also use this package, which will guarantee that the predicate true/0 is defined (note that this is the only Ciao builtin which cannot be redefined).
86.1 Usage and interface (pure)

- Library usage:
  ```prolog
  :- use_package(pure).
  or
  :- module(...,...,[pure]).
  ```

86.2 Known bugs and planned improvements (pure)

- Currently, the following builtin predicates/program constructs cannot be redefined, in addition to true/0: (->)/2 (,)/2 (+)/1 if/3
Chapter 87: Multiple Argument Indexing

Author(s): Tom Howland (http://home.pacbell.net/tomjdnh/pd.html), derived from work by Anil Nair, F. Bueno (for the Ciao package).

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#101 (2003/12/21, 20:21:31 CET)

This package is an extension of the idea of Prolog indexing, usually performed, in a limited way, on the first argument. This package provides more powerful indexing schemes. It lets you pick different arguments to index on, and provides for different combinations of arguments to index on. E.g., it will let you index on the first and third argument or the second and the third argument of a predicate.

87.1 Usage and interface (indexer)

- Library usage:
  This facility is used as a package, thus either including indexer in the package list of the module, or by using the use_package/1 declaration. The facility predicate hash_term/2, documented here, is defined in library module hash.

- Other modules used:
  - System library modules:
    assertions/native_props.

87.2 Documentation on internals (indexer)

index/1: DECLARATION

Usage: :- index(IndexSpecs).

Description: Declares an indexing scheme for a predicate. All specs of IndexSpecs must be terms for the same predicate. Each spec declares an indexing on a combination of the arguments. Indexing will be performed using any of the specs in IndexSpecs (being thus interpreted as an or).

You should use a * in an argument position if you wish to hash on the entire term in that argument. If a + is used only one level of the term in the argument is used for hashing. An i is used to indicate that argument is already an integer, and therefore its own value will be used for hashing. The argspec ? simply indicates not to use the argument for indexing.

For example, the index specification:

:- index foo(+,?,*,i), foo(?,,?,i).

declares indexing for foo/4 either on a combination of the first, third, and fourth arguments, or only on the last argument, which is an integer. In the first case, only the principal functor of the first argument will be used for hashing; the third argument will be used in its entirety.

The argspec n is a pragmatic extension and can not be used in conjunction with the other specifiers aside from ?. It stands for "nonvar" and implies that the argument will not be used for hashing, since only ground terms can effectively be used in hashing. Thus, it can not be used in combination with other specifiers within a particular index specification. It is often the fastest thing to use.
- The following properties should hold upon exit:
  IndexSpecs is an index specification. (indexer_doc:indexspecs/1)

indexspecs/1:
An index specification is defined as follows:

\[
\text{indexspecs}(\text{Spec}) :- \\
\quad \text{indexspec}(\text{Spec}). \\
\text{indexspecs}([\text{Spec}, \text{Specs}]) :- \\
\quad \text{indexspec}(\text{Spec}), \\
\quad \text{indexspecs}((\text{Specs}). \\
\text{indexspec}(\text{Spec}) :- \\
\quad \text{Spec}=..[_F|\text{Args}], \\
\quad \text{list}(\text{Args}, \text{argspec}).
\]

Usage: indexspecs(IndexSpecs)
- Description: IndexSpecs is an index specification.

argspec/1:
An argument hash specification is defined as follows:

\[
\text{argspec}(+).
\text{argspec}(*) .
\text{argspec}(i).
\text{argspec}(n).
\text{argspec}(?).
\]

Usage: argspec(Spec)
- Description: Spec is an argument hash specification.

hash_term/2:
hash_term(Term, HashValue)
Provides an efficient way to calculate an integer HashValue for a ground Term.

Usage 1: hash_term(T, N)
- Description: N is a hashing index for T.
- The following properties should hold at call time:
  T is currently ground (it contains no variables). (term_typing:ground/1)
  N is a free variable. (term_typing:var/1)
- The following properties should hold upon exit:
  N is an integer. (basic_props:int/1)

Usage 2: hash_term(T, N)
- The following properties should hold at call time:
  T is not ground. (native_props:nonground/1)
  N is a free variable. (term_typing:var/1)
- The following properties should hold upon exit:
  N is a free variable. (term_typing:var/1)
88 Higher-order

Author(s): Daniel Cabeza Gras.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#102 (2003/12/22, 16:52:51 CET)

This module is a wrapper to the implementation defined predicate call/1, and implements the call/2 predicate.

88.1 Usage and interface (hiord_rt)

- Library usage:
  :- use_module(library(hiord_rt)).

- Exports:
  - Predicates:
    call/1, call/2, SYSCALL/1, $nodebug_call/1, $meta_call/1.

88.2 Documentation on exports (hiord_rt)

call/1: PREDICATE

call(G)

Executes goal G, restricting the scope of the cuts to the execution of G. Equivalent to writing a variable G in a goal position.

Meta-predicate with arguments: call(goal).

Usage: ☑ ISO ☑

- The following properties should hold at call time:
  G is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

call/2: PREDICATE

call(Pred, Arg1)

There exists a set of built-in predicates of the form call/N with N > 1 which execute predicate Pred given arguments Arg1 ... ArgX. If Pred has already arguments Arg1 is added to the start, the rest to the end. This predicate, when Pred is a variable, can be written using the special Ciao syntax Pred(Arg1,...,ArgX).

Usage: ☑ ISO ☑

- The following properties should hold at call time:
  Pred is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (basic_props:native/1)
SYSCALL/1:
   No further documentation available for this predicate.

$nodebug_call/1:
   No further documentation available for this predicate.
   *Meta-predicate* with arguments: $nodebug_call(goal).

$meta_call/1:
   No further documentation available for this predicate.
Chapter 89: Higher-order predicates

Author(s): Daniel Cabeza, Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

This library implements a few basic higher-order predicates. These add functionality to the basic higher-order functionality of Ciao. Examples of the latter are:

Using pred(1):
- list(L, functor(_,2))
- list(L, >())

Using pred(2):

89.1 Usage and interface (hiordlib)

- Library usage:
  ```prolog```
  ```:- use_module(library(hiordlib)).```  
- Exports:
  - **Predicates:**
    - map/3, foldl/4, minimum/3.

89.2 Documentation on exports (hiordlib)

**map/3:**

*Meta-predicate* with arguments: `map(?,pred(2),?)`.

Usage: `map(LList, Op, RList)`

- Description: Examples of use:
  ```prolog```
  ```map([[1,3,2]], arg(f(a,b,c,d)), [a,c,b]) or```
  ```map([[1,3,2]], nth([a,b,c,d]), [a,c,b])```
  ```map(["D","C"], append("."), ["D.","C."])```

**foldl/4:**

*Meta-predicate* with arguments: `foldl(?,?,pred(3),?)`.

Usage: `foldl(List, Seed, Op, Result)`

- Description: Example of use:
  ```prolog```
  ```?- foldl(["daniel","cabeza","gras"], ",",```
  ```(\'(X,Y,Z) :- append(X, "," ||Y, Z)), R).```  
  ```R = "daniel cabeza gras" ?```
minimum/3:  
Meta-predicate with arguments: minimum(? , pred(2) , ?).

Usage: minimum(List, SmallerThan, Minimum)

– Description: Minimum is the smaller in the nonempty list List according to the relation SmallerThan: SmallerThan(X, Y) succeeds iff X is smaller than Y.

– The following properties should hold at call time:
  SmallerThan is currently a term which is not a free variable.  
  (termotyping:nonvar/1)
  List is a list.  
  (basic_props:list/1)
  SmallerThan is a term which represents a goal, i.e., an atom or a structure.  
  (basic_props:callable/1)
  Minimum is any term.  
  (basic_props:term/1)
90 Terms with named arguments -records/feature terms

Author(s): Daniel Cabeza and Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#118 (2001/8/28, 15:7:22 CEST)

This library package provides syntax which allows accessing term arguments by name (these terms are sometimes also referred to as records, and are also similar to feature terms [AKPS92]).

90.1 Usage and interface (argnames)

- Library usage:
  :- use_package(argnames).
  or
  :- module(...,...,[argnames]).

- New operators defined:
  $/2 [150,xfx], =>/2 [950, xfx], argnames/1 [1150,fx].

- New declarations defined:
  argnames/1.

90.2 Documentation on new declarations (argnames)

argnames/1:  

Usage: :- argnames(ArgNamedPredSpec).

\textbf{Description:} An \texttt{argnames/1} declaration assigns names to the argument positions of terms (or literal/goals) which use a certain functor/arity. This allows referring to these arguments by their name rather than by their argument position. Sometimes, argument names may be clearer and easier to remember than argument positions, specially for predicates with many arguments. Also, in some cases this may allow adding arguments to certain predicates without having to change the code that uses them. These terms with named arguments are sometimes also referred to as records, and are also similar to feature terms [AKPS92]. For example, in order to write a program for the zebra puzzle we might declare:

  :- use_package([argnames]).
  :- argnames house(color, nation, pet, drink, car).

which first includes the package and then assigns a name to each of the arguments of any term (or literal/goal) with \texttt{house/5} as the main functor.

For convenience the package extends the built-in \texttt{data/1} declaration so that names to arguments can be asigned as with the \texttt{argnames/1} declaration, as for example:

  :- data product(id, description, brand, quantity).

Once an \texttt{argnames/1} is given, is possible to use the names to refer to the arguments of any term (or literal/goal) which has the same main functor as that of the term which appears in the \texttt{argnames/1} declaration. This is done by first writing the functor name, then the infix operator $\$,$ and then, between curly brackets, zero, one, or more pairs \texttt{argument-name}=>$\texttt{argument-value}$, separated by commas (i.e., the infix operator $\$,$ is used between the name and the value). Again, argument names must be atomic.
Argument values can be any term. Arguments which are not specified are assumed to have a value of "_" (i.e., they are left unconstrained).

Thus, after the declaration for house/5 in the example above, any occurrence in that code of, for example, house$(\{\text{nation}=>\text{Owns_zebra}, \text{pet}=>\text{zebra}\})$ is exactly equivalent to $\text{house}(\_, \text{Owns_zebra}, \text{zebra}, \_, \_)$. Also, house${\{\}}$ is equivalent to $\text{house}(\_, \_, \_, \_, \_)$. The actual zebra puzzle specification might include a clause such as:

```prolog
zebra(\text{Owns_zebra}, \text{Drinks_water}, \text{Street}) :-
    \text{Street} = [\text{house}\{\}, \text{house}\{\}, \text{house}\{\}, \text{house}\{\}, \text{house}\{\}],
    \text{member}(\text{house}\{\text{nation}=>\text{Owns_zebra}, \text{pet}=>\text{zebra}\}, \text{Street}),
    \text{member}(\text{house}\{\text{nation}=>\text{Drinks_water}, \text{drink}=>\text{water}\}, \text{Street}),
    \text{member}(\text{house}\{\text{drink}=>\text{coffee}, \text{color}=>\text{green}\}, \text{Street}),
    \text{left_right}(\text{house}\{\text{color}=>\text{ivory}\}, \text{house}\{\text{color}=>\text{green}\}, \text{Street}),
    \text{member}(\text{house}\{\text{car}=>\text{porsche}, \text{pet}=>\text{snails}\}, \text{Street}),
    \ldots
```

Another syntax supported, useful mainly in declarations, to avoid specify the arity is house${/}$, which is equivalent in our example to house/5 (but for data declarations there is a special syntax as we have seen).

Any number of argnames/1 declarations can appear in a file, one for each functor whose arguments are to be accessed by name. As with other packages, argument name declarations are local to the file in which they appear. The argnames/1 declarations affect only program text which appears after the declaration. It is easy to make a set of declarations affect several files for example by putting such declarations in a separate file which is included by all such files.

An argnames/1 declaration does not change in any way the internal representation of the associated terms and does not affect run-time efficiency. It is simply syntactic sugar.

### 90.3 Other information (argnames)

Two simple examples of the use of the argnames library package follow.

#### 90.3.1 Using argument names in a toy database

```prolog
:- module(simple_db,_,[argnames,assertions,regtypes]).
:- use_module(library(aggregates)).

:- comment(title,"A simple database application using argument names").

:- data
product( id, description, brand, quantity ).
% ----------------------------------------------------------
product( 1, "Keyboard", "Logitech", 6 ).
product( 2, "Mouse", "Logitech", 5 ).

:- pred product$/\}$ :: int * string * string * int.

% Compute the stock of products from a given brand.
% Note call to findall is equivalent to: findall(Q,product(/,_,Brand,Q),L).
brand_stock(Brand,Stock) :-
```
findall(Q, product{brand=>Brand, quantity=>Q}, L),
sumlist(L, Stock).

sumlist([], 0).
sumlist([X|T], S) :-
    sumlist(T, S1),
    S is X + S1.

90.3.2 Complete code for the zebra example

:- module(_, zebra/3, [argnames]).
/* There are five consecutive houses, each of a different 
color and inhabited by men of different nationalities. They each 
own a different pet, have a different favorite drink, and drive a 
different car.

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. Coffee is drunk in the green house.
4. The Ukrainian drinks tea.
5. The green house is immediately to the right of the ivory 
house.
6. The Porsche driver owns snails.
7. The Masserati is driven by the man who lives in the yellow 
house.
8. Milk is drunk in the middle house.
9. The Norwegian lives in the first house on the left.
10. The man who drives a Saab lives in the house next to the man 
    with the fox.
11. The Masserati is driven by the man in the house next to the 
    house where the horse is kept.
12. The Honda driver drinks orange juice.
14. The Norwegian lives next to the blue house.

The problem is: Who owns the Zebra? Who drinks water?
*/
:- argnames house(color, nation, pet, drink, car).

zebra(Owns_zebra, Drinks_water, Street) :-
    Street = [house{}, house{}, house{}, house{}, house{}],
    member(house{nation => Owns_zebra, pet => zebra}, Street),
    member(house{nation => Drinks_water, drink => water}, Street),
    member(house{nation => englishman, color => red}, Street),
    member(house{nation => spaniard, pet => dog}, Street),
    member(house{drink => coffee, color => green}, Street),
    member(house{nation => ukrainian, drink => tea}, Street),
    left_right(house{color => ivory}, house{color => green}, Street),
    member(house{car => porsche, pet => snails}, Street),
member(house\{car => masserati, color => yellow\}, Street),
Street = [_, _, house\{drink => milk\}, _, _],
Street = [house\{nation => norwegian\}|_],
next_to(house\{car => saab\}, house\{pet => fox\}, Street),
next_to(house\{car => masserati\}, house\{pet => horse\}, Street),
member(house\{car => masserati\}, house\{pet => horse\}, Street),
member(house\{nation => norwegian\}|_],
next_to(house\{nation => norwegian\}|_, house\{color => blue\}, Street).

member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).

left_right(L,R,[L,R|_]).
left_right(L,R,[_|T]) :- left_right(L,R,T).

next_to(X,Y,L) :- left_right(X,Y,L).
next_to(X,Y,L) :- left_right(Y,X,L).

90.4 Known bugs and planned improvements (argnames)
• It would be nice to add a mechanism to portray terms with named arguments in a special
  (user definable) way.
91 Functional notation

Author(s): Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#291 (2004/2/13, 20:46:8 CET)

This library package allows the use of functional notation in a Ciao module/program.

It should be made clear that this package just provides a kind of syntactic sugar for defining and using predicates as if they were functions, and thus any function definition is in fact defining a predicate, and any predicate can be used as a function. The predicate associated to a function has the same name and one more argument, added to the right, to hold the result of the function.

Any term preceded by the operator ~ is a function application, as in write(~arg(1,T)), which is equivalent to the sequence arg(1,T,A), write(A). Functors can be declared as evaluable by using the declaration function/1, and thus avoiding the need to use the operator ~, as in

```
:- function(arg/2).
```

Note that this declaration, as is customary in Ciao Prolog, is local to the source code where it is included. In addition, the package defines several functors as evaluable by default, those being:

- All the functors understood by is/2. This feature can be disabled by a declaration :- function(arith(false)) (and reverted by using true instead of false).
- The functors used for disjunctive and conditional expressions, (|)/2 and (?)/2. A disjunctive expression has the form (V1|V2), and its value when first evaluated is V1, and on re-execution V2. A conditional expression has the form (Cond ? V1), or more commonly (Cond ? V1 | V2), and its value, if the execution of Cond as a goal succeeds, is V1, otherwise in the first form it causes backtracking, and on the second form its value is V2. Note that due to the operator precedences, these expressions need normally to be surrounded by parenthesis.

A functional clause is written using the binary operator :=, as in

```
opposite(red) := green.
```

Functional clauses can also have a body, which is executed before the result value is computed. It can serve as a guard for the clause or to provide the equivalent of a where-clause in a functional language:

```
fact(0) := 1.
fact(N) := N * ~fact(--N) :- N > 0.
```

Note that often a guard can be better defined using a conditional expression:

```
fact(N) := N = 0 ? 1
   | N > 0 ? N * ~fact(--N).
```

In clause heads (either defined as predicates or functions) functors can be prevented from being evaluated by using the (~)/1 prefix operator, as in

```
pair(A,B) := ~(A-B).
```

Note that this just prevents the evaluation of the principal functor of the enclosed term, not the possible occurrences of other evaluable functors inside. The operator is by now ignored outside clause heads, due to the recurrent nature of the goal translations used.

When using function applications inside the goal arguments of meta-predicates, there is an ambiguity as they could be evaluated either in the scope of the outer execution or the in the scope of the inner execution. The chosen behavior is by default to evaluate function applications in the scope of the outer execution, and if they should be evaluated in the inner scope, the goal containing the function application needs to be escaped with the (^^)/1 prefix operator, as in

```
findall(X, (d(Y), ^^ (X = Y+1)), L) (which could also be written as findall(X, ~(d(Y), X = Y+1), L)).
```
91.1 Usage and interface (functions)

- Library usage:
  
  ```prolog
  :- use_package(functions).
  
or
  :- module(...,...,[functions]).
  ```

91.2 Known bugs and planned improvements (functions)

- The (^)/1 operator only works in clause heads.
- Assumes that is/2 is imported.
92 global (library)

Version: 0.4#5 (1998/2/24)

92.1 Usage and interface (global)

- Library usage:
  :- use_module(library(global)).
- Exports:
  - Predicates:
    set_global/2, get_global/2, push_global/2, pop_global/2, del_global/1.

92.2 Documentation on exports (global)

set_global/2:
  No further documentation available for this predicate.

get_global/2:
  No further documentation available for this predicate.

push_global/2:
  No further documentation available for this predicate.

pop_global/2:
  No further documentation available for this predicate.

del_global/1:
  No further documentation available for this predicate.
93 Independent and-parallel execution

Author(s): Manuel Carro, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#160 (2001/11/27, 12:35:53 CET)
Note: This is just a partial first shot. The real library still needs to be written. Not difficult, just no time...

This library will eventually allow and-parallel execution of goals in (Herbrand-)independent fashion. It resembles the execution rules of &-Prolog [HG90]. Basically, goals are run in and-parallel provided that their arguments do not share bindings, i.e., are not bound to terms which contain a common variable.

93.1 Usage and interface (andprolog)

- Library usage:
  :- use_package(andprolog).
  or
  :- module(...,...,[andprolog]).

- New operators defined:
  &/2 [950, xfy].

93.2 Documentation on internals (andprolog)

&/2: PREDICATE
& (GoalA, GoalB)
GoalA and GoalB are run in independent and-parallel fashion. This is just a first sketch, and valid only for deterministic independent goals. The use is as
q :- a & b.

which would start a and b in separate threads (possibly in parallel, if the machine architecture and operating system allows that), and continue when both have finished. This type of execution is safe only when a and b are independent in the sense of variable sharing. This condition can be tested with the indep/2 predicate.

active_agents/1: PREDICATE
active_agents(NumberofAgents)
Tests/sets the NumberofAgents which are active looking for goals to execute. As for now, those agents are resource-consuming, even when they are just looking for work, and not executing any user goals.

indep/2: PREDICATE
indep(X, Y)
X and Y are independent, i.e., they are bound to terms which have no variables in common. For example, indep(X, Y) holds for X=f(Z), Y=g(K) and also for X=f(a), Y=X (since both X and Y are bound to ground terms). It does not hold for X=f(Z), Y=g(Z) and for X=Y.
indep/1: PREDICATE
\[\text{indep}(X)\]

\(X\) is a list of lists of length two, i.e., a list of the form \([([T_1, T_2], [T_3, T_4], \ldots])\). The variables in each pair of the list \(X\) are tested for independence using \text{indep}/2. This list-of-pairs format is the output of several independence analyzers for pair sharing.

93.3 Known bugs and planned improvements (andprolog)

- **Beware**: the current code is just a partial first shot. It is provided for the sole purpose of experimentation and development.
- The fact that only the first solution is returned for the conjunction is due to performance issues (and lack of time), and we expect to remove it in a near future.
- CGEs (i.e., \(\Rightarrow\)) are not supported.
- The \text{indep}/1, \text{indep}/2, and \text{ground}/1 tests are not very efficient; they will be replaced by native versions (taken from the &-Prolog code) in the future.
94 Andorra execution

Author(s): Claudio Vaucheret, Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#144 (2001/11/12, 17:57:47 CET)

This package allows the execution under the Basic Andorra Model [War88]. The model classifies goals as a determinate goal, if at most one clause matches the goal, or nondeterminate goal, otherwise. In this model a goal is delayed until either it becomes determinate or it becomes the leftmost goal and no determinate goal is available. The implementation of this selection rule is based on the use of attributed variables [Hol92,Hol90].

In order to test determinacy we verify only the heads of clauses and builtins in the bodies of clauses before the first cut, if any. By default, determinacy of a goal is detected dynamically: when called, if at most one clause matches, it is executed; otherwise, it is delayed. For goals delayed the test is repeated each time a variable appearing in the goal is instantiated. In addition, efficiency can be improved by using declarations that specify the determinacy conditions. These will be considered for testing instead of the generic test on all clauses that can match.

As with any other Ciao package, the andorra computation rule affects only the module that uses the package. If execution passes across two modules that use the computation rule, determinate goals are run in advance within one module and also within the other module. But determinate goals of one module do not run ahead of goals of the other module.

It is however possible to preserve the computation rule for calls to predicates defined in other modules. These modules should obviously also use this package. In addition all predicates from such modules should imported, i.e., the directive :- use_module(module), should be used in this case instead of :- use_module(module,[...]). Otherwise calls to predicates outside the module will only be called when they became the leftmost goal.

94.1 Usage and interface (andorra)

- Library usage:
  :- use_package(andorra).
  or
  :- module(...,...,[andorra]).
- Exports:
  - Regular Types:
    detcond/1, path/1.
- New operators defined:
  \=/2 [700,xfx], \=/2 [700,xfx].
- New declarations defined:
  determinate/2.

94.2 Documentation on new declarations (andorra)

determinate/2: DECLARATION
  :- determinate(Pred, Cond).

Declares determinacy conditions for a predicate. Conditions Cond are on variables of arguments of Pred. For example, in:
:- determinate(member(A,B,C), ( A ?= term(B,[1]) ; C?=[_[_]) ).

member(A,[A|B],B).
member(A,[B|C],[B|D]) :-
   A==B,
   member(A,C,D).

the declaration states that a call member(A,B,C) is determinate when either A doesn’t unify with the first argument of B or C doesn’t unify with [_|_].

Usage: :- determinate(Pred, Cond).

Description: States that the predicate Pred is determinate when Cond holds.

The following properties should hold at call time:

Pred is a Name/Arity structure denoting a predicate name:

predname(P/A) :-
   atm(P),
   nnegint(A).  (basic_props:predname/1)

Cond is a determinacy condition.  (user(... /andorra_doc):detcond/1)

94.3 Documentation on exports (andorra)

detcond/1:  REGTYPE

Defined by:

- detcond(ground(X)) :-
  var(X).
- detcond(nonvar(X)) :-
  var(X).
- detcond(instantiate(A,Path)) :-
  var(A),
  list(Path,int).
- detcond(\=(Term1,Term2)) :-
  path(Term1),
  path(Term2).
- detcond(\=(Term1,Term2)) :-
  path(Term1),
  path(Term2).
- detcond(Test) :-
  test(Test).

- ground/1 and nonvar/1 have the usual meaning.
- instantiate(A,Path) means that the subterm of A addressed by Path is not a variable. Path is a list of integer numbers describing a path to the subterm regarding the whole term A as a tree. For example, instantiated(f(g(X),h(i(Z),Y)),[2,1]) tests whether i(Z) is not a variable.
- Term1 ?\= Term2 means “terms Term1 and Term2 do not unify (when instantiated)”. Term1 and Term2 can be either an argument of the predicate or a term term(V,Path), which refers to the subterm of V addressed by Path.
- Term1 ?= Term2 means “terms Term1 and Term2 unify (when instantiated)”. The same considerations above apply to Term1 and Term2.
- any other test that does not unify variables can also be used ( ==/2, \=/2, atomic/1).
Usage: detcond(X)

- Description: X is a determinacy condition.

\textbf{path/1:}

Defined by:

\begin{verbatim}
path(X) :-
    var(X).
path(X) :-
    list(X,int).
\end{verbatim}

\section*{94.4 Other information (andorra)}

The andorra transformation will include the following predicates into the code of the module that uses the package. Be careful not to define predicates by these names:

- detcond_andorra/4
- path_andorra/4
- detcond_susp/4
- path_susp/4
- list_andorra2/5
- test_andorra2/4
95 Call on determinate

**Author(s):** José Morales, Manuel Carro.

**Version:** 1.7#149 (2001/11/19, 19:17:51 CET)

Offers an enriched variant of call and cut !!/0 which executes pending goals when the computation has no more alternatives.

This library is useful to, for example, get rid of external connections once the necessary data has been obtained.

### 95.1 Usage and interface (det_hook_rt)

- **Library usage:**
  ```prolog
  :- use_module(library(det_hook_rt)).
  ```
  in which case, !!/0 is not available.
  Typically, this library is used as a package:
  ```prolog
  :- use_package(det_hook).
  ```
- **Exports:**
  - Predicates:
    - `det_try/3`.

### 95.2 Documentation on exports (det_hook_rt)

**det_try/3:**

*Meta-predicate* with arguments: `det_try(goal,goal,goal)`.

*Usage:*

- **Description:** Action is called, and `OnCut` and `OnFail` are goals to be executed when `Goal` is cut or when it finitely fails, respectively. In order for this to work, cutting must be performed in a special way, by using the !!/0 predicate, also provided by this module.

- **The following properties should hold at call time:**
  - `Goal` is a term which represents a goal, i.e., an atom or a structure.  
    (basic_props:callable/1)
  - `OnCut` is a term which represents a goal, i.e., an atom or a structure.  
    (basic_props:callable/1)
  - `OnFail` is a term which represents a goal, i.e., an atom or a structure.  
    (basic_props:callable/1)

### 95.3 Documentation on internals (det_hook_rt)

**!/0:**

*Usage:*

- **Description:** Performs a special cut which prunes alternatives away, as the usual cut, but which also executes the goals specified as `OnCut` for any call in the scope of the cut.
95.4 Other information (det_hook_rt)

As an example, the program

```prolog
:- module(_, _, [det_hook]).

enumerate(X):-
    display(enumerating), nl,
    OnCut = (display('goal cut'), nl),
    OnFail = (display('goal failed'), nl),
    det_try(enum(X), OnCut, OnFail).

enum(1).
enum(2).
enum(3).
```

behaves as follows:

```prolog
?- enumerate(X).
enumerating
X = 1 ;
X = 2 ;
X = 3 ;
goal failed
```

(note the message inserted on failure). The execution can be cut as follows:

```prolog
?- use_package(det_hook).
{Including /home/clip/lib/ciao/ciao-1.7/library/det_hook/det_hook.pl}

yes
?- enumerate(X), '!!'.
enumerating
goal cut
X = 1 ;
```

95.5 Known bugs and planned improvements (det_hook_rt)

- If the started goals do not exhaust their solutions, and '!!'/0 is not used, the database will populate with facts which will be consulted the next time a '!!'/0 is used. This could cause incorrect executions.
96 Miscellaneous predicates

Author(s): Manuel Carro, Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.3#95 (1999/11/8, 18:37:30 MET)

This module implements some miscellaneous non-logical (but sometimes very useful) predicates.

96.1 Usage and interface (odd)

- Library usage:
  
  ```prolog
  :- use_module(library(odd)).
  ```

- Exports:
  
  - Predicates:
    
    ```prolog
    setarg/3, undo/1.
    ```

96.2 Documentation on exports (odd)

setarg/3:  

**Usage:** setarg(Index, Term, NewArg)

**Description:** Replace destructively argument Index in Term by NewArg. The assignment is undone on backtracking. This is a major change to the normal behavior of data assignment in Ciao Prolog.

- **The following properties should hold at call time:**
  
  - `Index` is currently instantiated to an integer.  
  - `Term` is a compound term.  
  - `NewArg` is any term.

- **The following properties hold upon exit:**
  
  - `Index` is currently instantiated to an integer.  
  - `Term` is a compound term.  
  - `NewArg` is any term.

undo/1:  

**Meta-predicate with arguments:** undo(goal).

**Usage:** undo(Goal)

**Description:** call(Goal) is executed on backtracking. This is a major change to the normal control of Ciao Prolog execution.

- **The following properties should hold at call time:**
  
  - `Goal` is a term which represents a goal, i.e., an atom or a structure.

- **The following properties hold upon exit:**
  
  - `Goal` is a term which represents a goal, i.e., an atom or a structure.
97 Delaying predicates (freeze)

**Author(s):** Manuel Carro, Daniel Cabeza.

**Version:** 1.11#222 (2004/5/24, 13:8:7 CEST)

**Version of last change:** 1.5#72 (2000/3/19, 19:9:14 CET)

This library offers a simple implementation of `freeze/2`, `frozen/2`, etc. [Col82, Nai85, Nai91, Car87] based on the use of attributed variables [Hol92, Hol90].

### 97.1 Usage and interface (freeze)

- **Library usage:**
  
  ```prolog
  :- use_module(library(freeze)).
  ```

- **Exports:**
  - **Predicates:**
    
    ```prolog
    freeze/2, frozen/2.
    ```
  - **Multi-files:**
    
    ```prolog
    $combine_attr/4, $check_attr/3.
    ```

- **Other modules used:**
  - **System library modules:**
    
    `mattr_global/mattr_global_code`.

### 97.2 Documentation on exports (freeze)

#### freeze/2:

**Meta-predicate** with arguments: `freeze(?goal)`.

**Usage:** `freeze(X, Goal)`

- **Description:** If `X` is free delay `Goal` until `X` is non-variable.
- **The following properties should hold at call time:**
  
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (**basicProps:callable/1**)

#### frozen/2:

**Meta-predicate** with arguments: `frozen(?goal)`.

**Usage:** `frozen(X, Goal)`

- **Description:** `Goal` is currently delayed until variable `X` becomes bound.
- **The following properties should hold upon exit:**
  
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (**basicProps:callable/1**).
97.3 Documentation on multifiles (freeze)

$combine_attr/4:
No further documentation available for this predicate.
The predicate is `multifile`.

$check_attr/3:
No further documentation available for this predicate.
The predicate is `multifile`.
98 Delaying predicates (when)

Author(s): Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.7#3 (2000/7/21, 11:54:59 CEST)

when/2 delays a predicate until some condition in its variable is met. For example, we may want to find out the maximum of two numbers, but we are not sure when they will be instantiated. We can write the standard max/3 predicate (but changing its name to gmax/3 to denote that the first and second arguments must be ground) as

\[
gmax(X, Y, X) :- X > Y, !.
gmax(X, Y, Y) :- X =< Y.
\]

and then define a 'safe' max/3 as

\[
max(X, Y, Z) :-
  when((ground(X), ground(Y)), gmax(X, Y, Z)).
\]

which can be called as follows:

?- max(X, Y, Z), Y = 0, X = 8.

X = 8,
Y = 0,
Z = 8 ?

yes

Alternatively, max/3 could have been defined as

\[
max(X, Y, Z) :-
  when(ground((X, Y)), gmax(X, Y, Z)).
\]

with the same effects as above. More complex implementations are possible. Look, for example, at the max.pl implementation under the when library directory, where a max/3 predicate is implemented which waits on all the arguments until there is enough information to determine their values:

?- use_module(library('when/max')).

yes

?- max(X, Y, Z), Z = 5, Y = 4.

X = 5,
Y = 4,
Z = 5 ?

yes
98.1 Usage and interface (when)

- **Library usage:**
  
  ```prolog
  :- use_module(library(when)).
  ```

- **Exports:**
  - **Predicates:**
    
    when/2.
  
  - **Regular Types:**
    
    wakeup_exp/1.
  
  - **Multifiles:**
    
    $combine_attr/4, $check_attr/3.

- **Other modules used:**
  
  - **System library modules:**
    
    mattr_global/mattr_global_code, terms_vars, sort, sets.

98.2 Documentation on exports (when)

**when/2:**

*Meta-predicate* with arguments: when(? , goal).

**Usage:** when(WakeupCond, Goal)

- **Description:** Delays / executes Goal according to WakeupCond given. The WakeupConds now acceptable are ground(T) (Goal is delayed until T is ground), nonvar(T) (Goal is delayed until T is not a variable), and conjunctions and disjunctions of conditions:
  
  ```prolog
  wakeup_exp(ground(_1)).
  wakeup_exp(nonvar(_1)).
  wakeup_exp((C1,C2)) :-
    wakeup_exp(C1),
    wakeup_exp(C2).
  ```

when/2 only fails it the WakeupCond is not legally formed. If WakeupCond is met at the time of the call no delay mechanism is involved — but there exists a time penalty in the condition checking.

In case that an instantiation-res the execution of several predicates, the order in which these are executed is not defined.

- **The following properties should hold at call time:**
  
  WakeupCond is a legal expression for delaying goals. (when: wakeup_exp/1)
  Goal is a term which represents a goal, i.e., an atom or a structure. (basic_props: callable/1)

**wakeup_exp/1:**

**Usage:** wakeup_exp(T)
Description: T is a legal expression for delaying goals.

98.3 Documentation on multifiles (when)

$combine_attr/4:
No further documentation available for this predicate.
The predicate is *multifile*.

$check_attr/3:
No further documentation available for this predicate.
The predicate is *multifile*.

98.4 Known bugs and planned improvements (when)

- Redundant conditions are not removed.
- Floundered goals are not appropriately printed.
99 Active modules (high-level distributed execution)

Author(s): Manuel Hermenegildo, Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#2 (2002/5/23, 17:48:34 CEST)

Active modules [CH95] provide a high-level model of inter-process communication and distributed execution (note that this is also possible using Ciao’s communication and concurrency primitives, such as sockets, concurrent predicates, etc., but at a lower level of abstraction). An active module (or an active object) is an ordinary module to which computational resources are attached, and which resides at a given location on the network. Compiling an active module produces an executable which, when running, acts as a server for a number of predicates: the predicates exported by the module. Predicates exported by an active module can be accessed by a program on the network by simply “using” the module, which then imports such “remote predicates.” The process of “using” an active module does not involve transferring any code, but rather setting up things so that calls in the module using the active module are executed as remote procedure calls to the active module. This occurs in the same way independently of whether the active module and the using module are in the same machine or in different machines across the network.

Except for having to compile it in a special way (see below), an active module is identical from the programmer point of view to an ordinary module. A program using an active module imports it and uses it in the same way as any other module, except that it uses “use_active_module” rather than “use_module” (see below). Also, an active module has an address (network address) which must be known in order to use it. In order to use an active module it is necessary to know its address: different “protocols” are provided for this purpose (see below).

From the implementation point of view, active modules are essentially daemons: executables which are started as independent processes at the operating system level. Communication with active modules is implemented using sockets (thus, the address of an active module is an IP socket address in a particular machine). Requests to execute goals in the module are sent through the socket by remote programs. When such a request arrives, the process running the active module takes it and executes it, returning through the socket the computed answers. These results are then taken and used by the remote processes. Backtracking over such remote calls works as usual and transparently. The only limitation (this may change in the future, but it is currently done for efficiency reasons) is that all alternative answers are precomputed (and cached) upon the first call to an active module and thus an active module should not export a predicate which has an infinite number of answers.

The first thing to do is to select a method whereby the client(s) (the module(s) that will use the active module) can find out in which machine/port (IP address/socket number) the server (i.e., the active module) will be listening once started, i.e., a “protocol” to communicate with the active module. The easiest way to do this is to make use of the rendezvous methods which are provided in the Ciao distribution in the library/actmods directory; currently, tmpbased..., filebased..., and webbased....

The first one is based on saving the IP address and socket number of the server in a file in a predefined directory (generally /tmp, but this can be changed by changing tmpbased_common.pl).

The second one is similar but saves the info in the directory in which the server is started (as <module_name>.addr), or in the directory that a .addr file, if it exists, specifies. The

1 It is also possible to provide active modules via a WWW address. However, we find it more straightforward to simply use socket addresses. In any case, this is generally hidden inside the access method and can be thus made transparent to the user.
clients must be started in the same directory (or have access to a file .addr specifying the same directory). However, they can be started in different machines, provided this directory is shared (e.g., by NFS or Samba), or the file can be moved to an appropriate directory on a different machine—provided the full path is the same.

The third one is based on a name server for active modules. When an active module is started, it communicates its address to the name server. When the client of the active module wants to communicate with it, it asks the name server the active module address. This is all done transparently to the user. The name server must be running when the active module is started (and, of course, when the application using it is executed). The location of the name server for an application must be specified in an application file named webbased_common.pl (see below).

These rendezvous methods are encoded in two modules: a first one, called ...publish.pl, is used by the server to publish its info. The second one, called ...locate.pl, is used by the client(s) to locate the server info. For efficiency, the client methods maintain a cache of addresses, so that the server information only needs to be read from the file system the first time the active module is accessed.

Active modules are compiled using the -a option of the Ciao compiler (this can also be done from the interactive top-level shell using make_actmod/2). For example, issuing the following command:

```
ciaoc -a 'actmods/filebased_publish' simple_server
```

compiles the simple server example that comes with the distribution (in the actmods/example directory). The simple_client_with_main example (in the same directory) can be compiled as usual:

```
ciaoc simple_client_with_main
```

Note that the client uses the actmods package, specifies the rendezvous method by importing library('actmods/filebased_locate'), and explicitly imports the “remote” predicates (implicit imports will not work). Each module using the actmods package should only use one of the rendezvous methods.

Now, if the server is running (e.g., simple_server & in Un*x or double-clicking on it in Win32) when the client is executed it will connect with the server to access the predicate(s) that it imports from it.

A simpler even client simple_client.pl can be loaded into the top level and its predicates called as usual (and they will connect with the server if it is running).

### 99.0.1 Active module name servers

An application using a name server for active modules must have a file named webbased_common.pl that specifies where the name server resides. It must have the URL and the path which corresponds to that URL in the file system of the server machine (the one that hosts the URL) of the file that will hold the name server address.

The current distribution provides a file webbased_common.pl that can be used (after proper setting of its contents) for a server of active modules for a whole installation. Alternatively, particular servers for each application can be set up (see below).

The current distribution also provides a module that can be used as name server by any application. It is in file examples/webbased_server/webbased_server.pl.

To set up a name server edit webbased_common.pl to change its contents appropriately as described above (URL and corresponding complete file path). Then recompile this library module:
ciaoc -c webbased_common

The name server has to be compiled as an active module itself:

    ciaoc -a actmods/webserver_publish webbased_server

It has to be started in the server machine before the application and its active modules are compiled.

Alternatively, you can copy webbased_common.pl and use it to set up name servers for particular applications. Currently, this is a bit complicated. You have to ensure that the name server, the application program, and all its active modules are compiled and executed with the same webbased_common.pl module. One way to do this is to create a subdirectory actmods under the directory of your application, copy webbased_common.pl to it, modify it, and then compile the name server, the application program, and its active modules using a library path that guarantees that your actmods directory is located by the compiler before the standard Ciao library. The same applies for when running all of them if the library loading is dynamic.

One way to do the above is using the -u compiler option. Assume the following file:

```prolog
:- module(paths,[],[]).
:- multifile library_directory/1.
:- dynamic library_directory/1.
:- initialization(asserta_fact( library_directory('/root/path/to/my/particular/application') )).
```

then you have file webbased_common.pl in a subdirectory actmods of the above cited path. You have to compile the name server, the active modules, and the rest of the application with:

    ciaoc -u paths -s ...

to use your particular webbased_common.pl and to make executables statically link libraries. If they are dynamic, then you have to provide for the above library_directory path to be set up upon execution. This can be done, for example, by including module paths into your executables.

Addresses of active modules are saved by the name server in a subdirectory webbased_db of the directory where you start it —see examples/webbased_server/webbased_db/webbased_server). This allows to restart the server right away if it dies (since it saves its state). This directory should be cleaned up regularly of addresses of active modules which are no more active. To do this, stop the server —by killing it (its pid is in PATH/FILE), and restart it after cleaning up the files in the above mentioned directory.

### 99.0.2 Active modules as agents

It is rather easy to turn Ciao active modules into agents for some kind of applications. The directory examples/agents contains a (hopefully) self-explanatory example.
99.1 Usage and interface (actmods)

- **Library usage:**
  
  ```prolog
  :- use_package(actmods).
  or
  :- module(...,...,[actmods]).
  ```

- **New declarations defined:**
  ```prolog
  use_active_module/2.
  ```

99.2 Documentation on new declarations (actmods)

**use_active_module/2:**

**Usage:** :- use_active_module(AModule, Imports).

- **Description:** Specifies that this code imports from the active module defined in AModule the predicates in Imports. The imported predicates must be exported by the active module.

- **The following properties should hold at call time:**
  
  - AModule is a source name.
  - Imports is a list of prednames.

99.3 Known bugs and planned improvements (actmods)

- The package provides no means for security: the accessing application must take care of this.

- It can happen that there is a unique process for an active module serving calls from several different simultaneous executions of the same application. In this case, there might be unwanted interactions (e.g., if the active module has state).

- Applications may fail if the name server or an active module is restarted during execution of the application (since they restart at a different port than the one cached by the application).

- One may want name servers to reside at a fixed and known machine and port number (this is known as a service and is defined in /etc/services in a Unix machine). Currently, the port number changes in each invocation of the server.

- One may want to have one name server dedicated to a single application. Currently, there is no easy way to do this.
100 Breadth-first execution

Author(s): Daniel Cabeza, Manuel Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.5#143 (2000/5/12, 13:54:34 CEST)

This package implements breadth-first execution of predicates. Predicates written with operators ‘<-/1 (facts) and ‘<-/2 (clauses) are executed using breadth-first search. This may be useful in search problems when a complete proof procedure is needed. An example of code would be:

```prolog
:- module(chain, _, [bf]).

test(bf) :- bfchain(a,d).
test(df) :- chain(a,d). % loops!

bfchain(X,X) <- .
bfchain(X,Y) <- arc(X,Z), bfchain(Z,Y).

chain(X,X).
chain(X,Y) :- arc(X,Z), chain(Z,Y).

arc(a,b).
arc(a,d).
arc(b,c).
arc(c,a).
```

There is another version, called bf/af, which ensures AND-fairness by goal shuffling. This version correctly says “no” executing the following test:

```prolog
:- module(sublistapp, [test/0,sublistapp/2], [‘bf/af’]).
test :- sublistapp([a],[b]).

sublistapp(S,L) <- append(_,S,Y), append(Y,_,L).

append([], L, L) <- .
append([X|Xs], L, [X|Ys]) <- append(Xs, L, Ys).
```

100.1 Usage and interface (bf)

- Library usage:
  ```prolog
  :- use_package(bf).
  or
  :- module(...,...,[bf]).
  ```
- New operators defined:
  ```prolog
  <=/2 [1200,xf,x], <=/1 [1200,xf].
  ```
100.2 Known bugs and planned improvements (bf)

- Does not correctly work in user files.
101 Iterative-deepening execution

Author(s): Claudio Vaucheret, Manuel Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#119 (2001/8/28, 15:39:1 CEST)

This package applies a compiling control technique to implement depth-first iterative deepening execution [Kor85]. It changes the usual depth-first computation rule by iterative-deepening on those predicates specifically marked. This is very useful in search problems when a complete proof procedure is needed.

When this computation rule is used, first all goals are expanded only up to a given depth. If no solution is found or more solutions are needed by backtracking, the depth limit is incremented and the whole goal is repeated. Although it might seem that this approach is very inefficient because all higher levels are repeated for the deeper ones, it has been shown that is performs only about b/(b - 1) times as many operations than the corresponding breadth-first search, (where b is the branching factor of the proof tree) while the waste of memory is the same as depth first.

The usage is by means of the following directive:

```
:- iterative(Name, FirstCut, Formula).
```

which states than the predicate 'Name' given in functor/arity form will be executed using iterative deepening rule starting at the depth 'FirstCut' with depth being incremented by the predicate 'Formula'. This predicate compute the new depth using the previous one. It must implement a dilating function i.e. the new depth must be greater. For example, to start with depth 5 and increment by 10 you can write:

```
:- iterative(p/1,5,f).
f(X,Y) :- Y is X + 10.
```
or if you prefer,

```
:- iterative(p/1,5,( _(X,Y):- Y is X + 10)).
```

You can also use a fourth parameter to set a limiting depth. All goals below the given depth limit simply fail. Thus, with the following directive:

```
:- iterative(p/1,5,(_(X,Y):- Y is X + 10),100).
```

all goals deeper than 100 will fail.

An example of code using this package would be:

```
:- module(example_id, _,[id]).

test(id) :-
idchain(a,d).
test(df) :-
chain(a,d). % loops!

:- iterative(idchain/2, 3, ( _(X,Z) :- Z is X + 1 ) ).
```

```
idchain(X,X).
idchain(X,Y) :-
arc(X,Z),
idchain(Z,Y).
```

```
chain(X,X).
chain(X,Y) :-
ar(X,Z),
chain(Z,Y).
```
arc(a,b).
arc(a,d).
arc(b,c).
arc(c,a).

The order of solutions are first the shallower and then the deeper. Solutions which are between two cutoff are given in the usual left to right order. For example,

% OLD EXAMPLE - Won't work!
:- module(_, _, [det_hook]).
:- use_module(engine(internals)).

pr(I, X, Y) :-
    display(open(I)), nl,
    '$metachoice'(C),
    det_try(pr2(X, Y, C),
     (display(close_done(I)), nl),
     (display(close_abort(I)), nl)).

pr2(X, _, _) :- X =< 0, !, fail.
pr2(2, 2, C) :- '$metacut'(C).
pr2(X, Y, C) :- (X = Y ; X1 is X - 1, pr2(X1, Y, C)).

test1 :-
    pr(x, 4, X),
    pr(y, 4, Y),
    display(X),
    display(Y),
    nl, X = 1,
    Y = 3, !.

It is possible to preserve the iterative-deepening behavior for calls to predicates defined in other modules. These modules should obviously also use this package. In addition all predicates from such modules should imported, i.e., the directive :- use_module(module), should be used in this case instead of :- use_module(module,[...]). Otherwise calls to predicates outside the module will be treated in the usual way i.e. by depth-first computation.

Another complete proof procedure implemented is the bf package (breadth first execution).

101.1 Usage and interface (id)

- Library usage:
  :- use_package(id).
  or
  :- module(...,...,[id]).
102 Constraint programming over rationals

Author(s): Christian Holzbaur, Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#184 (2003/12/5, 14:7:53 CET)
Note: This package is currently being adapted to the new characteristics of the Ciao module system. This new version now works right now to some extent, but it is under further development at the moment. Use with (lots of) caution.

102.1 Usage and interface (clpq)

- Library usage:
  
  :- use_package(clpq).
  or
  :-module(...,...,[clpq]).

102.2 Other information (clpq)

102.2.1 Some CLP(Q) examples
(Other examples can be found in the source and library directories.)

- 'Reversible' Fibonacci (clpq):
  
  :- module(_, [fib/2], []).
  :- use_package(clpq).
  fib(X,Y):- X .-. 0, Y .-. 0.
  fib(X,Y):- X .-. 1, Y .-. 1.
  fib(N,F) :-
      N .>. 1,
      N1 .-. N - 1,
      N2 .-. N - 2,
      fib(N1, F1),
      fib(N2, F2),
      F .-. F1+F2.

- Matrix multiplication (clpq):
  
  :- use_package(clpq).
  :- use_module(library(write)).
  mmultiply([],_,[]).
  mmultiply([V0|Rest], V1, [Result|Others]):-
      mmultiply(Rest, V1, Others),
      multiply(V1,V0,Result).
  multiply([],_,[]).
multiply([V0|Rest], V1, [Result|Others]):-
multiply(Rest, V1, Others),
vmul(V0,V1,Result).

vmul([],[],0).
vmul([H1|T1], [H2|T2], Result):-
vmul(T1,T2, Newresult),
Result =. H1*H2+Newresult.

matrix(1,[[1,2,3,4,5], [4,0,-1,5,6], [7,1,-2,8,9], [-1,0,1,3,2], [1,5,-3,2,4]]).
matrix(2,[[3,2,1,0,-1], [-2,1,3,0,2], [1,2,0,-1,5], [1,3,2,4,5], [-5,1,4,2,2]]).

% Call with: go(M).
go(M):-
  matrix(1,M1),
  matrix(2,M2),
  mmultiply(M1,M, M2).

Queens (clpq):
:- use_package(clpq).
queens(N, Qs) :- constrain_values(N, N, Qs), place_queens(N, Qs).
constrain_values(0, _N, []).
constrain_values(N, Range, [X|Xs]) :-
  N >. 0, X >. 0, X =<. Range,
  N1 =. N - 1,
  constrain_values(N1, Range, Xs), no_attack(Xs, X, 1).
no_attack([], _Queen, _Nb).
no_attack([Y|Ys], Queen, Nb) :-
  Queen =<. Y+Nb,
  Queen =<. Y-Nb,
  Nb1 =. Nb + 1,
  no_attack(Ys, Queen, Nb1).
place_queens(0, _).
place_queens(N, Q) :-
  N > 0, member(N, Q), N1 is N-1, place_queens(N1, Q).

102.3 Known bugs and planned improvements (clpq)
- clp(Q) and clp(R) cannot be used simultaneously in the same program, or even within the same toplevel session.
103 Constraint programming over reals

Author(s): Christian Holzbaur, Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#103 (2003/12/22, 16:56:50 CET)

Note: This package is currently being adapted to the new characteristics of the Ciao module system. This new version now works right now to some extent, but it under further development at the moment. Use with (lots of) caution.

103.1 Usage and interface (clpr)

- Library usage:
  :- use_package(clpr).
  or
  :- module(...,...,[clpr]).

103.2 Other information (clpr)

103.2.1 Some CLP(R) examples

(Other examples can be found in the source and library directories.)

- 'Reversible' Fibonacci (clpr):

  :- module(_, [fib/2], []).
  :- use_package(clpr).

  fib(X,Y):- X .=. 0, Y .=. 0.
  fib(X,Y):- X .=. 1, Y .=. 1.
  fib(N,F) :-
    N .>. 1,
    N1 .=. N - 1,
    N2 .=. N - 2,
    fib(N1, F1),
    fib(N2, F2),
    F .=. F1+F2.

- Dirichlet problem for Laplace’s equation (clpr):

  
  %
  % Solve the Dirichlet problem for Laplace’s equation using
  % Leibman’s five-point finit-differenc approximation.
  % The goal ?- go1 is a normal example, while the goal ?- go2
  % shows output constraints for a small region where the boundary conditions
  % are not specified.
  %
:- use_package(clpq).
:- use_module(library(format)).

laplace([_, _]).
laplace([H1, H2, H3|T]):-
    laplace_vec(H1, H2, H3),
    laplace([H2, H3|T]).

laplace_vec([_, _], [_, _], [_, _]).
laplace_vec([_TL, T, TR|T1], [ML, M, MR|T2], [BL, B, BR|T3]):-
    B + T + ML + MR - 4 * M =. 0,
    laplace_vec([T, TR|T1], [M, MR|T2], [B, BR|T3]).

printmat([]).
printmat([H|T]):-
    printvec(H),
    printmat(T).

printvec([]):- nl.
printvec([H|T]):-
    printrat(H),
    printvec(T).

printrat(rat(N,D)) :- !,
    X is N/D,
    format(" ~2f",X).
printrat(N) :-
    X is N*100,
    format(" ~2d",X).

go1:-
    X = [
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [100, _, _, _, _, _, _, _, _, _, 100],
        [100, _, _, _, _, _, _, _, _, _, 100],
        [100, _, _, _, _, _, _, _, _, _, 100],
        [100, _, _, _, _, _, _, _, _, _, 100],
        [100, _, _, _, _, _, _, _, _, _, 100],
        [100, _, _, _, _, _, _, _, _, _, 100],
        [100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
    ],
    laplace(X),
    printmat(X).

% Answer:
% 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
% 100.00 51.11 32.52 24.56 21.11 20.12 21.11 24.56 32.52 51.11 100.00
% 100.00 71.91 54.41 44.63 39.74 38.26 39.74 44.63 54.41 71.91 100.00
% 100.00 82.12 68.59 59.80 54.97 54.97 59.80 68.59 82.12 100.00
% 100.00 87.97 78.03 71.00 66.90 65.56 66.90 71.00 78.03 87.97 100.00
% 100.00 91.71 84.58 79.28 75.00 76.07 75.00 79.28 84.58 91.71 100.00
% 100.00 94.30 89.29 85.47 83.10 82.30 83.10 85.47 89.29 94.30 100.00
% 100.00 96.20 92.82 90.20 88.56 88.00 88.56 90.20 92.82 96.20 100.00
% 100.00 97.67 95.59 93.96 92.93 92.58 92.93 93.96 95.59 97.67 100.00
% 100.00 98.89 97.90 97.12 96.63 96.46 96.63 97.12 97.90 98.89 100.00
% 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00

\begin{verbatim}
go2([B31, M32, M33, B34, B42, B43, B12, B13, B21, M22, M23, B24]) :-
    laplace([%
        [B11, B12, B13, _B14],%
        [B21, M22, M23, B24],%
        [B31, M32, M33, B34],%
        [_B41, B42, B43, _B44]
    ]).
\end{verbatim}

% Answer:
% %
% % B34.=. -4*M22+B12+B21+4*M33-B43,
% % M23.=.4*M22-M32-B12-B21,
% % B31.=. -M22+4*M32-M33-B42,
% % B24.=.15*M22-4*M32-4*B12-4*B21-M33-B13 ?

### 103.3 Known bugs and planned improvements (clpr)

- `clp(Q)` and `clp(R)` cannot be used simultaneously in the same program, or even within the same toplevel session.
Chapter 104: Fuzzy Prolog

Author(s): Claudio Vaucheret, Sergio Guadarrama, Francisco Bueno.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#201 (2004/2/26, 15:42:27 CET)

This package implements an extension of prolog to deal with uncertainty. We implement a fuzzy prolog that models interval-valued fuzzy logic. This approach is more general than other fuzzy prologs in two aspects:

1. Truth values are sub-intervals on [0,1]. In fact, it could be a finite union of sub-intervals, as we will see below. Having a unique truth value is a particular case modeled with a unitary interval.

2. Truth values are propagated through the rules by means of a set of aggregation operators. The definition of an aggregation operator is a generalization that subsumes conjunctive operators (triangular norms as min, prod, etc.), disjunctive operators (triangular co-norms as max, sum, etc.), average operators (averages as arithmetic average, quasi-linear average, etc.) and hybrid operators (combinations of previous operators).

We add uncertainty using CLP(R) instead of implementing a new fuzzy resolution as other fuzzy prologs. In this way, we use the original inference mechanism of Prolog, and we use the constraints and its operations provided by CLP(R) to handle the concept of partial truth. We represent intervals as constrains over real numbers and aggregation operators as operations with constraints.

Each fuzzy predicate has an additional argument which represents its truth value. We use “¬” instead of “:-” to distinguish fuzzy clauses from prolog clauses. In fuzzy clauses, truth values are obtained via an aggregation operator. There is also some syntactic sugar for defining fuzzy predicates with certain membership functions, the fuzzy counterparts of crisp predicates, and the fuzzy negation of a fuzzy predicate.

104.1 Usage and interface (fuzzy)

- Library usage:
  ```prolog
  :- use_package(fuzzy).
  or
  :- module(...,...,[fuzzy]).
  ```

- Exports:
  - Predicates:
    - `#:2, fuzzy_predicate/1, fuzzy/1, fnot/1, :-/2, =>/4`
  - Properties:
    - `fuzzybody/1`
  - Regular Types:
    - `faggregator/1`

- New operators defined:
  ```prolog
  -:2 [1200,xf], -:1 [1200,xf], :=/2 [1200,xf], :=/1 [1200,xf], #:2 [1200,xf], =>/1 [1175,fx], fnot/1 [1150,fx], aggr/1 [1150,fx], ##/2 [1120,xfy], <#/2 [1120,xfy], #>/2 [1120,xfy], fuzzy/1 [1150,fx], fuzzy_predicate/1 [1190,fx], fuzzy_discrete/1 [1190,fx].
  ```

- New declarations defined:
  ```prolog
  aggr/1.
  ```
104.2 Documentation on new declarations (fuzzy)

aggr/1:

Usage: :- aggr(Name).

- Description: Declares Name an aggregator. Its binary definition has to be provided.
  For example:
  
  ```prolog
  :- aggr myaggr.
  ```

  ```prolog
  myaggr(X,Y,Z):- Z .=. X*Y.
  ```

  defines an aggregator identical to prod.

  - The following properties hold at call time:
    Name is an atomic term (an atom or a number).  
    (basic_props:constant/1)

104.3 Documentation on exports (fuzzy)

#:2:

Usage: #:,(Name, Decl)

- Description: Defines fuzzy predicate Name from the declaration Decl.

- The following properties hold upon exit:
  Name is a Name/Arity structure denoting a predicate name:
  ```prolog
  prename(P/A) :-
  atm(P),
  nnegint(A).
  ```

  (basic_props:predname/1)

Decl is one of the following three:

```prolog
fuzzydecl(fuzzy_predicate(_1)).
fuzzydecl(fuzzy(_1)).
fuzzydecl(fnot(_1)).
```

(user(... /fuzzy_doc):fuzzydecl/1)

fuzzy_predicate/1:

Usage: fuzzy_predicate(Domain)

- Description: Defines a fuzzy predicate with piecewise linear continuous membership function. This is given by Domain, which is a list of pairs of domain-truth values, in increasing order and exhaustive. For example:

  ```prolog
  young :# fuzzy_predicate([[0,1),(35,1),(45,0),(120,0)]).
  ```

  defines the predicate:

  ```prolog
  young(X,1):- X .>=. 0, X .<. 35.
  young(X,2):- X .>=. 35, X .<. 45, 10*M .=. 45-X.
  young(X,0):- X .>=. 45, X .=<. 120.
  ```

  - The following properties should hold at call time:
    Domain is a list.  
    (basic_props:list/1)
**fuzzy/1:**

**Usage:** fuzzy(Name)

- **Description:** Defines a fuzzy predicate as the fuzzy counterpart of a crisp predicate Name. For example,

  \[ p_f :\# \text{fuzzy} \ p/2 \]

  defines a new fuzzy predicate \( p_f/2 \) (the last argument is the truth value) with truth value equal to 0 if \( p/2 \) fails and 1 otherwise.

- **The following properties should hold at call time:**
  Name is a Name/Arity structure denoting a predicate name:

  \[
  \text{predname}(P/A) :-
  \text{atm}(P),
  \text{nnegint}(A).
  \]

  \( \text{(basic.props:predname/1)} \)

**fnot/1:**

**Usage:** fnot(Name)

- **Description:** Defines a fuzzy predicate as the fuzzy negation of another fuzzy predicate Name. For example,

  \[ \text{notp}_f :\# \text{fnot} \ p_f/3 \]

  defines the predicate:

  \[
  \text{notp}_f(X,Y,M) :-
  p_f(X,Y,Mp),
  M .\_ = . 1 - Mp.
  \]

- **The following properties should hold at call time:**
  Name is a Name/Arity structure denoting a predicate name:

  \[
  \text{predname}(P/A) :-
  \text{atm}(P),
  \text{nnegint}(A).
  \]

  \( \text{(basic.props:predname/1)} \)

**:~/2:**

**Usage:** :~(Head, Body)

- **Description:** Defines a fuzzy clause for a fuzzy predicate. The clause contains calls to either fuzzy or crisp predicates. Calls to crisp predicates are automatically fuzzified. The last argument of Head is the truth value of the clause, which is obtained as the aggregation of the truth values of the body goals. An example:

  \[
  :- \text{module(young2,_,[fuzzy])}.
  
  \text{young_couple}(X,Y,Mu) :~ \text{min}
  \text{age}(X,X1),
  \text{age}(Y,Y1),
  \text{young}(X1,MuX),
  \text{young}(Y1,MuY).
  
  \text{age}(\text{john},37).
  \]
age(rose,39).

young :# fuzzy_predicate([[0,1),(35,1),(45,0),(120,0)]).

so that:

?- young_couple(john,rose,M).

\[ M = 0.6 \]

The following properties should hold at call time:

- **Head** is a term which represents a goal, i.e., an atom or a structure. \(\text{(basic_props:callable/1)}\)
- **Body** is a clause body plus an optional aggregation operator. \(\text{(user(.../fuzzy_doc):fuzzybody/1)}\)

**fuzzybody/1:**

A clause body, optionally prefixed by the name of an aggregation operator. The aggregators currently provided are listed under `faggregator/1`. By default, the aggregator used is `min`.

**Usage:** `fuzzybody(B)`

- **Description:** `B` is a clause body plus an optional aggregation operator.

**faggregator/1:**

The first three are, respectively, the T-norms: minimum, product, and Lukasiewicz’s. The last three are their corresponding T-conorms. Aggregators can be defined by the user, see `aggr/1`.

\[
\begin{align*}
\text{faggregator(min).} \\
\text{faggregator(prod).} \\
\text{faggregator(luka).} \\
\text{faggregator(max).} \\
\text{faggregator(dprod).} \\
\text{faggregator(dluka).}
\end{align*}
\]

**Usage:** `faggregator(Aggr)`

- **Description:** `Aggr` is an aggregator which is cumulative, i.e., its application to several values by iterating pairwise the binary operation is safe.

\[=\rightarrow/4: \]

**Usage:** `=\rightarrow(Aggr, A, B, Truth)`

- **Description:** The fuzzy implication \(A \implies B\) defined by aggregator `Aggr`, resulting in the truth value `Truth`.

- The following properties should hold at call time:

  - **Aggr** is an aggregator which is cumulative, i.e., its application to several values by iterating pairwise the binary operation is safe. \(\text{(user(.../fuzzy_doc):faggregator/1)}\)
  - **A** is a term which represents a goal, i.e., an atom or a structure. \(\text{(basic_props:callable/1)}\)
B is a term which represents a goal, i.e., an atom or a structure.  

Truth is a free variable.

104.4 Other information (fuzzy)

An example program:

```prolog
:- module(dicesum5,_,[fuzzy]).
% this example tries to measure which is the possibility
% that a couple of values, obtained throwing two loaded dice, sum 5. Let
% us suppose we only know that one die is loaded to obtain a small value
% and the other is loaded to obtain a large value.
% % the query is   ? sum(5,M)
% %
small :# fuzzy_predicate([[(1,1),(2,1),(3,0.7),(4,0.3),(5,0),(6,0)]]).
large :# fuzzy_predicate([[(1,0),(2,0),(3,0.3),(4,0.7),(5,1),(6,1)]]).
die1(X,M) :~
  small(X,M).
die2(X,M) :~
  large(X,M).
two_dice(X,Y,M):~ prod
die1(X,M1),
die2(Y,M2).
sum(2,M) :~
two_dice(1,1,M1).
sum(5,M) :~ dprod
two_dice(4,1,M1),
two_dice(1,4,M2),
two_dice(3,2,M3),
two_dice(2,3,M4).
```

There are more examples in the subdirectory `fuzzy/examples` of the distribution.

104.5 Known bugs and planned improvements (fuzzy)

- General aggregations defined by users.
- Inconsistent behaviour of meta-calls in fuzzy clauses.
- Some meta-predicate constructions need be added, specially for 'disjunctive' fuzzy clauses, e.g., `sum/2` in the dice example.
105 Object oriented programming

Author(s): Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.3#63 (1999/9/29, 19:54:17 MEST)

O’Ciao is a set of libraries which allows object-oriented programming in Ciao Prolog. It extends the Ciao Prolog module system by introducing two new concepts:

- Inheritance.
- Instantiation.

Polymorphism is the third fundamental concept provided by object oriented programming. This concept is not mentioned here since traditional PROLOG systems are polymorphic by nature.

Classes are declared in the same way as modules. However, they may be enriched with inheritance declarations and other object-oriented constructs. For an overview of the fundamentals of O’Ciao, see http://www.clip.dia.fi.upm.es/~clip/papers/ociao-tr.ps.gz. However, we will introduce the concepts in a tutorial way via examples.

105.1 Early examples

The following one is a very simple example which declares a class — a simple stack. Note that if you replace class/1 declaration with a module/1 declaration, it will compile correctly, and can be used as a normal Prolog module.

```prolog
%%----------------------------------------------%%
%% A class for stacks. %%
%%----------------------------------------------%%

%% Class declaration: the current source defines a class.
:- class(stack,[],[]).

%% State declaration: storage/1 is an attribute.
:- dynamic storage/1.

%% Interface declaration: the following predicates will
%% be available at run-time.
:- export(push/1).
:- export(pop/1).
:- export(top/1).
:- export(is_empty/0).

% Methods

push(Item) :-
    nonvar(Item),
    asserta_fact(storage(Item)).

pop(Item) :-
    var(Item),
    retract_fact(storage(Item)).

top(Top) :-
```

```prolog```
storage(Top), !.

is_empty :-
    storage(_), !, fail.
    is_empty.

If we load this code at the Ciao toplevel shell:
?- use_package(objects).
yes
?- use_class(library('class/examples/stack')).
yes
?- we can create two stack instances:
?- St1 new stack, St2 new stack.
   St1 = stack('9254074093385163'),
   St2 = stack('9254074091') ? ,
and then, we can operate on them separately:
1 ?- St1:push(8), St2:push(9).
   St1 = stack('9254074093385163'),
   St2 = stack('9254074091') ?
yes
1 ?- St1:top(I), St2:top(K).
   I = 8,  
   K = 9,  
   St1 = stack('9254074093385163'),
   St2 = stack('9254074091') ?
yes
1 ?-

The interesting point is that there are two stacks. If the previous example had been a normal module, we would have a stack, but only one stack.

The next example introduces the concepts of inheritable predicate, constructor, destructor and virtual method. Refer to the following sections for further explanation.

%%----------------------------------------------%%
%% A generic class for item storage.          %%
%%----------------------------------------------%%
:- class(generic).

% Public interface declaration: 
:- export([set/1, get/1, callme/0]).

% An attribute 
:- data datum/1.
% Inheritance declaration: datum/1 will be available to
% descendant classes (if any).
:- inheritable(datum/1).

% Attribute initialization: attributes are easily initialized
% by writing clauses for them.
datum(none).

% Methods
set(X) :-
    type_check(X),
    set_fact(datum(X)).

get(X) :-
    datum(X).

callme :-
    a_virtual(IMPL),
    display(IMPL),
    display(' implementation of a_virtual/0 '),
    nl.

% Constructor: in this case, every time an instance
% of this class is created, it will display a message.
generic :-
    display(' generic class constructor '),
    nl.

% Destructor: analogous to the previous constructor,
% it will display a message every time an instance
% of this class is eliminated.
destructor :-
    display(' generic class destructor '),
    nl.

% Predicates:
% cannot be called as messages (X:method)

% Virtual declaration: tells the system to use the most
% descendant implementation of a_virtual/1 when calling
% it from inside this code (see callme/0).
% If there is no descendant implementation for it,
% the one defined bellow will be used.
:- virtual a_virtual/1.
a_virtual(generic).

:- virtual type_check/1.
type_check(X) :-
nonvar(X).

And the following example, is an extension of previous class. This is performed by establishing an inheritance relationship:

```
%%----------------------------------------------%%
%% This class provides additional functionality %%
%% to the "generic" class. %%
%%----------------------------------------------%%
:- class(specific).

% Establish an inheritance relationship with class "generic".
:- inherit_class(library('class/examples/generic')).

% Override inherited datum/1.
% datum/1 is said to be overridden because there are both an
% inherited definition (from class "generic") and a local one,
% which overrides the one inherited.
:- data datum/1.
:- inheritable datum/1.

% Extend the public interface inherited from "generic".
% note that set/1 and a_virtual/0 are also overridden.
% undo/0 is a new functionality added.
:- export([set/1,undo/0]).

% Methods
set(Value) :-
    inherited datum(OldValue), !,
    inherited set(Value),
    asserta_fact(datum(OldValue)).
set(Value) :-
    inherited set(Value).

undo :-
    retract_fact(datum(Last)), !,
    asserta_fact(inherited(datum(Last))).
undo :-
    retractall_fact(inherited(datum(_))).

% Constructor
specific :-
    generic,
    retractall_fact(inherited(datum(_))),
    display(' specific class constructor '),
    nl.

% Destructor
destructor :-
    display(' specific class destructor '),
% Predicates

% New implementation of a_virtual/1.
% Since this predicate was declared virtual, the
% implementation below will be called from the inherited
% method callme/0 instead of the version defined at "generic".
a_virtual(specific).

Additional examples may be found on the library/class/examples directory relative to your Ciao Prolog instalation.

105.2 Recommendations on when to use objects

We would like to give some advice in the use of object oriented programming, in conjunction with the declarative paradigm.

You should reconsider using O'Ciao in the following cases:

- The pretended "objects" have no state, i.e., no data or dynamic predicates. In this case, a normal module will suffice.
- There is state, but there will be only one instance of a pretended class. Again, a module suffices.
- The "objects" are data structures (list, trees, etc) already supported by Prolog. However, it does make sense to model, using objects, data structures whose change implies a side-effect such as drawing a particular window on the screen.

We recommend the usage of O'Ciao in the following cases:

- You feel you will need to have several copies of a "module".
- Local copies of a module are needed instead of a global module being modified by several ones.
- The "classes" are a representation of external entities to Prolog. For example: the X-Window system.
- There is state or code outside the Prolog system which needs to be manipulated. For example: interfaces to Java or Tcl/Tk code.
- You are not familiar with Prolog, but you know about object oriented programming. O'Ciao may be used as a learning tool to introduce yourself on the declarative programming paradigm.

105.3 Limitations on object usage

O'Ciao run-time speed is limited by the usage of meta-programming structures, for instance: $X = \text{Object:myMethod(25)}, \text{call}(X)$. O'Ciao will optimize static manipulation of objects (those that can be determined at compile time).
106 Declaring classes and interfaces

Author(s): Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#162 (2001/12/4, 16:2:58 CET)

O’Ciao classes are declared in the same way as traditional prolog modules. The general mechanism of source expansion will translate object-oriented declarations to normal prolog code. This is done transparently to the user.

Abstract interfaces are restricted classes which declare exported predicates with no implementation. The implementation itself will be provided by some class using an implements/1 declaration. Only export/1 and data/1 declarations are allowed when declaring an interface. Normal classes may treated as interfaces just ignoring all exported predicate implementations.

106.1 Usage and interface (class)

- Library usage:
  To declare a class the compiler must be told to use the class source expansion. To do so, source code must start with a module declaration which loads the class package:
  
  ```prolog
  :- class(ClassName).
  ```
  
or a module/3 declaration, as follows:
  
  ```prolog
  :- module(ClassName,[],[class]).
  ```

  Interfaces are declared in a similar way:
  
  ```prolog
  :- interface(InterfaceName).
  ```

  Please, do not use SICStus-like module declaration, with a non-empty export list. In other case, some non-sense errors will be reported by normal Ciao module system.

  Most of the regular Ciao declarations may be used when defining a class, such as concurrent/1, dynamic/1, discontiguous/1, multihile/1, and so on.

  However, there are some restrictions wich apply to those declarations:

  - **meta_predicate/1** declaration is not allowed to hold addmodule and pred(N) meta-arguments, except for previously declared multihiles.
  - Attribute and multihile predicates must be declared before any clause of the related predicate.
  - There is no sense in declaring an attribute as meta_predicate.

  It is a good practique to put all your declarations at the very begining of the file, just before the code itself.

- **Exports:**
  - **Predicates:**
    inherited/1, self/1, constructor/0, destructor/0.

- **New declarations defined:**
  export/1, public/1, inheritable/1, data/1, dynamic/1, concurrent/1, inherit_class/1, implements/1, virtual/1.

- **Other modules used:**
  - **System library modules:**
    objects/objects_rt.
106.2 Documentation on new declarations (class)

export/1:  
Declarations a method or attribute to be part of the public interface.  
The public interface is the set of predicates which will be accessible from any code establishing an usage relationship with this class (see use_class/1 for further information).  
Publishing an attribute or method is very similar to exporting a predicate in a Prolog module.  
Whether an inherited and exported predicate is overridden, it must be explicitly exported again.  
An inherited (but not exported) predicate may become exported, without overriding it by the usage of this declaration.  
Usage: :- export(Spec).

Description: Spec will be part of the public (exported) interface.

The following properties should hold at call time:
Spec is a method or attribute specification.  
(objects_rt:method_spec/1)

public/1:  
Just an alias for export/1.  
Usage: :- public(Spec).

Description: This declaration may be used instead of export/1.

The following properties should hold at call time:
Spec is a method or attribute specification.  
(objects_rt:method_spec/1)

inheritable/1:  
Declares a method or attribute to be inherited by descendant classes. Notice that all public predicates are inheritable by default. There is no need to mark them as inheritable.  
Traditionally, object oriented languages makes use of the protected concept. Inheritable/1 may be used as the same concept.  
The set of inheritable predicates is called the inheritable interface.  
Usage: :- inheritable(MethodSpec).

Description: MethodSpec is accessible to descendant classes.

The following properties should hold at call time:
MethodSpec is a method or attribute specification.  
(objects_rt:method_spec/1)

data/1:  
Declares an attribute at current class. Attributes are used to build the internal state of instances. So, each instance will own a particular copy of those attribute definitions. In this way, one instance may have different state from another.  
O’Ciao attributes are restricted to hold simple facts. It is not possible to hold a Head :- Body clause at an instance attribute.  
Notice that attributes are multi-evaluated by nature, and may be manipulated by the habitual assert/retract family of predicates.
Attributes may also be initialized. In order to do so, simply put some clauses after the attribute definition. Each time an instance is created, its initial state will be built from those initialization clauses.

Note: whether a data/1 declaration appears inside an interface, it will be automatically exported.

Usage: :- data Spec.

- Description: Spec is an attribute.
- The following properties should hold at call time:
  Spec is a method or attribute specification. (objects_rt:method_spec/1)

**dynamic/1:**

Just an alias for data/1.

Usage: :- dynamic Spec.

- Description: You may use this declaration instead of data/1.
- The following properties should hold at call time:
  Spec is a method or attribute specification. (objects_rt:method_spec/1)

**concurrent/1:**

Declares a concurrent attribute at current class. Concurrent attributes are just the same as normal attributes, those declared using data/1, except for they may freeze the calling thread instead of failing when no more choice points are remaining on the concurrent attribute.

In order to get more information about concurrent behavior take a look to the concurrent/1 built-in declaration on Ciao Prolog module system.

Usage: :- concurrent Spec.

- Description: Declares Spec to be a concurrent attribute.
- The following properties should hold at call time:
  Spec is a method or attribute specification. (objects_rt:method_spec/1)

**inherit_class/1:**

Makes any public and/or inheritable predicate at inherited class to become accessible by any instance derived from current class.

Inherited class is also called the super class.

Only one inherit_class/1 declaration is allowed to be present at current source.

Notice that inheritance is public by default. Any public and/or inheritable declaration will remain the same to descendant classes. However, any inherited predicate may be overridden (redefined).

A predicate is said to be overridden when it has been inherited from super class, but there are clauses (or a data/1 declaration) present at current class for such a predicate.

Whether a public predicate is overridden, the local definition must also be exported, otherwise an error is reported.

Whether an inheritable predicate (not public) is overridden, the local definition must also be marked as inheritable or exported, otherwise an error is also reported.

Note: whether inherit_class/1 appears inside an interface, it will be used as an implements/1 declaration.

Usage: :- inherit_class(Source).
- **Description:** Establish an *inheritance relationship* between current class and the class defined at *Source* file.

- **The following properties should hold at call time:**
  
  *Source* is a valid path to a prolog file containing a class declaration (without .pl extension).

### implements/1:

**DECLARATION**

Forces current source to provide an implementation for the given interface file. Such interface file may declare another class or a specific interface.

Every public predicate present at given interface file will be automatically declared as public at current source, so you **must** provide an implementation for such predicates.

The effect of this declaration is called *interface inheritance*, and there is no restriction on the number of implements/1 declarations present at current code.

**Usage:** :- implements(Interface).

- **Description:** Current source is supposed to provide an implementation for *Interface*.

- **The following properties should hold at call time:**
  
  *Interface* is a valid path to a prolog file containing a class declaration or an interface declaration (without .pl extension).

### virtual/1:

**DECLARATION**

This declaration may be used whenever descendant classes are to implement different versions of a given predicate.

*virtual* predicates give a chance to handle, in an uniform way, different implementations of the same functionality.

Whether a virtual predicate is declared as a method, there must be at least one clause of it present at current source. Whenever no special implementation is needed at current class, a never-fail/always-fail clause may be defined (depending on your needs). For example:

```
:- virtual([ test1/1 , test2/2 ]).
  test1(_).
  test2(_,_,) :- fail.
```

This kind of virtual methods are also known as *abstract methods*, since implementation is fully delegated to descendant classes.

An attribute may be also declared as a virtual one, but there is no need to write clauses for it.

**Usage:** :- virtual(VirtualMethodSpec).

- **Description:** All calls to *VirtualMethodSpec* predicate in current source will use the most descendant implementation of it.

- **The following properties should hold at call time:**
  
  *VirtualMethodSpec* is a method specification.


106.3 Documentation on exports (class)

**inherited/1:**
This predicate qualifier may be used whenever you need to reference an attribute or method on the super class.

Since methods and attributes may be redefined, this qualifier is need to distinguish between a locally declared predicate and the inherited one, which has the same name.

There is no need to use inherited/1 if a particular inherited predicate has not been redefined at current class.

**Usage:** inherited(Goal)

- **Description:** References a given Goal at the super class
- **The following properties should hold at call time:**
  Goal is a term which represents a goal, i.e., an atom or a structure.  

**self/1:**
Determines which instance is currently executing self/1 goal.

Predicate will fail if argument is not a free variable. Otherwise, it will always succeed, retrieving the instance identifier which is executing current code.

This functionality is very useful since an object must have knowledge of other object’s identifier in order to send messages to it. For example:

```prolog
:- concurrent ack/0.
send_data_to_object(Data,Obj) :- self(X), Obj:take_this(Data,X), current_fact(ack).
acknowledge :- asserta_fact(ack).
take_this(Data,Sender) :- validate_data(Data), Sender:acknowledge.
```

**Usage:** self(Variable)

- **Description:** Retrieves current instance identifier in Variable
- **The following properties should hold at call time:**
  Variable is a free variable.  
- **The following properties should hold upon exit:**
  Variable is an unique term which identifies an object.  

**constructor/0:**
A constructor is a special case of method which complies the following conditions:

- The constructor functor matches the current class name.
- A constructor may hold any number of arguments.
- If an inheritance relationship was defined, an inherited constructor must be manually called (see below).
- When instance creation takes place, any of the declared constructors are implicitly called. The actual constructor called depends on the **new/2** goal specified by the user.

This is a simple example of constructor declaration for the foo class:
foo :-
    display('an instance was born').

Constructor declaration is not mandatory, and there may be more than one constructor declarations (with different arity) at the source code.
This functionality is usefull when some computation is needed at instance creation. For example: opening a socket, clearing the screen, etc.
Whenever an inheritance relationship is established, and there is any constructor defined at the super class, you must call manually an inherited constructor. Here is an example:
    :- class(foo).
    :- inherit_class(myclass).

foo :-
    myclass(0),
    display('an instance was born').

foo(N) :- myclass(N).

Consequences may be unpredictable, if you forget to call an inherited constructor. You should also take care not to call an inherited constructor twice.
All defined constructors are inheritable by default. A constructor may also be declared as public (by the user), but it is not mandatory.
Usage:
  – Description: Constructors are implicitly declared

destructor/0: PREDICATE

A destructor is a special case of method which will be automatically called when instance destruction takes place.
A destructor will never be wanted to be part of the public interface, and there is no need to mark them as inheritable, since all inherited destructors are called by O'Ciao just before yours.
This is a simple example of destructor declaration:
    destructor :-
        display('goodbye, cruel world!!!').

Destructor declaration is not mandatory. Failure or sucess of destructors will be ignored by O'Ciao, and they will be called only once.
This functionality is useful when some computation is need at instance destruction. For example: closing an open file.
Usage:
  – Description: Destructors are implicitly declared

106.4 Other information (class)

This describes the errors reported when declaring a class or an interface. The first section will explain compile-time errors, this is, any semantic error which may be determined at compile time. The second section will explain run-time errors, this is, any exception that may be raisen by the incorrect usage of O'Ciao. Some of those errors may be not reported at compile time, due to the use of meta-programational structures. For example:
    functor(X,my_method,0),call(X).

O'Ciao is not able to check whether my_method/0 is a valid method or not. So, this kind of checking is left to run time.
106.4.1 Class and Interface error reporting at compile time

- **ERROR** : multiple inheritance not allowed.
  There are two or more inherit_class/1 declarations found at your code. Only one declaration
  is allowed, since there is no multiple code inheritance support.

- **ERROR** : invalid inheritance declaration.
  The given parameter to inherit_class/1 declaration is not a valid path to a Prolog source.

- **ERROR** : sorry, addmodule meta-arg is not allowed at F/A.
  You are trying to declare $F/A$ as meta-predicate, and one of the meta-arguments is addmodule. This is not allowed in O'Ciao due to implementation restrictions. For example:
  
  ```prolog
  :- meta_predicate example(addmodule).
  example(X,FromModule) :- call(FromModule:X).
  ```

- **ERROR** : invalid attribute declaration for Arg.
  Argument to data/1 or dynamic/1 declaration is not a valid predicate specification of the form Functor/Arity. For example:
  
  ```prolog
  :- data attr.
  :- dynamic attr(_).
  :- data attr/m.
  etc,etc...
  ```

- **ERROR** : pretended attribute $F/A$ was assumed to be a method.
  You put some clauses of $F/A$ before the corresponding data/1 or dynamic/1 declaration. For example:
  
  ```prolog
  attr(initial_value).
  :- data attr/1.
  ```
  It is a must to declare attributes before any clause of the given predicate.

- **ERROR** : destructor/0 is not allowed to be an attribute.
  There is a :- data(destructor/0) or :- dynamic(destructor/0). declaration in your code. This
  is not allowed since destructor/0 is a reserved predicate, and must be allways a method.

- **ERROR** : Constructor is not allowed to be an attribute.
  As the previos error, you are trying to declare a constructor as an attribute. A constructor
  must allways be a method.

- **ERROR** : invalid multifile: destructor/0 is a reserved predicate.
  There is a :- multifile(destructor/0). declaration in your code. This is not allowed since
  destructor/0 is a reserved predicate, and must be allways a method.

- **ERROR** : invalid multifile: Constructor is a reserved predicate.
  As the previos error, you are trying to declare a constructor as a multifile. Any constructor
  must allways be a method.

- **ERROR** : multifile declaration of $F/A$ ignored: it was assumed to be a method.
  You put some clauses of $F/A$ before the corresponding multifile/1 declaration. For example:
  
  ```prolog
  example(a,b).
  :- multifile example/2.
  ```
  Multifile predicates must be declared before any clause of the given predicate.

- **ERROR** : invalid multifile declaration: multifile(Arg).
  Given argument to multifile/1 declaration is not a valid predicate specification, of the form Functor/Arity.
• **ERROR : invalid public declaration: Arg.**
  Given argument Arg to public/1 or export/1 declaration is not a valid predicate specification, of the form Functor/Arity.

• **ERROR : invalid inheritable declaration: inheritable(Arg).**
  Given argument Arg to inheritable/1 declaration is not a valid predicate specification, of the form Functor/Arity.

• **ERROR : destructor/0 is not allowed to be virtual.**
  There is a :- virtual(destructor/0) declaration present at your code. Destructors and/or constructors are not allowed to be virtual.

• **ERROR : Constructor is not allowed to be virtual.**
  As the previous error, you are trying to declare a constructor as virtual. This is not allowed.

• **ERROR : invalid virtual declaration: virtual(Arg).**
  Given argument to virtual/1 declaration is not a valid predicate specification, of the form Functor/Arity.

• **ERROR : clause of F/A ignored: only facts are allowed as initial state.**
  You declared F/A as an attribute, then you put some clauses of that predicate in the form Head :- Body. For example:
  ```prolog
  :- data my_attribute/1.
  my_attribute(X) :- X>=0 , X=<2.
  ```
  This is not allowed since attributes are assumed to hold simple facts. The correct usage for those initialization clauses is:
  ```prolog
  :- data my_attribute/1.
  my_attribute(0).
  my_attribute(1).
  my_attribute(2).
  ```

• **ERROR : multifile F/A is not allowed to be public.**
  The given F/A predicate is both present at multifile/1 and public/1 declarations. For example:
  ```prolog
  :- public(p/1).
  :- multifile(p/1).
  ```
  This is not allowed since multifile predicates are not related to Object Oriented Programming.

• **ERROR : multifile F/A is not allowed to be inheritable.**
  Analogous to previous error.

• **ERROR : multifile F/A is not allowed to be virtual.**
  Analogous to previous error.

• **ERROR : virtual F/A must be a method or attribute defined at this class.**
  There is a virtual/1 declaration for F/A, but there is not any clause of that predicate nor a data/1 declaration. You must declare at least one clause for every virtual method. Virtual attributes does not require any clause but a data/1 declaration must be present.

• **ERROR : implemented interface Module is not a valid interface.**
  There is an implements/1 declaration present at your code where given Module is not declared as class nor interface.

• **ERROR : predicate F/A is required both as method (at Itf1 interface) and attribute (at Itf2 interface).**
  There is no chance to give a correct implementation for F/A predicate since Itf1 and Itf2 interfaces require different definitions. To avoid this error, you must remove one of the related implements/1 declaration.
- **ERROR : inherited** *Source* must be a class.
  There is an :- inherit_class(*Source*) declaration, but that source was not declared as a class.

- **ERROR : circular inheritance** *Source* is not a valid super-class.
  Establishing an inheritance relationship with *Source* will cause current class to be present twice in the inheritance line. This is not allowed. The cause of this is error is simple: There is some inherited class from *Source* which also establishes an inheritance relationship with current source.

- **ERROR : method/attribute** *F/A* must be implemented.
  Some of the implemented interfaces requires *F/A* to be defined, but there is no definition for such predicate, even an inherited one.

- **ERROR : local implementation of** *F/A* hides inheritable/public definition.
  There is an inherited definition for *F/A* which is been redefined at current class, but there is no valid inheritable/public declaration for the last one. Overridden public predicates must be also declared as public. Overridden inheritable predicates must be declared either as public or inheritable.

- **ERROR : public predicate** *F/A* was not defined nor inherited.
  There is a public/1 declaration for *F/A*, but there is no definition for it at current class nor an inherited one.

- **ERROR : argument to self/1 must be a free variable.**
  Argument to self/1 is not a variable, for example: `self(abc)`.

- **ERROR : unknown inherited attribute in** *Goal*.
  *Goal* belongs to assert/retract family of predicates, and given argument is not a valid inherited attribute. The most probable causes of this error are:
  - The given predicate is defined at super-class, but you forgot to mark it as inheritable (or public), at such class.
  - The given predicate was not defined (at super-class) as an attribute, just as a method.

- **ERROR : unknown inherited goal** *Goal*.
  The given *Goal* was not found at super-class, or it is not accessible. Check whether *Goal* was marked as inheritable (or public) at super-class.

- **ERROR : invalid argument** *F/A* is not an attribute.
  You are trying to pass a method as an argument to any predicate which expect a *fact* predicate.

- **ERROR : unknown inherited fact** *Fact*.
  There is a call to any predicate which expects a *fact* argument (those declared as data or dynamic), but the actual argument is not an inherited attribute. For example:
  ```
  asserta_fact(inherited(not_an_attribute(8)))
  ```
  where `not_an_attribute/1` was not declared as data or dynamic by the super-class (or corresponding ascendant).

- **ERROR : unknown inherited spec** *F/A*.
  There is a reference to an inherited predicate specification, but the involved predicate has not been inherited.

- **WARNING : meta-predicate specification of** *F/A* ignored since this is an attribute.
  You declared *F/A* both as an attribute and a meta-predicate. For example:
  ```
  :- meta_predicate attr(goal).
  :- data attr/1.
  ```
  There is no sense in declaring an attribute as meta-predicate.
• **WARNING : class destructor is public**  
  There is a :- public(destructor/0) declaration present at your code. Marking a destructor as public is a very bad idea since anybody may destroy or corrupt an instance before the proper time.

• **WARNING : class destructor is inheritable**  
  Analogous to previous error.

• **WARNING : There is no call to inherited constructor/s**  
  You have not declared any constructor at your class, but there is any inherited constructor that should be called. Whenever you do not need constructors, but there is an inheritance relationship (where super-class declares a constructor), you should write a simple constructor as the following example:

  ```prolog
  :- class(mycall).
  :- inherit_class(other_class).

  mycall :-
    other_class.
  ```

• **WARNING : multi-arity hides inherited predicate.**  
  You declared as multi-arity a predicate which matches an inherited predicate name. Any reference to the inherited predicate must be done by the ways of the inherited/1 qualifier.

### 106.4.2 Class and Interface error reporting at run time

• **EXCEPTION : error(existence_error(object_goal,Goal),Mod).**  
  Called Goal from module (or class) Mod is unknown or has not been published.

### 106.4.3 Normal Prolog module system interaction

O’Ciao works in conjunction with the Ciao Prolog module system, which also reports its own error messages. This will cause Ciao to report a little criptic error messages due to the general mechanism of source-to-source expansion. Those are some tips you must consider when compiling a class:

• Any error relative to method 'm' with arity A will be reported for predicate 'obj$m'/A+1. For example:

  ```prolog
  WARNING: (lns 28-30) [Item,Itema] - singleton variables in obj$remove/2
  ```
  This error is relative to method remove/1.

• **set_prolog_flag/1 declaration will be usefull when declaring multiple constructors. It will avoid some awful warnings. Example:**

  ```prolog
  :- class(mycall).

  %% Use this declaration whenever several constructors are needed.

  :- set_prolog_flag(multi_arity_warnings,off).

  mycall(_).

  myclass(_,_).

  :- set_prolog_flag(multi_arity_warnings,on).
  ```
106.5 Known bugs and planned improvements (class)

- addmodule and pred(N) meta-arguments are not allowed on meta-predicates.
107 Compile-time usage of objects

Author(s): Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

This package is required to enable user code to create objects and manipulate them, as well as loading any needed class.

107.1 Usage and interface (objects)

- Library usage:
  Any code which needs to use objects must include the objects package:
  ```prolog
  :- module(ModuleName, Exports, [objects]).
  ```
  You can use objects even if your code is a class. Note that declaring a class does not automatically enables the code to create instances.
  ```prolog
  :- class(ModuleName, [], [objects]).
  ```
  This package enables both static and dynamic usage of objects.

- New declarations defined:
  use_class/1, instance_of/2, new/2.

- Other modules used:
  - System library modules:
    objects/objects_rt.

107.2 Documentation on new declarations (objects)

use_class/1: DECLARATION

It establishes an usage relationship between the given file (which is supposed to declare a class) and current source. Usage relationships are needed in order to enable code to create instances of the given class, and to make calls to instances derived from such class.

Since an interface is some kind of class, they may be used within this declaration but only for semantic checking porpouses. Instances will not be derived from interfaces.

use_class/1 is used in the same way as use_module/1.

Usage: :- use_class(ClassSource).

- Description: Establish usage relationship with ClassSource.
- The following properties should hold at call time:
  ClassSource is a valid path to a prolog file containing a class declaration (without .pl extension).

instance_of/2: DECLARATION

Statically declares an identifier to be an instance of a given class.

It may be used as new/2 predicate except for:
• The instance identifier will not be a variable, it must be provided by the user, and must be unique.
• Instance creation will never fail, even if the constructor fails.

For every statically declared object the given constructor will be called at program startup. Those instances may be destroyed manually, but it is not recommended. When reloading the involved class from the Ciao toplevel shell. It may destroy statically declared instances, and create them again.

Statically declared instances must be called using a specifically designed module-qualification: `ClassName(Object):Goal`. For example:

```prolog
:- module(example,[main/0],[objects]).
:- use_class(library(counter)).
:- cnt instance_of counter(10).

main :-
    counter(cnt):decrease(1),
    counter(cnt):current_value(X),
    display(X).
```

But **statically written code** (only) is allowed to use module-style qualifications as a macro:

```prolog
main :-
    cnt:decrease(1),
    cnt:current_value(X),
    display(X).
```

Notice that dynamically expanded goals such as `X=cnt,X:decrease(1)` will not work, use `X=counter(cnt),X:decrease(1)` instead.

**Usage:** `:- instance_of(Object, Constructor).

Description:** Declares `Object` to be an instance of the class denoted by `Constructor`.

**The following properties should hold at call time:**
- `Object` is an unique term which identifies an object. (`objects_rt:instance_id/1`)
- `Constructor` is a term whose functor matches a class name. (`objects_rt:constructor/1`)

### new/2:

This declaration has the same effect as `instance_of/2.`

**Usage:** `:- new(Object, Constructor).

Description:** Just an alias for `instance_of/2`.

**The following properties should hold at call time:**
- `Object` is an unique term which identifies an object. (`objects_rt:instance_id/1`)
- `Constructor` is a term whose functor matches a class name. (`objects_rt:constructor/1`)

### 107.3 Other information (objects)

Compile-time errors are restricted to some local analysis. Since there is no type declaration in the Prolog language, there is no possibility to determine whenever a given variable will hold an instance of any class.
However, little semantic analysis is performed. User may aid to perform such an analysis by the usage of run time checks (which are also detected at compile time), or static declarations. For example:

```
clause(Obj) :- Obj:a_method(334).
```

O’Ciao may be not able to determine whenever `a_method/1` is a valid method for instance `Obj`, unless some help is provided:

```
clause(Obj) :- Obj instance_of myclass, Obj:a_method(334).
```

In such case, O’Ciao will report any semantic error at compile-time.

Most of the run-time errors are related to normal Ciao Prolog module system. Since objects are treated as normal Prolog modules at run time, there is no further documentation here about that stuff.

### 107.3.1 Error reporting at compile time (objects)

- **ERROR : invalid instance identifier `ID`: must be an atom**
  There is a `instance_of/2` or `new/2` declaration where first argument `ID` must be an unique atom, but currently it is not. Statically declared instances needs an identifier to be provided by the user.

- **ERROR : instance identifier `ID` already in use**
  There are two or more `instance_of/2` declarations with the same first argument `ID`. Instance identifiers must be unique.

- **ERROR : invalid use_class/1 declaration: `SourceFile` is not a class**
  Those are the causes for this error:
  - The given `SourceFile` does not exist, or is not accesible.
  - The given `SourceFile` is not a Prolog source.
  - The given `SourceFile` is a valid Prolog source, but it does not declare a class.

- **ERROR : unknown class on `ID` instance declaration**
  The class defined on the `instance_of/2` declaration for `ID` instance has not been loaded by a `use_class/1` declaration.

- **ERROR : instance identifier `ID` is an existing Prolog module**
  There is an statically declared instance whose identifier may cause interference with the Ciao Prolog module system. Use another instance identifier.

- **ERROR : unknown constructor on `ID` instance declaration**
  The given constructor on the `instance_of/2` declaration for `ID` has not been defined at the corresponding class.

- **ERROR : constructor is needed on `ID` instance declaration**
  No constructor was defined on the `instance_of/2` declaration for `ID` and default constructor is not allowed. You must provide a constructor.

- **ERROR : static instance `ID` was derived from a different constructor at `AnotherModule`**
  `ID` has been declared to be an static instance both on `AnotherModule` and current source, but different constructors were used. The most probable causes for this error are:
  - Occasionally, there is another module using the same instance identifier and it was not noticed by you. Another different identifier may be used instead.
  - It was you intention to use the same object as declared by the other module. In this case, the same constructor must be used.

- **ERROR : invalid first argument in call to new(`Arg_`)**
  There is a new/1 goal in your code where first argument is not a free variable. For example:
myobj new myclass
First argument must be a variable in order to receive a run-time generated object identifier.

- **ERROR : unknown class in call to new(?,Constructor)**
  The given Constructor in call to new/2 does not correspond to any used class at current code. The most probable cause of this may be:
  - You forgot to include a use_class/1 declaration in your code.
  - There is a spelling mistake in the constructor. For example:
    ```prolog
    :- use_class(myclass).
    foo(X) :- X new mclass.
    ```
- **ERROR : can not create an instance from an interface: new(?,Constructor)**
  Given Constructor references an interface rather than a class. Instances can not be derived from interface-expanded code.
- **ERROR : unknown constructor in call to new(? Constructor)**
  As the previous error, there is a mistake in the given Constructor. This error is reported when you are trying to call a constructor which was not defined at the corresponding class. Check the class definition to find what is going on.
  Another cause for this error is the incorrect usage of the default constructor. Whenever there are one or more constructors defined at the involved class, you are restricted to chose one of them. This seems that default constructor will be available, if and only if, there are no constructors defined at the involved class.
- **ERROR : call to non-public ID:Goal**
  You are trying to call a method which was not declared as public by the class specified in instance_of/2 declaration for ID.
- **ERROR : call to inaccessible predicate at instance ID:Goal**
  There is a call to Goal at statically declared instance ID which is unknown or was not declared as public.
- **ERROR : unknown instance ID of class Class at Goal**
  There is a call to Goal where involved statically declared instance ID is unknown or is not derived from Class. Check whether it was declared by a instance_of/2 declaration.
- **ERROR : inaccessible attribute Fact at instance ID**
  There is an attempt to use ID:Fact but it was not declared as public.
- **ERROR : unknown attribute Fact at instance ID**
  There is an attempt to use ID:Fact but it is unknown or it is not an attribute (may be a method).
- **WARNING : invalid call to new(?,.)**
  There is a call to new/2 in your code where first argument variable has been determined to hold any other instance. For example:
  ```prolog
  foo :- X new myclass, X new otherclass.
  ```
  or
  ```prolog
  foo(X) :- X instance_of myclass, X new myclass.
  ```
  The related call to new/2 will always fail.
- **WARNING : called Goal is not public at any used class**
  There is a call to Var:Goal where Var has not been determined to be compatible with any class. However, Goal is not public at any class specified by the use_class/1 declaration.
  This is a warning (not an error) since Var:Goal may be not related to Object Oriented Programming.
107.3.2 Error reporting at run time (objects)

- **EXCEPTION : instantiation_error('1st argument must be free variable')**  
  Calling to new/1 requires first argument to be a free variable. For example:  
  \[ X = \text{this\_will\_raise\_an\_exception}, X \text{ new myclass} \].

- **EXCEPTION : instantiation_error('class not given')**  
  You called new/2 using a free variable as second argument.

- **EXCEPTION : instantiation_error(inaccesible_class(Class), from(Module))**  
  Module tried to create an instance of Class by the ways of new/2, but there is no usage relationship between Module and Class.

- **EXCEPTION : instantiation_error(invalid_constructor(Constructor))**  
  Constructor was not defined by the corresponding class.
108 Run time usage of objects

Author(s): Angel Fernandez Pineda, Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#51 (2001/1/25, 21:33:0 CET)

This library provides run-time support for object creation and manipulation. Objects are also called class instances, or simply instances.

Objects in Ciao are treated as normal modules. This is, an object is a run-time generated Prolog module, which may be identified by an unique term across the whole application.

This is a very simple example of how to create an instance, and how to make calls to it:

```
AnObj new myclass,
AnObj:mymethod.
```

In order to make any object accessible from code, an usage relationship must be established between the class (from which instances are derived) and the code itself. Refer to `use_class/1` predicate or `use_class/1` declaration in order to do so.

108.1 Usage and interface (objects_rt)

- **Library usage:**
  This library is automatically loaded when using the `objects` package:
  ```prolog
  :- module(ModuleName, Exports, [objects]).
  ```
  Nothing special needs to be done.

- **Exports:**
  - *Predicates:*
    - `new/2, instance_of/2, derived_from/2, interface/2, instance_codes/2, destroy/1, use_class/1`.
  - *Properties:*
    - `constructor/1, class_name/1, interface_name/1, instance_id/1, class_source/1, interface_source/1, method_spec/1, virtual_method_spec/1`.

- **Other modules used:**
  - *System library modules:*
    - `compiler/compiler, prolog_sys, system`.

108.2 Documentation on exports (objects_rt)

**new/2:**

Dynamic instance creation takes place by the ways of this predicate.

It takes a free variable as first argument which will be instantiated to an internal object identifier.

Second argument must be instantiated to a class constructor. Class constructors are designed to perform an initialization on the new created instance. Notice that instance initialization may involve some kind of computation, not only *state initialization.*
A class constructor is made by a functor, which must match the intended class name, and any number of parameters. For example:

```
Obj new myclass(1500, 'hello, world!!!')
```

Those parameters depends (obviously) on the constructors defined at the class source. If no constructors where defined, no parameters are needed. This is called the default constructor. An example:

```
Obj new myclass
```

The default constructor can not be called if there is any constructor available at the class source.

Instantiation will raise an exception and fail whenever any of this conditions occur:

- First argument is not a free variable.
- Second argument functor is a class, but there is no usage relationship with it.
- Second argument functor is not a class.
- The given constructor is unknown.
- The given constructor fails (notice that default constructor never fails).

Objects may also be statically declared, refer to `instance_of/2` declaration.

**Usage:** `new(InstanceVar, Constructor)`

- **Description:** Creates a new instance of the class specified by `Constructor` returning its identifier in `InstanceVar`.
- **The following properties should hold at call time:**
  - `InstanceVar` is a free variable. (term_typing:var/1)
  - `Constructor` is a term whose functor matches a class name. (objects_rt:constructor/1)
- **The following properties should hold upon exit:**
  - `InstanceVar` is an unique term which identifies an object. (objects_rt:instance_id/1)

**instance_of/2:**

This predicate is used to perform dynamic type checking. You may check whether a particular instance belongs to a particular class or related descendants.

`instance_of/2` is used to perform static semantic analysis over object oriented code constructions.

By the use of `instance_of/2` you may help to perform such analysis.

**Usage 1:** `instance_of(Instance, Class)`

- **Description:** Test whether `Instance` was derived from any descendant of `Class`, or that class itself.
- **The following properties should hold at call time:**
  - `Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)
  - `Class` is an atom denoting a class. (objects_rt:class_name/1)

**Usage 2:** `instance_of(Instance, Class)`

- **Description:** Retrieves, on backtracking, the inheritance line of `Instance` commencing on the creation class (that specified on call to `new/2`) and continuing on the rest of ascendant classes, if any.
The following properties should hold at call time:

- **Instance** is an unique term which identifies an object. \(\text{objects}_{-}\text{rt}::\text{instance}_{-}\text{id}/1\)
- **Class** is a free variable. \(\text{term}_{-}\text{typing}::\text{var}/1\)

The following properties should hold upon exit:

- **Class** is an atom denoting a class. \(\text{objects}_{-}\text{rt}::\text{class}_{-}\text{name}/1\)

---

**derived_from/2:**

Test whether an object identifier was derived directly from a class, by the usage of `new/2` or a static instance declaration \(\text{instance}_{-}\text{of}/2\).

**Usage 1:** `derived_from(Instance, Class)`

- **Description:** Test derivation of `Instance` from `Class`
- **The following properties should hold at call time:**
  - **Instance** is an unique term which identifies an object. \(\text{objects}_{-}\text{rt}::\text{instance}_{-}\text{id}/1\)
  - **Class** is an atom denoting a class. \(\text{objects}_{-}\text{rt}::\text{class}_{-}\text{name}/1\)

**Usage 2:** `derived_from(Instance, Class)`

- **Description:** Retrieves the `Class` responsible of the derivation of `Instance`.
- **The following properties should hold at call time:**
  - **Instance** is an unique term which identifies an object. \(\text{objects}_{-}\text{rt}::\text{instance}_{-}\text{id}/1\)
  - **Class** is a free variable. \(\text{term}_{-}\text{typing}::\text{var}/1\)
- **The following properties should hold upon exit:**
  - **Class** is an atom denoting a class. \(\text{objects}_{-}\text{rt}::\text{class}_{-}\text{name}/1\)

---

**interface/2:**

This predicate is used to ensure a given interface to be implemented by a given instance.

**Usage 1:** `interface(Instance, Interface)`

- **Description:** Check whether `Instance` implements the given `Interface`.
- **The following properties should hold at call time:**
  - **Instance** is an unique term which identifies an object. \(\text{objects}_{-}\text{rt}::\text{instance}_{-}\text{id}/1\)
  - **Interface** is an unique atom which identifies a public interface. \(\text{objects}_{-}\text{rt}::\text{interface}_{-}\text{name}/1\)

**Usage 2:** `interface(Instance, Interfaces)`

- **Description:** Retrieves on backtracking all the implemented `Interfaces` of `Instance`.
- **The following properties should hold at call time:**
  - **Instance** is an unique term which identifies an object. \(\text{objects}_{-}\text{rt}::\text{instance}_{-}\text{id}/1\)
  - **Interfaces** is a free variable. \(\text{term}_{-}\text{typing}::\text{var}/1\)
- **The following properties should hold upon exit:**
  - **Interfaces** is an unique atom which identifies a public interface. \(\text{objects}_{-}\text{rt}::\text{interface}_{-}\text{name}/1\)
instance_codes/2:

Retrieves a character string representation from an object identifier and vice-versa.

Usage 1: instance_codes(Instance, String)
- Description: Retrieves a String representation of given Instance.
- The following properties should hold at call time:
  Instance is an unique term which identifies an object. (objects_rt:instance_id/1)
  String is a free variable. (term_typing:var/1)
- The following properties should hold upon exit:
  String is a string (a list of character codes). (basic_props:string/1)

Usage 2: instance_codes(Instance, String)
- Description: Reproduces an Instance from its String representation. Such an instance must be alive across the application: this predicate will fail whether the involved instance has been destroyed.
- The following properties should hold at call time:
  Instance is a free variable. (term_typing:var/1)
  String is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold upon exit:
  Instance is an unique term which identifies an object. (objects_rt:instance_id/1)

destroy/1:

As well as instances are created, they must be destroyed when no longer needed in order to release system resources.

Unfortunately, current O'Ciao implementation does not support automatic instance destruction, so user must manually call destroy/1 in order to do so.

The programmer must ensure that no other references to the involved object are left in memory when destroy/1 is called. If not, unexpected results may be obtained.

Usage: destroy(Instance)
- Description: Destroys the object identified by Instance.
- The following properties should hold at call time:
  Instance is an unique term which identifies an object. (objects_rt:instance_id/1)

use_class/1:

The behaviour of this predicate is identical to that provided by the declaration of the same name use_class/1. It allows user programs to dynamically load classes. Whether the given source is not a class it will perform a use_module/1 predicate call.

Usage: use_class(ClassSource)
- Description: Dynamically loads the given ClassSource
- The following properties should hold at call time:
  ClassSource is a valid path to a prolog file containing a class declaration (without .pl extension). (objects_rt:class_source/1)
**constructor/1:**

- **Usage:** `constructor(Cons)`
- **Description:** `Cons` is a term whose functor matches a class name.

**class_name/1:**

- **Usage:** `class_name(ClassName)`
- **Description:** `ClassName` is an atom denoting a class.

**interface_name/1:**

- **Usage:** `interface_name(Interface)`
- **Description:** `Interface` is an unique atom which identifies a public interface.

**instance_id/1:**

- **Usage:** `instance_id(ID)`
- **Description:** `ID` is an unique term which identifies an object.

**class_source/1:**

- **Usage:** `class_source(Source)`
- **Description:** `Source` is a valid path to a prolog file containing a class declaration (without .pl extension).

**interface_source/1:**

- **Usage:** `interface_source(Source)`
- **Description:** `Source` is a valid path to a prolog file containing a class declaration or an interface declaration (without .pl extension).

**method_spec/1:**

- **Usage:** `method_spec(Spec)`
- **Description:** `Spec` is a method or attribute specification.

- **virtual_method_spec/1:**
  - **Usage:** `virtual_method_spec(Spec)`
  - **Description:** `Spec` is a method specification.
108.3 Known bugs and planned improvements (objects_rt)

- Usage of objects from the user module does not work properly. It is better to use the objects package in a (proper) module.
- Not really a bug: when loading code which declares static instances from the toplevel shell, predicate use_module/1) will not work properly: those instances may be not correctly created, and predicates will fail whenever they are not supposed to do. This may be avoided by reloading again the involved module, but make sure it is modified and saved to disk before doing so.
109 The Ciao Remote Services Package

Author(s): Manuel Carro.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#104 (2003/12/22, 17:9:19 CET)
Module for The Ciao Remote Services Package

109.1 Usage and interface (remote)

- Library usage:
  :- use_module(library(remote)).
- Other modules used:
  - System library modules:
    remote/ciao_client_rt.

109.2 Documentation on exports (remote)

@/2: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

@/2: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

server_stop/1: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

server_stop/1: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

server_trace/1: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

server_trace/1: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

server_notrace/1: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).

server_notrace/1: (UNDOC_REEXPORT)
   Imported from ciao_client_rt (see the corresponding documentation for details).
109.3 Known bugs and planned improvements (remote)

- Dynamic loading of code not yet implemented.
- :- remote/1 predicate declaration not yet implemented.
- Remote use of modules (http, ftp, ciaotp) not yet implemented.
- Remote creation of objects not yet implemented.
- Code migration not yet implemented (several algorithms possible).
- Evaluation of impact of marshalling and/or attribute encoding not yet done.
- Secure transactions not yet implemented.
PART VIII - Interfaces to other languages and systems

Author(s): The CLIP Group.
The following interfaces to/from Ciao Prolog are documented in this part:

- External interface (e.g., to C).
- Socket interface.
- Tcl/tk interface.
- Web interface (http, html, xml, etc.);
- Persistent predicate databases (interface between the Prolog internal database and the external file system).
- SQL-like database interface (interface between the Prolog internal database and external SQL/ODBC systems).
- Java interface.
- Calling emacs from Prolog.
Chapter 110: Foreign Language Interface

Author(s): Jose Morales, Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#47 (2003/1/7, 14:22:36 CET)

Ciao Prolog includes a high-level, flexible way to interface C and Prolog, based on the use of assertions to declare what are the expected types and modes of the arguments of a Prolog predicate, and which C files contain the corresponding code. To this end, the user provides:

- A set of C files, or a precompiled shared library,
- A Ciao Prolog module defining which predicates are implemented in the C files and the types and modes of their arguments, and
- an (optional) set of flags required for the compilation of the files.

The Ciao Prolog compiler analyzes the Prolog code written by the user and gathers this information in order to generate automatically C "glue" code implementing the data translation between Prolog and C, and to compile the C code into dynamically loadable C object files, which are linked automatically when needed.

110.1 Declaration of Types

Each predicate implemented as a foreign C function must have accompanying declarations in the Ciao Prolog associated file stating the types and modes of the C function. A sample declaration for `prolog_predicate` which is implemented as `foreign_function_name` is:

```prolog
:- true pred prolog_predicate(m1(Arg1), ... mN(ArgN)) ::
   type1 * ... * typeN +
   (foreign(foreign_function_name), returns(ArgR)).
```

where `m1`, ..., `mN` and `type1`, ..., `typeN` are respectively the modes and types of the arguments. `foreign_function_name` is the name of the C function implementing `prolog_predicate/N`, and the result of this function is unified with `ArgR`, which must be one of `Arg1` ... `ArgN`.

This notation can be simplified in several ways. If the name of the foreign function is the same as the name of the Ciao Prolog predicate, `foreign(foreign_function_name)` can be replaced by `foreign/0`. `returns(ArgR)` specifies that the result of the function corresponds to the `ArgR` argument of the Ciao Prolog predicate. If the foreign function does not return anything (or if its value is ignored), then `returns(ArgR)` must be removed. Note that `returns` cannot be used without `foreign`. A simplified, minimal form is thus:

```prolog
:- true pred prolog_predicate(m1(Arg1), ... mN(ArgN)) ::
   type1 * ... * typeN + foreign.
```

110.2 Equivalence between Ciao Prolog and C types

The automatic translation between Ciao Prolog and C types is defined (at the moment) only for some simple but useful types. The translation to be performed is solely defined by the types of the arguments in the Ciao Prolog file (i.e., no inspection of the corresponding C file is done). The names (and meaning) of the types known for performing that translation are to be found in Chapter 111 [Foreign Language Interface Properties], page 485; they are also summarized below (Prolog types are on the left, and the corresponding C types on the right):

- `num` <-> `double`
- `int` <-> `int`
• atm <-> char *
• string <-> char * (with trailing zero)
• byte_list <-> char * (a buffer of bytes, with associated length)
• int_list <-> int * (a buffer of integers, with associated length)
• address <-> void *

Strings, atoms, and lists of bytes are passed to (and from) C as dynamically (ciao_malloc) created arrays of characters (bytes). Those arrays are freed by Ciao Prolog upon return of the foreign function unless the property do_not_free/2 is specified (see examples below). This caters for the case in which the C files save in a private state (either by themselves, or by a library function being called by them) the values passed on from Prolog. The type byte_list/1 requires an additional property, size_of/2, to indicate which argument represents its size.

Empty lists of bytes and integers are converted into C NULL pointers, and vice versa. Empty strings (\[
\]) and null atoms (""") are converted into zero-length, zero-ended C strings ("""). C NULL strings and empty buffers (i.e., buffers with zero length) are transformed into the empty list or the null atom (""").

Most of the work is performed by the predicates in the Chapter 113 [Foreign Language Interface Builder], page 493, which can be called explicitly by the user. Doing that is not usually needed, since the Ciao Prolog Compiler takes care of building glue code files an of compiling and linking whatever is necessary.

110.3 Equivalence between Ciao Prolog and C modes

The (prefix) +/-1 ISO mode (or, equivalently, the in/1 mode) states that the corresponding Prolog argument is ground at the time of the call, and therefore it is an input argument in the C part; this groundness is automatically checked upon entry. The (prefix) +/-1 ISO mode (or, equivalently, the go/1 mode) states that Prolog expects the C side to generate a (ground) value for that argument. Arguments with output mode should appear in C functions as pointers to the corresponding base type (as it is usual with C), i.e., an argument which is an integer generated by the C file, declared as

```prolog
:- true pred get_int(go(ThisInt)) :: int + foreign
```

or as

```prolog
:- true pred get_int(-ThisInt) :: int + foreign
```

should appear in the C code as

```c
void get_int(int *thisint)
{
    ....
}
```

Note the type of the (single) argument of the function. Besides, the return value of a function can always be used as an output argument, just by specifying to which Prolog arguments it corresponds, using the foreign/1 property. The examples below illustrate this point, and the use of several assertions to guide the compilation.

110.4 Custom access to Prolog from C

Automatic type conversions does not cover all the possible cases. When the automatic type conversion is not enough (or if the user, for any reason, does not want to go through the automatic conversion), it is possible to instruct Ciao Prolog not to make implicit type conversion. The strategy in that case is to pass the relevant argument(s) with a special type (a ciao_term) which can represent any term which can be built in Prolog. Operations to construct, traverse,
and test this data abstraction from C are provided. The prototypes of these operations are placed on the "ciao_prolog.h" file, under the include subdirectory of the installation directory (the Ciao Prolog compiler knows where it has been installed, and gives the C compiler the appropriate flags). This non direct correspondence mode is activated whenever a Ciao Prolog type unknown to the foreign interface (i.e., none of these in Chapter 111 [Foreign Language Interface Properties], page 485) or the type any_term (which is explicitly recognised by the foreign language interface) is found. The latter is preferred, as it is much more informative, and external tools, as the the CiaoPP preprocessor, can take advantage of them.

110.4.1 Term construction

All term construction primitives return an argument of type ciao_term, which is the result of constructing a term. All Ciao Prolog terms can be built using the interface operations ciao_var(), ciao_structure(), ciao_integer(), and ciao_float(). There are, however, variants and specialized versions of these operations which can be freely intermixed. Using one version or another is a matter of taste and convenience. We list below the prototypes of the primitives in order of complexity.

Throughout this section, true, when referred to a boolean value, correspond to the integer value 1, and false correspond to the integer value 0, as is customary in C boolean expressions. These values also available as the (predefined) constants ciao_true and ciao_false, both of type ciao_bool.

- ciao_term ciao_var();
  Returns a fresh, unbound variable.
- ciao_term ciao_integer(int i);
  Creates a term, representing an integer from the Prolog point of view, from a C integer.
- ciao_term ciao_float(double i);
  Creates a term, representing a floating point number, from a floating point number.
- ciao_term ciao_put_number_chars(char *number_string);
  It converts number_string (which must a string representing a syntactically valid number) into a ciao_term.
- ciao_term ciao_atom(char *name);
  Creates an atom whose printable name is given as a C string.
- ciao_term ciao_structure_a(char *name, int arity, ciao_term *args);
  Creates a structure with name 'name' (i.e., the functor name), arity 'arity' and the components of the array 'args' as arguments: args[0] will be the first argument, args[1] the second, and so on. The 'args' array itself is not needed after the term is created, and can thus be a variable local to a procedure. An atom can be represented as a 0-arity structure (with ciao_structure(name, 0)), and a list cell can be constructed using the '.'/2 structure name. The _a suffix stands for array.
- ciao_term ciao_structure(char *name, int arity, ...);
  Similar to ciao_structure_a, but the C arguments after the arity are used to fill in the arguments of the structure.
- ciao_term ciao_list(ciao_term head, ciao_term tail);
  Creates a list from a head and a tail. It is equivalent to ciao_structure(".", 2, head, tail).
- ciao_term ciao_empty_list();
  Creates an empty list. It is equivalent to ciao_atom("[]").
- ciao_term ciao_listn_a(int len, ciao_term *args);
  Creates a list with 'len' elements from the array args. The nth element of the list (starting at 1) is args[n-1] (starting at zero).
- `ciao_term ciao_listn(int length, ...);
  Like `ciao_listn_a()`, but the list elements appear explicitly as arguments in the call.
- `ciao_term ciao_dlist_a(int len, ciao_term *args, ciao_term base);
  Like `ciao_listn_a()`, but a difference list is created. `base` will be used as the tail of the list, instead of the empty list.
- `ciao_term ciao_dlist(int length, ...);
  Similar to `ciao_dlist_a()` with a variable number of arguments. The last one is the tail of the list.
- `ciao_term ciao_copy_term(ciao_term src_term);
  Returns a new copy of the `term`, with fresh variables (as `copy_term/2` does).

110.4.2 Testing the Type of a Term

A `ciao_term` can contain *any* Prolog term, and its implementation is opaque to the C code. Therefore the only way to know reliably what data is passed on is using explicit functions to test term types. Below, `ciao_bool` is a type defined in "ciao_prolog.h" which can take the values 1 (for `true`) and 0 (for `false`).

- `ciao_bool ciao_is_variable(ciao_term term);
  Returns true if `term` is currently an uninstantiated variable.
- `ciao_bool ciao_is_number(ciao_term term);
  Returns true if `term` is an integer (of any length) or a floating point number.
- `ciao_bool ciao_is_integer(ciao_term term);
  Returns true if `term` is instantiated to an integer.
- `ciao_bool ciao_fits_in_int(ciao_term term);
  Returns true if `term` is instantiated to an integer which can be stored in an `int`, and false otherwise.
- `ciao_bool ciao_is_atom(ciao_term atom);
  Returns true if `term` is an atom.
- `ciao_bool ciao_is_list(ciao_term term);
  Returns true if `term` is a list (actually, a `cons` cell).
- `ciao_bool ciao_is_empty_list(ciao_term term);
  Returns true if `term` is the atom which represents the empty list (i.e., `[]`).
- `ciao_bool ciao_is_structure(ciao_term term);
  Returns true if `term` is a structure of any arity. This includes atoms (i.e., structures of arity zero) and lists, but excludes variables and numbers.

110.4.3 Term navigation

The functions below can be used to recover the value of a `ciao_term` into C variables, or to inspect Prolog structures.

- `int ciao_to_integer(ciao_term term);
  Converts `term` to an integer. `ciao_is_integer(term)` must hold.
- `ciao_bool ciao_to_integer_check(ciao_term term, int *result);
  Checks whether `term` fits into the size of an integer. If so, true is returned and `*result` is unified with the integer `term` represents. Otherwise, false is returned and `*result` is not touched.
• double ciao_to_float(ciao_term term);
  Converts term to a float value. ciao_is_number(term) must hold.

• char *ciao_get_number_chars(ciao_term term);
  It converts ciao_term (which must be instantiated to a number) into a C string representing
  the number in the current radix. The string returned is a copy, which must (eventually) be
  explicitly deallocated by the user C code using the operation ciao_free().

• char *ciao_atom_name(ciao_term atom);
  Returns the name of the atom. The returned string is the one internally used by Ciao
  Prolog, and should not be deallocated, changed or altered in any form. The advantage of
  using it is that it is fast, as no data copying is needed.

• char *ciao_atom_name_dup(ciao_term atom);
  Obtains a copy of the name of the atom. The string can be modified, and the programmer
  has the responsibility of deallocating it after being used. Due to the copy, it is slower than
  calling char *ciao_atom_name().

• ciao_term ciao_list_head(ciao_term term);
  Extracts the head of the list term. Requires term to be a list.

• ciao_term ciao_list_tail(ciao_term term);
  Extracts the tail of the list term. Requires term to be a list.

• char *ciao_structure_name(ciao_term term);
  Extracts the name of the structure term. Requires term to be a structure.

• int ciao_structure_arity(ciao_term term);
  Extracts the arity of the structure term.
  Requires term to be a structure.

• ciao_term ciao_structure_arg(ciao_term term, int n);
  Extracts the nth argument of the structure term. It behaves like arg/3, so the first argument
  has index 1. Requires term to be a structure.

110.4.4 Testing for Equality and Performing Unification

Variables of type ciao_term cannot be tested directly for equality: they are (currently)
implemented as a sort of pointers which may be aliased (two different pointers may refer to the
same object). The interface provides helper functions for testing term equality and to perform
unification of terms.

• ciao_bool ciao_unify(ciao_term x, ciao_term y);
  Performs the unification of the terms x and y, and returns true if the unification was
  successful. This is equivalent to calling the (infix) Prolog predicate =/2. The bindings are
  trailed and undone on backtracking.

• ciao_bool ciao_equal(ciao_term x, ciao_term y);
  Performs equality testing of terms, and returns true if the test was successful. This is
  equivalent to calling the (infix) Prolog predicate ==/2. Equality testing does not modify
  the terms compared.

110.4.5 Raising Exceptions

The following functions offers a way of throwing exceptions from C that can be caught in
Prolog with catch/3. The term that reaches Prolog is exactly the same which was thrown by
C. The execution flow is broken at the point where ciao_raise_exception() is executed, and
it returns to Prolog.

• void ciao_raise_exception(ciao_term ball);
  Raises an exception an throws the term ball.
110.4.6 Creating and disposing of memory chunks

Memory to be used solely by the user C code can be reserved/disposed of using, e.g., the well-known `malloc()`/`free()` functions (or whatever other functions the user may have available). However, memory explicitly allocated by Ciao Prolog and passed to C code, or allocated by C code and passed on to Ciao Prolog (and subject to garbage collection by it) should be allotted and freed (when necessary) by using the functions:

- `void *ciao_malloc(int size);`
- `void ciao_free(void *pointer);`

whose behavior is similar to `malloc()`/`free()`, but which will coordinate properly with Ciao Prolog’s internal memory management.

110.4.7 Calling Prolog from C

It is also possible to make arbitrary calls to Prolog predicates from C. There are two basic ways of make a query, depending on whether only one solution is needed (or if the predicate to be called is known to generate only one solution), or if several solutions are required.

When only one solution is needed `ciao_commit_call` obtains it (the solution obtained will obviously be the first one) and discards the resources used for finding it:

- `ciao_bool ciao_commit_call(char *name, int arity, ...);`
  Makes a call to a predicate and returns true or false depending on whether the query has succeeded or not. In case of success, the (possibly) instantiated variables are reachable from C.

- `ciao_bool ciao_commit_call_term(ciao_term goal);`
  Like `ciao_commit_call()` but uses the previously built term `goal` as goal.

If more than one solution is needed, it is necessary to use the `ciao_query` operations. A consult begins with a `ciao_query_begin` which returns a `ciao_query` object. Whenever an additional solution is required, the `ciao_query_next` function can be called. The query ends by calling `ciao_query_end` and all pending search branches are pruned:

- `ciao_query *ciao_query_begin(char *name, int arity, ...);`
  The predicate with the given name, arity and arguments (similar to the `ciao_structure()` operation) is transformed into a `ciao_query` object which can be used to make the actual query.

- `ciao_query *ciao_query_begin_term(ciao_term goal);`
  Like `ciao_query_begin` but using the term `goal` instead.

- `ciao_bool ciao_query_ok(ciao_query *query);`
  Determines whether the query may have pending solutions. A false return value means that there are no more solutions; a true return value means that there are more possible solutions.

- `void ciao_query_next(ciao_query *query);`
  Ask for a new solution.

- `void ciao_query_end(ciao_query *query);`
  Ends the query and frees the used resources.

110.5 Examples
110.5.1 Mathematical functions

In this example, the standard mathematical library is accessed to provide the sin, cos, and fabs functions. Note that the library is specified simply as

```prolog
:- use_foreign_library([m]).
```

The foreign interface adds the `-lm` at compile time. Note also how some additional options are added to optimize the compiled code (only glue code, in this case) and mathematics (only in the case of Linux in an Intel processor).

**File** `math.pl`:

```prolog
:- module(math, [sin/2, cos/2, fabs/2], [foreign,interface]).
:- true pred sin(in(X),go(Y)) :: num * num + (foreign,returns(Y)).
:- true pred cos(in(X),go(Y)) :: num * num + (foreign,returns(Y)).
:- true pred fabs(in(X),go(Y)) :: num * num + (foreign,returns(Y)).
:- extra_compiler_opts(['-O2']).
:- extra_compiler_opts('LINUXi86', ['-ffast-math']).
:- use_foreign_library('LINUXi86', m).
```

110.5.2 Addresses and C pointers

The address type designates any pointer, and provides a means to deal with C pointers in Prolog without interpreting them whatsoever. The C source file which implements the operations accessed from Prolog is declared with the

```prolog
:- use_foreign_source(objects_c).
```

directive.

**File** `objects.pl`:

```prolog
:- module(objects, [object/2, show_object/1], [foreign,interface]).
:- true pred object(in(N),go(Object)) :: int * address + (foreign,returns(Object)).
:- true pred show_object(in(Object)) :: address + foreign.
:- use_foreign_source(objects_c).
:- extra_compiler_opts('-O2').
```

**File** `objects.c.c`:

```c
#include <stdio.h>
struct object {
    char *name; char *colour;
};
#define OBJECTS 3
struct object objects[OBJECTS] = {
    {"ring","golden"}, {"table","brown"}, {"bottle","green"}
};
struct object *object(int n) { return &objects[n % OBJECTS]; }
void show_object(struct object *o) {
    printf("I show you a %s %s\n", o->colour, o->name);
}
```

110.5.3 Lists of bytes and buffers

A list of bytes (c.f., a list of ints) corresponds to a byte buffer in C. The length of the buffer is associated to that of the list using the property `size_of/2`. The returned buffer is freed by Ciao Prolog upon its reception, unless the `do_not_free/1` property is specified (see later). Conversely, a list of natural numbers in the range 0 to 255 can be passed to C as a buffer.

**File** `byte_lists.pl`:

```prolog
:- module(byte_lists, [obtain_list/3, show_list/2], [foreign,interface]).
:- true pred obtain_list(in(N),go(Length),go(List)) :: int * int * byte_list + (foreign,size_of(List,Length)).
:- true pred show_list(in(Length),in(List)) :: int * byte_list + (foreign,size_of(List,Length)).
```
The Ciao Prolog System

:- use_foreign_source(bytes_op).

**File bytes_op.c:**
#include <stdlib.h> #include <stdio.h>

```c
void obtain_list(int n, int *l, char **s) { int i; int c; if (n < 0) n = 0; *l = n; *s = (char *)malloc(*l); for (i = 0; i < *l; i++) { (*s)[i] = i; }
}
```

```c
void show_list(int l, char *s) {
if (s)
{
int n; printf("From C:"); for (n = 0; n < l; n++) {
printf(" %d", s[n]);
} printf("\n");
} else {
printf("From C: \[\n");
}
```

**110.5.4 Lists of integers**

**File int_lists.pl:**
:- module(int_lists, [obtain_list/3, show_list/2], [foreign_interface]).
:- true pred obtain_list(in(N), go(Length), go(List)) :: int * int * int list + (foreign, size_of(List, Length)). :- true pred show_list(in(Length), in(List)) :: int * int list + (foreign, size_of(List, Length)). :- use_foreign_source(ints_op).

**File ints_op.c:**
#include <stdlib.h> #include <stdio.h>

```c
void obtain_list(int n, int *l, int **s) {
int i; int c; if (n < 0) n = 0; *l = n; *s = (int *)malloc(*l * sizeof(int)); for (i = 0; i < *l; i++) {
(*s)[i] = i;
}
```

```c
void show_list(int l, int *s) {
if (s)
{
int n; printf("From C:"); for (n = 0; n < l; n++) {
printf(" %d", s[n]);
} printf("\n");
} else {
printf("From C: \[\n");
}
```

**110.5.5 Strings and atoms**

A C string can be seen as a buffer whose end is denoted by the trailing zero, and therefore stating its length is not needed. Two translations are possible into Ciao Prolog: as a Prolog string (list of bytes, with no trailing zero) and as an atom. These are selected automatically just by choosing the corresponding type (look at the examples below).

Note how the **do_not_free/1** property is specified in the **a_string/1** predicate: the string returned by C is static, and therefore it should not be freed by Prolog.

**File strings_and_atoms.pl:**
:- module(strings_and_atoms, [lookup_string/2, lookup_atom/2, a_string/1, show_string/1, show_atom/1], [foreign_interface]).
:- true pred a_string(go(S)) :: string + (foreign(get_static_str), returns(S), do_not_free(S)).
:- true pred lookup_string(in(N), go(S)) :: int * string + (foreign(get_str), returns(S)).
:- true pred lookup_atom(in(N), go(S)) :: int * atm + (foreign(get_str), returns(S)).
:- true pred show_string(in(S)) :: string + foreign(put_str). :- true pred show_atom(in(S)) :: atm + foreign(put_str).
:- use_foreign_source(str_op).

**File str_op.c:**
#include <stdlib.h> #include <stdio.h>

```c
char *get_static_str() { return "this is a string Prolog should not free"; }
```

```c
char *get_str(int n) { char *s; int size; int i; int c; if (n < 0) n = -n; size = (n%4) + 5; s = (char *)malloc(size+1); for (i = 0, c = ((i + n) % ('z' - 'a' + 1)) + 'a'; i < size; i++, c++) {
if (c > 'z') c = 'a'; s[i] = c;
} s[i] = 0; return s;
}
```

```c
void put_str(char *s) {
if (s) {
printf("From C: \"%s\n", s);
} else {
printf("From C: null\n");
}
```
110.5.6 Arbitrary Terms

This example shows how data Prolog can be passed untouched to C code, and how it can be manipulated there.

File `any_term.pl`:

```prolog
:- module(any_term, [custom_display_term/1, custom_create_term/2], [foreign_interface]).
:- true pred custom_display_term(in(X)) :: any_term + foreign. :- true pred custom_create_term(in(L), go(X)) :: int * any_term + (foreign, returns(X)).
:- use_foreign_source(any_term_c). :- extra_compiler_opts('-O2').
```

File `any_term.c`:

```c
#include <stdio.h>
#include "ciao_prolog.h"

ciao_term custom_create_term(int n) {
  ciao_term t; t = ciao_empty_list(); while (n > 0) {
    t = ciao_list(ciao_integer(n), t); n--;
  } return t;
}

void custom_display_term(ciao_term term) {
  if (ciao_is_atom(term)) {
    printf("<atom name="", ciao_atom_name(term));
  } else if (ciao_is_structure(term)) {
    int i; int a = ciao_structure_arity(term); printf("<structure name="", ciao_structure_name(term), a); for (i = 1; i <= a; i++) {
      printf("<argument number="", i); custom_display_term(ciao_structure_arg(term, i));
    } printf("</structure>");
  } else if (ciao_is_list(term)) {
    printf("<list>");
    if (ciao_is_empty_list(term)) {
      printf("<empty_list/>");
    } else if (ciao_is_integer(term)) {
      printf("<integer value="", ciao_to_integer(term));
    } else if (ciao_is_number(term)) {
      printf("<float value="", ciao_to_float(term));
    } else {
      printf("<unknown/>");
    }
  }
}
```

110.5.7 Exceptions

The following example defines a predicate in C that converts a list of codes into a number using `strtol()`. If this conversion fails, then an exception is raised.

File `exceptions_example.pl`:

```prolog
:- module(exceptions_example, [codes_to_number_c/2, safe_codes_to_number/2], [foreign_interface]).
:- use_module(library(format)).
% If the string is not a number raises an exception. :- true pred codes_to_number_c(in(X), go(Y)) :: string * int + (foreign, returns(Y)).
safe_codes_to_number(X, Y) :- catch(codes_to_number_c(X, Y), Error, handle_exception(Error)).
handle_exception(Error) :- format("Exception caught ~w"n", [Error]).
:- use_foreign_source(exceptions_c). :- extra_compiler_opts('-O2').
```

File `exceptions_c.c`:

```c
#include <string.h> #include "ciao_prolog.h"
int codes_to_number_c(char *s) {
  char *endptr; int n; n = strtol(s, &endptr, 10); if (endptr == NULL || *endptr != '0') {
    ciao_raise_exception(ciao_structure("codes_to_number_exception", 1, ciao_atom(s)));
  } return n;
}
```
110.5.8 Testing number types and using unbound length integers

Unbound length integers (and, in general, any number) can be converted to/from ciao_terms by using strings. The following examples show two possibilities: one which tries to be as smart as possible (checking whether numbers fit into a machine int or not), and being lazy and simpler - and probably slower.

File bigints.pl:

```
:- module(bigints, [ make_smart_conversion/3, % Checks and uses convenient format force_string_conversion/2 % Passes around using strings ], [foreign_interface]).
:- true pred make_smart_conversion_c(in(X), go(Y), go(How)):: any_term * any_term * any_term + foreign # "Given a number @var{X}, it is unified with @var{Y} by using the most specific internal representation (short integer, float, or long integer). @var{How} returns how the conversion was done. It behaves unpredictably if @var{X} is not a number."

:- true pred force_string_conversion_c(in(X), go(Y)):: any_term * any_term + foreign # "Given a number @var{X}, it is unified with @var{Y} by using the most general internal representation (a string of characters). It behaves unpredictably if @var{X} is not a number."

:- use_foreign_source(bigints_c).
make_smart_conversion(A, B, C):- number(A), % Safety test make_smart_conversion_c(A, B, C).
force_string_conversion(A, B):- number(A), % Safety test force_string_conversion_c(A, B).
```

File bigints.c.c:

```
#include "ciao_prolog.h"

void make_smart_conversion_c(ciao_term number_in, ciao_term *number_out, ciao_term *how_converted) { int inter_int; double inter_float; char * inter_str;
if (ciao.fits_in_int(number_in)) { /* Includes the case of being a float */ inter_int = ciao.to_integer(number_in); *number_out = ciao.integer(inter_int); *how_converted = ciao.atom("machine_integer"); } else if (ciao.is_integer(number_in)) { /* Big number */ inter_str = ciao.get_number_chars(number_in); *number_out = ciao.put_number_chars(inter_str); ciao.free(inter_str); *how_converted = ciao.atom("string"); } else { /* Must be a float */ inter_float = ciao.to_float(number_in); *number_out = ciao.float(inter_float); *how_converted = ciao.atom("float"); } }

void force_string_conversion_c(ciao_term number_in, ciao_term *number_out) { char *inter_str; inter_str = ciao.get_number_chars(number_in); *number_out = ciao.put_number_chars(inter_str); ciao.free(inter_str); }
```

110.6 Usage and interface (foreign_interface)

- **Library usage:**
  
  The foreign interface is used by including foreign_interface in the include list of a module, or by means of an explicit :- use_package(foreign_interface).
111 Foreign Language Interface Properties

Author(s): Jose Morales, Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

The foreign language interface uses some properties to specify linking regimes, foreign files to be compiled, types of data available, memory allocation policies, etc.

111.1 Usage and interface (foreign_interface_properties)

- Library usage:
  ```prolog
  :- use_module(library(foreign_interface_properties)).
  ```

- Exports:
  - Properties:
    native/1, native/2, size_of/3, foreign/1, foreign/2, returns/2, do_not_free/2.
  - Regular Types:
    int_list/1, byte_list/1, byte/1, null/1, address/1, any_term/1.

111.2 Documentation on exports (foreign_interface_properties)

int_list/1:
  Usage: int_list(List)
  - Description: List is a list of integers.

byte_list/1:
  Usage: byte_list(List)
  - Description: List is a list of bytes.

byte/1:
  Usage: byte(Byte)
  - Description: Byte is a byte.

null/1:
  Usage: null(Address)
  - Description: Address is a null address.

address/1:
  Usage: address(Address)
  - Description: Address is a memory address.
any_term/1:  REGTYPE
   Usage: any_term(X)
   Description: X is any term. The foreign interface passes it to C functions as a general term.

native/1:  PROPERTY
   Usage: native(Name)
   Description: The Prolog predicate Name is implemented using the function Name. The implementation is not a common C one, but it accesses directly the internal Ciao Prolog data structures and functions, and therefore no glue code is generated for it.

native/2:  PROPERTY
   Usage: native(PrologName, ForeignName)
   Description: The Prolog predicate PrologName is implemented using the function ForeignName. The same considerations as above example are to be applied.

size_of/3:  PROPERTY
   Usage: size_of(Name, ListVar, SizeVar)
   Description: For predicate Name, the size of the argument of type byte_list/1, ListVar, is given by the argument of type integer SizeVar.

foreign/1:  PROPERTY
   Usage: foreign(Name)
   Description: The Prolog predicate Name is implemented using the foreign function Name.

foreign/2:  PROPERTY
   Usage: foreign(PrologName, ForeignName)
   Description: The Prolog predicate PrologName is implemented using the foreign function ForeignName.

returns/2:  PROPERTY
   Usage: returns(Name, Var)
   Description: The result of the foreign function that implements the Prolog predicate Name is unified with the Prolog variable Var. Cannot be used without foreign/1 or foreign/2.

do_not_free/2:  PROPERTY
   Usage: do_not_free(Name, Var)
   Description: For predicate Name, the C argument passed to (returned from) the foreign function will not be freed after calling the foreign function.
111.3 Documentation on internals (foreign_interface_properties)

use_foreign_source/1: 
Usage: :- use_foreign_source(Files).

- Description: Files is the (list of) foreign file(s) that will be linked with the glue-code file.
- The following properties hold at call time:
  Files is an atom or a list of atoms.  
  
use_foreign_source/2: 
Usage: :- use_foreign_source(OsArch, Files).

- Description: Files are the OS and architecture dependant foreign files. This allows compiling and linking different files depending on the O.S. and architecture.
- The following properties hold at call time:
  OsArch is an atom.  
  Files is an atom or a list of atoms.  

use_foreign_library/1: 
Usage: :- use_foreign_library(Libs).

- Description: Libs is the (list of) external library(es) needed to link the C files. Only the short name of the library (i.e., what would follow the -l in the linker) is needed.
- The following properties hold at call time:
  Libs is an atom or a list of atoms.  

use_foreign_library/2: 
Usage: :- use_foreign_library(OsArch, Libs).

- Description: Libs are the OS and architecture dependant libraries.
- The following properties hold at call time:
  OsArch is an atom.  
  Libs is an atom or a list of atoms.  

extra_compiler_opts/1: 
Usage: :- extra_compiler_opts(Opts).

- Description: Opts is the list of additional compiler options (e.g., optimization options) that will be used during the compilation.
- The following properties hold at call time:
  Opts is an atom or a list of atoms.  
  
  

extra_compiler_opts/2:  DECLARATION
Usage: :- extra_compiler_opts(OsArch, Opts).

- Description: Opts are the OS and architecture dependant additional compiler options.
- The following properties hold at call time:
  OsArch is an atom.  (basic_props:atm/1)
  Opts is an atom or a list of atoms.  (basic_props:atm_or_atm_list/1)

use_compiler/1:  DECLARATION
Usage: :- use_compiler(Compiler).

- Description: Compiler is the compiler to use in this file. When this option is used,
  the default (Ciao-provided) compiler options are not used; those specified in extra_  compiler_options are used instead.
- The following properties hold at call time:
  Compiler is an atom.  (basic_props:atm/1)

use_compiler/2:  DECLARATION
Usage: :- use_compiler(OsArch, Compiler).

- Description: Compiler is the compiler to use in this file when compiling for the  architecture OsArch. The option management is the same as in use_compiler/2.
- The following properties hold at call time:
  OsArch is an atom.  (basic_props:atm/1)
  Compiler is an atom.  (basic_props:atm/1)

extra_linker_opts/1:  DECLARATION
Usage: :- extra_linker_opts(Opts).

- Description: Opts is the list of additional linker options that will be used during the  linkage.
- The following properties hold at call time:
  Opts is an atom or a list of atoms.  (basic_props:atm_or_atm_list/1)

extra_linker_opts/2:  DECLARATION
Usage: :- extra_linker_opts(OsArch, Opts).

- Description: Opts are the OS and architecture dependant additional linker options.
- The following properties hold at call time:
  OsArch is an atom.  (basic_props:atm/1)
  Opts is an atom or a list of atoms.  (basic_props:atm_or_atm_list/1)

use_linker/1:  DECLARATION
Usage: :- use_linker(Linked).

- Description: Linked is the linker to use in this file. When this option is used, the  default (Ciao-provided) linker options are not used; those specified in extra_linker_  options/1 are used instead.
The following properties hold at call time:
Linker is an atom.  

use_linker/2:
Usage: :- use_linker(OsArch, Linker).
- Description: Compiler is the linker to use in this file when compiling for the architecture OsArch. The option management is the same as in use_compiler/2.

foreign_inline/2:
Usage 1: :- foreign_inline(Term, Text).
- Description: Term is a predicate name. Text is a source C code that define the predicate Term. Term is present for future use with the analyzers. Example of this can be viewed in the hrtimer library.

Usage 2: :- foreign_inline(Term, Text).
- Description: This usage of foreign_inline is to add globally the C source code that are in Text.

111.4 Known bugs and planned improvements (foreign_interface_properties)
- The size_of/3 property has an empty definition
112 Utilities for on-demand compilation of foreign files

Author(s): Manuel Carro, Jose Morales.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.3#36 (1999/7/20, 10:37:31 MEST)

This module provides two predicates which give the user information regarding how to compile external (C) files in order to link them with the Ciao Prolog engine at runtime.

These predicates are not intended to be called directly by the end-user. Instead, a tool or module whose aim is generating dynamically loadable files from source files should use the predicates in this file in order to find out what are the proper compiler and linker to use, and which options must be passed to them in the current architecture.

112.1 Usage and interface (foreign_compilation)

- Library usage:
  :use_module(library(foreign_compilation)).

- Exports:
  - Predicates:
    compiler_and_opts/2, linker_and_opts/2.

- Other modules used:
  - System library modules:
    system.

112.2 Documentation on exports (foreign_compilation)

compiler_and_opts/2:

Usage: compiler_and_opts(Compiler, Opts)

Description: If you want to compile a foreign language file for dynamic linking in the current operating system and architecture, you have to use the compiler Compiler and give it the options Opts. A variable in Opts means that no special option is needed.

The following properties should hold at call time:

Compiler is currently instantiated to an atom. (term_typing:atom/1)
Opts is a list of atoms. (basic_props:list/2)

linker_and_opts/2:

Usage: linker_and_opts(Compiler, Opts)

Description: If you want to link a foreign language file for dynamic linking in the current operating system and architecture, you have to use the linker Compiler and give it the options Opts. A variable in Opts means that no special option is needed.

The following properties should hold at call time:

Linker is currently instantiated to an atom. (term_typing:atom/1)
Options is a list of atoms. (basic_props:list/2)
113 Foreign Language Interface Builder

Author(s): Jose Morales, Manuel Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#218 (2004/4/13, 18:31:33 CEST)

Low-level utilities for building foreign interfaces. End-users should not need to use them, as the Ciao Prolog Compiler reads the user assertions and calls appropriately the predicates in this module.

113.1 Usage and interface (build_foreign_interface)

- **Library usage:**
  
  ```prolog
  :- use_module(library(build_foreign_interface)).
  ```

- **Exports:**
  - **Predicates:**
    - `build_foreign_interface/1`
    - `rebuild_foreign_interface/1`
    - `build_foreign_interface_explicit_decls/2`
    - `rebuild_foreign_interface_explicit_decls/2`
    - `build_foreign_interface_object/1`
    - `rebuild_foreign_interface_object/1`
    - `do_interface/1`

- **Other modules used:**
  - **System library modules:**
    - `write_c`, `write_c`
    - `streams`
    - `terms`
    - `lists`
    - `llists`
    - `aggregates`
    - `system`
    - `format`
    - `messages`
    - `assertions/assrt_lib`
    - `foreign_compilation`
    - `compiler/c_itf`
    - `ctrlcclean`
    - `errhandle`

113.2 Documentation on exports (build_foreign_interface)

**build_foreign_interface/1:**

**PREDICATE**

**Usage:** `build_foreign_interface(File)`

- **Description:** Reads assertions from `File`, generates the gluecode for the Ciao Prolog interface, compiles the foreign files and the gluecode file, and links everything in a shared object. Checks modification times to determine automatically which files must be generated/compiled/linked.

- **Call and exit should be compatible with:**
  - `File` is a source name.  
    `streams_basic:sourcename/1`
  - **The following properties should hold at call time:**
    - `File` is currently ground (it contains no variables).  
      `term_typing:ground/1`
  - **The following properties should hold upon exit:**
    - `File` is currently ground (it contains no variables).  
      `term_typing:ground/1`

**rebuild_foreign_interface/1:**

**PREDICATE**

**Usage:** `rebuild_foreign_interface(File)`
- **Description:** Like `build_foreign_interface/1`, but it does not check the modification time of any file.

- **Call and exit should be compatible with:**
  - `File` is a source name.  
  - `Declis` is a list of terms.  

- **The following properties should hold at call time:**
  - `File` is currently ground (it contains no variables).  
  - `Declis` is currently ground (it contains no variables).

- **The following properties should hold upon exit:**
  - `File` is currently ground (it contains no variables).  
  - `Declis` is currently ground (it contains no variables).

---

**build_foreign_interface_explicit_decls/2:**

**Usage:** `build_foreign_interface_explicit_decls(File, Declis)`

- **Description:** Like `build_foreign_interface/1`, but use declarations in `Declis` instead of reading the declarations from `File`.

- **Call and exit should be compatible with:**
  - `File` is a source name.  
  - `Declis` is a list of terms.  

- **The following properties should hold at call time:**
  - `File` is currently ground (it contains no variables).  
  - `Declis` is currently ground (it contains no variables).

- **The following properties should hold upon exit:**
  - `File` is currently ground (it contains no variables).  
  - `Declis` is currently ground (it contains no variables).

---

**rebuild_foreign_interface_explicit_decls/2:**

**Usage:** `rebuild_foreign_interface_explicit_decls(File, Declis)`

- **Description:** Like `build_foreign_interface_explicit_decls/1`, but it does not check the modification time of any file.

- **Call and exit should be compatible with:**
  - `File` is a source name.  
  - `Declis` is a list of terms.  

- **The following properties should hold at call time:**
  - `File` is currently ground (it contains no variables).  
  - `Declis` is currently ground (it contains no variables).

- **The following properties should hold upon exit:**
  - `File` is currently ground (it contains no variables).  
  - `Declis` is currently ground (it contains no variables).

---

**build_foreign_interface_object/1:**

**Usage:** `build_foreign_interface_object(File)`

- **Description:** Compiles the gluecode file with the foreign source files producing an unique object file.
Chapter 113: Foreign Language Interface Builder

- **Call and exit should be compatible with:**
  - File is a source name. (streams_basic:sourcename/1)

- **The following properties should hold at call time:**
  - File is currently ground (it contains no variables). (term_typing:ground/1)

- **The following properties should hold upon exit:**
  - File is currently ground (it contains no variables). (term_typing:ground/1)

**PREDICATE**

**rebuild_foreign_interface_object/1:**

**Usage:** rebuild_foreign_interface_object(File)

- **Description:** Compiles (again) the gluecode file with the foreign source files producing an unique object file.
- **Call and exit should be compatible with:**
  - File is a source name. (streams_basic:sourcename/1)
- **The following properties should hold at call time:**
  - File is currently ground (it contains no variables). (term_typing:ground/1)
- **The following properties should hold upon exit:**
  - File is currently ground (it contains no variables). (term_typing:ground/1)

**PREDICATE**

**do_interface/1:**

**Usage:** do_interface(Decls)

- **Description:** Given the declarations in Decls, this predicate succeeds if these declarations involve the creation of the foreign interface.
- **Call and exit should be compatible with:**
  - Decls is a list of terms. (basic_props:list/2)
- **The following properties should hold at call time:**
  - Decls is currently ground (it contains no variables). (term_typing:ground/1)
- **The following properties should hold upon exit:**
  - Decls is currently ground (it contains no variables). (term_typing:ground/1)
114 Interface to daVinci

Author(s): Francisco Bueno.

This library allows connecting a Ciao Prolog application with daVinci V2.X.

The communication is based on a two-way channel: after daVinci is started, messages are sent in to it and read in from it on demand by different Prolog predicates. Messages are sent via writing the term as text; messages are received by reading text and returning an atom. Commands sent and answers received are treated as terms from the Prolog side, since for daVinci they are text but have term syntax; the only difficulty lies in strings, for which special Prolog syntax is provided.

See accompanying file library('davinci/commands') for examples on the use of this library. daVinci is developed by U. of Bremen, Germany.

114.1 Usage and interface (davinci)

| Library usage: |
| :-- use_module(library(davinci)). |

| Exports: |
| – Predicates: |
| davinci/0, topd/0, davinci_get/1, davinci_get_all/1, davinci_put/1, davinci_quit/0, davinci_ugraph/1, davinci_lgraph/1, ugraph2term/2, formatting/2. |

| Other modules used: |
| – System library modules: |
| aggregates, prompt, errhandle, format, read, graphs/ugraphs, write, system. |

114.2 Documentation on exports (davinci)

davinci/0: PREDICATE
Start up a daVinci process.

topd/0: PREDICATE
A toplevel to send to daVinci commands from standard input.

davinci_get/1: PREDICATE
Usage: davinci_get(Term)
   – Description: Get a message from daVinci. Term is a term corresponding to daVinci’s message.

davinci_get_all/1: PREDICATE
Usage: davinci_get_all(List)
   – Description: Get all pending messages. List is a list of terms as in davinci_get/1.
The following properties should hold upon exit:
List is a list. 

**davinci_put/1:**

*Usage:* `davinci_put(Term)`

- *Description:* Send a command to daVinci.
- *The following properties should hold at call time:*
  
  `davinci:davinci_command(Term)`

**davinci_quit/0:**

Exit daVinci process. All pending answers are lost!

**davinci_ugraph/1:**

*Usage:* `davinci_ugraph(Graph)`

- *Description:* Send a graph to daVinci.
- *The following properties should hold at call time:*
  
  `davinci:ugraph(Graph)`

**davinci_lgraph/1:**

*Usage:* `davinci_lgraph(Graph)`

- *Description:* Send a labeled graph to daVinci.
- *The following properties should hold at call time:*
  
  `davinci:lgraph(Graph)`

**ugraph2term/2:**

No further documentation available for this predicate.

**formatting/2:**

No further documentation available for this predicate.

### 114.3 Documentation on internals (davinci)

**davinci_command/1:**

Syntactically, a command is a term. Semantically, it has to correspond to a command understood by daVinci. Two terms are interpreted in a special way: *string/1* and *text/1*: `string(Term)` is given to daVinci as "Term"; `text(List)` is given as "Term1 Term2 ...Term" for each Term in List. If your term has functors *string/1* and *text/1* that you don't want to be interpreted this way, use it twice, i.e., `string(string(Term))` is given to daVinci as `string(Term')` where Term' is the interpretation of Term."
**ugraph/1:**

\texttt{ugraph(Graph)}

\texttt{Graph} is a term which denotes an ugraph as in \texttt{library(ugraphs)}. Vertices of the form \texttt{node/2} are interpreted in a special way: \texttt{node(Term,List)} is interpreted as a vertex \texttt{Term} with attributes \texttt{List}. \texttt{List} is a list of terms conforming the syntax of \texttt{davinci\_put/1} and corresponding to daVinci’s graph nodes attributes. If your vertex has functor \texttt{node/2} and you don’t want it to be interpreted this way, use it twice, i.e., \texttt{node(node(T1,T2),[])} is given to daVinci as vertex \texttt{node(T1,T2)}. A vertex is used both as label and name of daVinci’s graph node. daVinci’s graph edges have label \texttt{V1-V2} where \texttt{V1} is the source and \texttt{V2} the sink of the edge. There is no support for multiple edges between the same two vertices.

**lgraph/1:**

\texttt{lgraph(Graph)}

\texttt{Graph} is a term which denotes a wgraph as in \texttt{library(wgraphs)}, except that the weights are labels, i.e., they do not need to be integers. Vertices of the form \texttt{node/2} are interpreted in a special way. Edge labels are converted into special intermediate vertices. Duplicated labels are solved by adding dummy atoms ".". There is no support for multiple edges between the same two vertices.
The Ciao Prolog System
Chapter 115: The Tcl/Tk interface


Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#314 (2004/2/25, 18:27:47 CET)

The tcltk library package is a bidirectional interface to the Tcl language and the Tk toolkit. Tcl is an interpreted scripting language with many extension packages, particularly the graphical interface toolkit, Tk. The interaction between both languages is expressed in terms of an interface between the Tcl/Tk process and the Prolog process. This approach allows the development of mixed applications where both sides, Tcl/Tk and Prolog, can be combined in order to exploit their respective capabilities.

This library uses two sockets to connect both the Tcl and the Prolog processes: event_socket and term_socket. There are also two Tcl global variables: prolog_variables and terms. The value of any of the bound variables in a goal will be stored in the array prolog_variables with the variable name as index. Terms is the string which contains the printed representation of prolog terms.

Prolog to Tcl

The Tcl/Tk side waits for requests from the Prolog side, and executes the Tcl/Tk code received. Also, the Tcl/Tk side handles the events and exceptions which may be raised on its side, passing on control to the Prolog side in case it is necessary.

To use Tcl, you must create a Tcl interpreter object and send commands to it. A Tcl command is specified as follows:

```
Command        --> Atom { other than [ ] } |
| Number         |
| chars(PrologString) |
| write(Term)    |
| format(Fmt, Args) |
| dq(Command)    |
| br(Command)    |
| sqb(Command)   |
| min(Command)   |
| ListOfCommands |

ListOfCommands --> []

[Command|ListOfCommands]
```

where:

- **Atom** denotes the printed representation of the atom.
- **Number** denotes their printed representations.
- **chars(PrologString)** denotes the string represented by PrologString (a list of character codes).
- **write(Term)** denotes the string that is printed by the corresponding built-in predicate.
- **format(Term)** denotes the string that is printed by the corresponding built-in predicate.
- **dq(Command)** denotes the string specified by Command, enclosed in double quotes.
- **br(Command)** denotes the string specified by Command, enclosed in braces.
sqb(Command) denotes the string specified by Command, enclosed in square brackets.

min(Command) denotes the string specified by Command, immediately preceded by a hyphen.

ListOfCommands denotes the strings denoted by each element, separated by spaces.

The predicates to use Tcl from Prolog are tcl_new/1, tcl_delete/1, tcl_eval/3, and tcl_event/3.

An example of use with Prolog as master and Tcl as slave, consisting of a GUI to a program which calculates the factorial of a number:

```prolog
:- use_module(library(tcltk)).

go :-
  tcl_new(X),
  tcl_eval(X,[button,'.b',min(text),
      dq('Compute!')],_),
  tcl_eval(X,[button,'.c',
      'text',
      dq('Quit')],_),
  tcl_eval(X,[entry,'.e1',
      min(textvariable),
      'inputval'],_),
  tcl_eval(X,[label,'.l1',
      min(text),
      dq('The factorial of ')],_),
  tcl_eval(X,[pack,'.l1','.e1'],_),
  tcl_eval(X,[entry,'.e2',
      min(textvariable),
      'outputval'],_),
  tcl_eval(X,[label,'.l2',
      min(text),
      dq('is ')],_),
  tcl_eval(X,[pack,'.l2','.e2'],_),
  tcl_eval(X,[pack,'.b','.c',
      min(side),
      'left'],_),
  tcl_eval(X,[bind,'.b','<ButtonPress-1>',
      br([set,'inputval','$inputval','
         prolog_one_event,
         dq(write(execute(tk_test_aux:factorial('$inputval',
             set, 'outputval','$prolog_variables(Outputval)')))])],_),
  tcl_eval(X,[bind,'.c','<ButtonPress-1>',
      br([prolog_one_event,
         dq(write(execute(exit_tk_event_loop))))]],_),
  tk_event_loop(X).
```

Tcl to Prolog

This is the usual way to build a GUI application. The slave, Prolog, behaves as a server that fulfills eventual requests from the master side, Tcl. At some point, during the user interaction with the GUI, an action may take place that triggers the execution of some procedure on the slave side (a form submit, for example). Thus, the slave is invoked, performs a service, and returns the result to the GUI through the socket connection.

This library includes two main specific Tcl commands:

prolog Goal

Goal is a string containing the printed representation of a Prolog goal. The goal will be called in the user module unless it is prefixed with another module name. The call is always deterministic and its can be either of the following:

1, in case of success

The value of any of the variables in the goal that is bound to a term will be returned to Tcl in the array prolog_variables with the variable name as index.

0, if the execution fails

The Prolog exception Tcl exception is raised. The error message will be "Prolog Exception: " appended with a string representation of such exception.
prolog_event Term
Add the new term to the terms queue. These can be later retrieved through
predicates tcl_event/3 and tk_next_event/2.

Additionally, seven extra Tcl commands are defined.

prolog_delete_event
Deletes the first term of the terms queue.

prolog_list_events
Sends all the terms of the terms queue through the event_socket. The last element
is end_of_event_list.

prolog_cmd Command
Receives as an argument the Tcl/Tk code, evaluates it and returns through the
term_socket the term tcl_error in case of error or the term tcl_result with the result
of the command executed. If the command is prolog, upon return, the goal run on
the prolog side is received. In order to get the value of the variables, predicates are
compared using the unify_term command. Returns 0 when the script runs without
errors, and 1 if there is an error.

prolog_one_event Term
Receives as an argument the term associated to one of the Tk events. Sends the term
through the event_socket and waits for its unification. Then unify_term command
is called to update the prolog_variables array.

prolog_thread_event Term
Receives as an argument the term associated to one of the Tk events. Sends the term
through the event_socket and waits for its unification. Then unify_term command
is called to update the prolog_variables array. In this case the term_socket is non
blocking.

convert_variables String
Its argument is a string containing symbols that can not be sent through the sockets.
This procedure deletes them from the input string and returns the new string.

unify_term Term1 Term2
Unifies Term1 and Term2 and updates the prolog_variables array.

The predicates to use Prolog from Tcl are tk_event_loop/1, tk_main_loop/1, tk_new/2,
and tk_next_event/2.

An example of use with Tcl as master and Prolog as slave, implementing the well known "Hello,
world!" dummy program (more can be seen in directory examples):

Prolog side:
:
:- use_module(library(tcltk)).
:- use_package(classic).

hello('Hello, world!').

go :-

    tk_new([name('Simple'), Tcl],
    tcl_eval(Tcl, 'source simple.tcl', _),
    tk_main_loop(Tcl),
    tcl_delete(Tcl).

Tcl side (simple.tcl):

label .l -textvariable tvar
button .b -text "Go!" -command {run}
pack .l .b -side top

proc run {} {
    global prolog_variables
    global tvar

    prolog hello(X)
    set tvar $prolog_variables(X)
}

115.1 Usage and interface (tcltk)

- Library usage:
  :- use_module(library(tcltk)).

- Exports:
  - Predicates:
    - tcl_new/1, tcl_eval/3, tcl_delete/1, tcl_event/3, tk_event_loop/1, tk_main_loop/1, tk_new/2, tk_next_event/2.
  - Regular Types:
    - tclInterpreter/1, tclCommand/1.

- Other modules used:
  - System library modules:
    - tcltk/tcltk_low_level, iso_misc, write, strings, lists.

115.2 Documentation on exports (tcltk)

**tcl_new/1:**

**PREDICATE**

**Usage:** tcl_new(TclInterpreter)

- **Description:** Creates a new interpreter, initializes it, and returns a handle to it in TclInterpreter.
- **Call and exit should be compatible with:**
  - TclInterpreter is a reference to a Tcl interpreter. (tcltk:tclInterpreter/1)
  - **The following properties should hold at call time:**
  - TclInterpreter is a free variable. (term_typing:var/1)

**tcl_eval/3:**

**PREDICATE**

Meta-predicate with arguments: tcl_eval(? ,? ,addmodule).

**Usage:** tcl_eval(TclInterpreter, Command, Result)

- **Description:** Evaluates the commands given in Command in the Tcl interpreter TclInterpreter. The result will be stored as a string in Result. If there is an
error in Command an exception is raised. The error messages will be Tcl Exception: if the error is in the syntax of the Tcl/Tk code or Prolog Exception:; if the error is in the prolog term.

- Call and exit should be compatible with:
  TclInterpreter is a reference to a Tcl interpreter.  (tcltk:tclInterpreter/1)
  Command is a Tcl command.  (tcltk:tclCommand/1)
  Result is a string (a list of character codes).  (basic_props:string/1)

- The following properties should hold at call time:
  TclInterpreter is currently a term which is not a free variable.  (term_typing:nonvar/1)
  Command is currently a term which is not a free variable.  (term_typing:nonvar/1)
  Result is a free variable.  (term_typing:var/1)

**tcl_delete/1:**  
**PREDICATE**

**Usage:** tcl_delete(TclInterpreter)

- **Description:** Given a handle to a Tcl interpreter in variable TclInterpreter, it deletes the interpreter from the system.

- Call and exit should be compatible with:
  TclInterpreter is a reference to a Tcl interpreter.  (tcltk:tclInterpreter/1)

- The following properties should hold at call time:
  TclInterpreter is currently a term which is not a free variable.  (term_typing:nonvar/1)

**tcl_event/3:**  
**PREDICATE**

**Usage:** tcl_event(TclInterpreter, Command, Events)

- **Description:** Evaluates the commands given in Command in the Tcl interpreter whose handle is provided in TclInterpreter. Events is a list of terms stored from Tcl by prolog_event. Blocks until there is something on the event queue

- Call and exit should be compatible with:
  TclInterpreter is a reference to a Tcl interpreter.  (tcltk:tclInterpreter/1)
  Command is a Tcl command.  (tcltk:tclCommand/1)
  Events is a list.  (basic_props:list/1)

- The following properties should hold at call time:
  TclInterpreter is currently a term which is not a free variable.  (term_typing:nonvar/1)
  Command is currently a term which is not a free variable.  (term_typing:nonvar/1)
  Events is a free variable.  (term_typing:var/1)

**tclInterpreter/1:**  
**REGTYPE**

**Usage:** tclInterpreter(I)

- **Description:** I is a reference to a Tcl interpreter.
tclCommand/1: REGTYPE
   Usage: tclCommand(C)
   
   – Description: C is a Tcl command.

tk_event_loop/1: PREDICATE
   Usage: tk_event_loop(TclInterpreter)
   
   – Description: Waits for an event and executes the goal associated to it. Events are
   stored from Tcl with the prolog command. The unified term is sent to the Tcl
   interpreter in order to obtain the value of the tcl array of prolog_variables. If the
   term received does not have the form execute(Goal), the predicate silently exits. If
   the execution of Goal raises a Prolog error, the interpreter is deleted and an error
   message is given.
   
   – Call and exit should be compatible with:
     TclInterpreter is a reference to a Tcl interpreter. (tcltk:tclInterpreter/1)
   
   – The following properties should hold at call time:
     TclInterpreter is currently a term which is not a free variable. (term_
     typing:nonvar/1)

tk_main_loop/1: PREDICATE
   Usage: tk_main_loop(TclInterpreter)
   
   – Description: Passes control to Tk until all windows are gone.
   
   – Call and exit should be compatible with:
     TclInterpreter is a reference to a Tcl interpreter. (tcltk:tclInterpreter/1)
   
   – The following properties should hold at call time:
     TclInterpreter is currently a term which is not a free variable. (term_
     typing:nonvar/1)

tk_new/2: PREDICATE
   Usage: tk_new(Options, TclInterpreter)
   
   – Description: Performs basic Tcl and Tk initialization and creates the main window
   of a Tk application. Options is a list of optional elements according to:
   
   name(+ApplicationName)
   Sets the Tk main window title to ApplicationName. It is also used for
   communicating between Tcl/Tk applications via the Tcl send command.
   Default name is an empty string.
   
   display(+Display)
   Gives the name of the screen on which to create the main window. Default
   is normally determined by the DISPLAY environment variable.
   
   file
   Opens the script file. Commands will not be read from standard input
   and the execution returns back to Prolog only after all windows (and the
   interpreter) have been deleted.
   
   – Call and exit should be compatible with:
     Options is a list. (basic_props:list/1)
     TclInterpreter is a reference to a Tcl interpreter. (tcltk:tclInterpreter/1)
The following properties should hold at call time:

Options is currently a term which is not a free variable.  (term_typing:nonvar/1)
TclInterpreter is a free variable.  (term_typing:var/1)

---

**tk_next_event/2:**

**Usage:** tk_next_event(TclInterpreter, Event)

**Description:** Processes events until there is at least one Prolog event associated with TclInterpreter. Event is the term corresponding to the head of a queue of events stored from Tcl with the prolog_event command.

**Call and exit should be compatible with:**

TclInterpreter is a reference to a Tcl interpreter.  (tcltk:tclInterpreter/1)
Event is a string (a list of character codes).  (basic_props:string/1)

**The following properties should hold at call time:**

TclInterpreter is currently a term which is not a free variable.  (term_typing:nonvar/1)
Event is a free variable.  (term_typing:var/1)
116 Low level interface library to Tcl/Tk

Author(s): Montse Iglesias Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#315 (2004/2/25, 18:28:19 CET)

The `tcltk_low_level` library defines the low level interface used by the `tcltk` library. Essentially it includes all the code related directly to the handling of sockets and processes. This library should normally not be used directly by user programs, which use `tcltk` instead. On the other hand in some cases it may be useful to understand how this library works in order to understand possible problems in programs that use the `tcltk` library.

116.1 Usage and interface (`tcltk_low_level`)  

- **Library usage:**  
  ```prolog
  :- use_module(library(tcltk_low_level)).
  ```

- **Exports:**
  - **Predicates:**
    - `new_interp/1, new_interp/2, new_interp_file/2, tcltk/2, tcltk_raw_code/2, receive_result/2, send_term/2, receive_event/2, receive_list/2, receiveconfirm/2, delete/1`.
  - **Other modules used:**
    - **System library modules:**
      - terms, sockets/sockets, system, write, read, strings, lists, format.

116.2 Documentation on exports (`tcltk_low_level`)  

**new_interp/1:**

**Usage:** `new_interp(TclInterpreter)`

- **Description:** Creates two sockets to connect to the `wish` process, the term socket and the event socket, and opens a pipe to process `wish` in a new shell.
- **Call and exit should be compatible with:**
  - `TclInterpreter` is a reference to a `Tcl` interpreter. (tcltk_low_level:tclInterpreter/1)
  - **The following properties should hold at call time:**
    - `TclInterpreter` is a free variable. (term_typing:var/1)

**new_interp/2:**

**Usage:** `new_interp(TclInterpreter, Options)`

- **Description:** Creates two sockets, the term socket and the event socket, and opens a pipe to process `wish` in a new shell invoked with the `Options`.
- **Call and exit should be compatible with:**
  - `TclInterpreter` is a reference to a `Tcl` interpreter. (tcltk_low_level:tclInterpreter/1)
  - `Options` is currently instantiated to an atom. (term_typing:atom/1)
The following properties should hold at call time:

TclInterpreter is a free variable. \((term\_typing:var/1)\)

Options is currently a term which is not a free variable. \((term\_typing:nonvar/1)\)

**new_interp_file/2:**

Usage: `new_interp_file(FileN\_ame, Tcl\_Interpreter)`

Description: Creates two sockets, the term socket and the event socket, and opens a pipe to process `wish` in a new shell invoked with a `FileName`. `FileName` is treated as a name of a script file.

Call and exit should be compatible with:

`FileName` is a string (a list of character codes). \((basic\_props:string/1)\)

TclInterpreter is a reference to a `Tcl` interpreter. \((tcltk\_low\_level:tcl\_Interpreter/1)\)

The following properties should hold at call time:

`FileName` is currently a term which is not a free variable. \((term\_typing:nonvar/1)\)

TclInterpreter is a free variable. \((term\_typing:var/1)\)

**tcltk/2:**

Usage: `tcltk(Code, Tcl\_Interpreter)`

Description: Sends the `Code` converted to string to the `Tcl\_Interpreter`.

Call and exit should be compatible with:

`Code` is a `Tcl` command. \((tcltk\_low\_level:tcl\_Command/1)\)

TclInterpreter is a reference to a `Tcl` interpreter. \((tcltk\_low\_level:tcl\_Interpreter/1)\)

The following properties should hold at call time:

`Code` is currently a term which is not a free variable. \((term\_typing:nonvar/1)\)

TclInterpreter is currently a term which is not a free variable. \((term\_typing:nonvar/1)\)

**tcltk\_raw\_code/2:**

Usage: `tcltk\_raw\_code(String, Tcl\_Interpreter)`

Description: Sends the tcltk code items of the `Stream` to the `Tcl\_Interpreter`.

Call and exit should be compatible with:

`String` is a string (a list of character codes). \((basic\_props:string/1)\)

TclInterpreter is a reference to a `Tcl` interpreter. \((tcltk\_low\_level:tcl\_Interpreter/1)\)

The following properties should hold at call time:

`String` is currently a term which is not a free variable. \((term\_typing:nonvar/1)\)

TclInterpreter is currently a term which is not a free variable. \((term\_typing:nonvar/1)\)
receive_result/2:

Usage: receive_result(Result, TclInterpreter)

- Description: Receives the Result of the last TclCommand into the TclInterpreter. If the TclCommand is not correct the wish process is terminated and a message appears showing the error.

- Call and exit should be compatible with:
  - Result is a string (a list of character codes). (basic_props:string/1)
  - TclInterpreter is a reference to a Tcl interpreter. (tcltk_low_level:tclInterpreter/1)

- The following properties should hold at call time:
  - Result is a free variable. (term_typing:var/1)
  - TclInterpreter is currently a term which is not a free variable. (term_typing:nonvar/1)

send_term/2:

Usage: send_term(String, TclInterpreter)

- Description: Sends the goal executed to the TclInterpreter. String has the predicate with unified variables.

- Call and exit should be compatible with:
  - String is a string (a list of character codes). (basic_props:string/1)
  - TclInterpreter is a reference to a Tcl interpreter. (tcltk_low_level:tclInterpreter/1)

- The following properties should hold at call time:
  - String is currently a term which is not a free variable. (term_typing:nonvar/1)
  - TclInterpreter is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_event/2:

Usage: receive_event(Event, TclInterpreter)

- Description: Receives the Event from the event socket of the TclInterpreter.

- Call and exit should be compatible with:
  - Event is a list. (basic_props:list/1)
  - TclInterpreter is a reference to a Tcl interpreter. (tcltk_low_level:tclInterpreter/1)

- The following properties should hold at call time:
  - Event is a free variable. (term_typing:var/1)
  - TclInterpreter is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_list/2:

Usage: receive_list(List, TclInterpreter)

- Description: Receives the List from the event socket of the TclInterpreter. The List has all the predicates that have been inserted from Tcl/Tk with the command prolog_event. It is a list of terms.
Call and exit should be compatible with:

- List is a list. (basic_props:list/1)
- TclInterpreter is a reference to a Tcl interpreter. (tcltk_low_level:tclInterpreter/1)

The following properties should hold at call time:

- List is a free variable. (term_typing:var/1)
- TclInterpreter is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_confirm/2:

**Usage:** receive_confirm(String, TclInterpreter)

- **Description:** Receives the String from the event socket of the TclInterpreter when a term inserted into the event queue is managed.

Call and exit should be compatible with:

- String is a string (a list of character codes). (basic_props:string/1)
- TclInterpreter is a reference to a Tcl interpreter. (tcltk_low_level:tclInterpreter/1)

The following properties should hold at call time:

- String is a free variable. (term_typing:var/1)
- TclInterpreter is currently a term which is not a free variable. (term_typing:nonvar/1)

delete/1:

**Usage:** delete(TclInterpreter)

- **Description:** Terminates the wish process and closes the pipe, term socket and event socket. Deletes the interpreter TclInterpreter from the system

Call and exit should be compatible with:

- TclInterpreter is a reference to a Tcl interpreter. (tcltk_low_level:tclInterpreter/1)

The following properties should hold at call time:

- TclInterpreter is currently a term which is not a free variable. (term_typing:nonvar/1)

116.3 Documentation on internals (tcltk_low_level)

core/1:

**Usage:** core(String)

- **Description:** core/1 is a set of facts which contain Strings to be sent to the Tcl/Tk interpreter on startup. They implement miscellaneous Tcl/Tk procedures which are used by the Tcl/Tk interface.

Call and exit should be compatible with:

- String is a string (a list of character codes). (basic_props:string/1)

The following properties should hold at call time:

- String is currently a term which is not a free variable. (term_typing:nonvar/1)
117 The Tcl/Tk Class Interface


Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#152 (2003/12/4, 17:35:17 CET)

This library implements an object-oriented graphical library with a number of predefined objects, using the Prolog Tcl/Tk interface. This interface allows creating and destroying objects and modifying their properties. The window_class contains three classes: widget class, menu class, and canvas class. The constructor class is window_class.

Note: This library (and the documentation) are still under development.

117.1 Usage and interface (window_class)

- Library usage:
  :- use_module(library(window_class)).

- Other modules used:
  - System library modules:
    class/class_rt, class/virtual, objects/objects_rt, system, strings, lists, tcltk/tcltk, tcltk/tcltk_low_level, aggregates.

117.2 Documentation on multiforms (window_class)

$class$/1: PREDICATE
No further documentation available for this predicate.
The predicate is multifile.

class$super/2: PREDICATE
No further documentation available for this predicate.
The predicate is multifile.

class$initial_state/3: PREDICATE
No further documentation available for this predicate.
The predicate is multifile.

class$virtual/6: PREDICATE
No further documentation available for this predicate.
The predicate is multifile.
class$attr_template/4:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$default_cons/1:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$constructor/4:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$destructor/3:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$implements/2:
   No further documentation available for this predicate.
   The predicate is *multifile*.
118 widget_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#150 (2003/12/4, 17:35:4 CET)

118.1 Usage and interface (widget_class)

- Library usage:
  :- use_module(library(widget_class)).
- Exports:
  - Predicates:
    text_characters/1, font_type/1, background_color/1, borderwidth_value/1,
    foreground_color/1, highlightbackground_color/1, highlight_color/1, width_value/1,
    relief_type/1, side_type/1, expand_value/1, fill_type/1, padx_value/1, pady_value/1,
    row_value/1, rowspan_value/1, column_value/1, columnspan_value/1,
    event_type_widget/1, action_widget/3, action_widget/1, creation_options/1,
    creation_position/1, creation_position_grid/1, creation_bind/1.
- Other modules used:
  - System library modules:
    objects/objects_rt.

118.2 Documentation on exports (widget_class)

text_characters/1:
  Usage 1: text_characters(Text)
  Description: Indicates the Text to be displayed in the widget.
  Call and exit should be compatible with:
  Text is currently instantiated to an atom. (term_typing:atom/1)
  The following properties should hold at call time:
  Text is currently a term which is not a free variable. (term_typing:nonvar/1)
  Usage 2: text_characters(Text)
  Description: Text which is displayed in the widget.
  Call and exit should be compatible with:
  Text is currently instantiated to an atom. (term_typing:atom/1)
  The following properties should hold at call time:
  Text is a free variable. (term_typing:var/1)

font_type/1:
  Usage 1: font_type(Font)
  Description: Indicates the Font of the widget’s text.
Call and exit should be compatible with:
Font is currently instantiated to an atom. (term_typing:atom/1)
The following properties should hold at call time:
Font is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: font_type(Font)
Description: Gets the Font of the widget’s text.
Call and exit should be compatible with:
Font is currently instantiated to an atom. (term_typing:atom/1)
The following properties should hold at call time:
Font is a free variable. (term_typing:var/1)

background_color/1:
Usage 1: background_color(Background)
Description: Indicates the Background color. Default to gray.
Call and exit should be compatible with:
Background is currently instantiated to an atom. (term_typing:atom/1)
The following properties should hold at call time:
Background is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: background_color(Background)
Description: Returns the Background color.
Call and exit should be compatible with:
Background is currently instantiated to an atom. (term_typing:atom/1)
The following properties should hold at call time:
Background is a free variable. (term_typing:var/1)

borderwidth_value/1:
Usage 1: borderwidth_value(BorderWidth)
Description: Indicates the width’s border. Default to 2.
Call and exit should be compatible with:
BorderWidth is a number. (basic_props:num/1)
The following properties should hold at call time:
BorderWidth is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: borderwidth_value(BorderWidth)
Description: Gets the width’s border.
Call and exit should be compatible with:
BorderWidth is currently instantiated to an atom. (term_typing:atom/1)
The following properties should hold at call time:
BorderWidth is a free variable. (term_typing:var/1)
foreground_color/1:
Usage 1: foreground_color(Foreground)
  - Description: Indicates the Foreground color. Default to black
  - Call and exit should be compatible with:
    Foreground is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Foreground is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: foreground_color(Foreground)
  - Description: Gets the Foreground color.
  - Call and exit should be compatible with:
    Foreground is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Foreground is a free variable. (term_typing:var/1)

highlightbackground_color/1:
Usage 1: highlightbackground_color(Color)
  - Description: Color specifies the highlight background color. Default to white
  - Call and exit should be compatible with:
    Color is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Color is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: highlightbackground_color(Color)
  - Description: Gets the Color of the highlight background.
  - Call and exit should be compatible with:
    Color is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Color is a free variable. (term_typing:var/1)

highlight_color/1:
Usage 1: highlight_color(Color)
  - Description: Color specifies the highlight color. Default to white
  - Call and exit should be compatible with:
    Color is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Color is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: highlight_color(Color)
  - Description: Gets the Color of the highlight.
  - Call and exit should be compatible with:
    Color is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Color is a free variable. (term_typing:var/1)
**width_value/1:**

**Usage 1:** width_value(Width)
- **Description:** Specifies the Width for the widget. Default to 0
- **Call and exit should be compatible with:**
  Width is an integer. (basic_props:int/1)
- **The following properties should hold at call time:**
  Width is currently a term which is not a free variable. (term_typing:nonvar/1)

**Usage 2:** width_value(Width)
- **Description:** Gets the Width specified for the widget.
- **Call and exit should be compatible with:**
  Width is an integer. (basic_props:int/1)
- **The following properties should hold at call time:**
  Width is currently a term which is not a free variable. (term_typing:nonvar/1)

**relief_type/1:**

**Usage 1:** relief_type(Relief)
- **Description:** Specifies a desired Relief for the widget. Default to sunken
- **Call and exit should be compatible with:**
  Relief is currently instantiated to an atom. (term_typing:atom/1)
- **The following properties should hold at call time:**
  Relief is currently a term which is not a free variable. (term_typing:nonvar/1)

**Usage 2:** relief_type(Relief)
- **Description:** Gets the Relief of the widget.
- **Call and exit should be compatible with:**
  Relief is currently instantiated to an atom. (term_typing:atom/1)
- **The following properties should hold at call time:**
  Relief is a free variable. (term_typing:var/1)

**side_type/1:**

**Usage 1:** side_type(Side)
- **Description:** Specifies which Side of the master, the slave(s) will be packed against. Must be left, right, top or bottom. Defaults to top
- **Call and exit should be compatible with:**
  Side is currently instantiated to an atom. (term_typing:atom/1)
- **The following properties should hold at call time:**
  Side is currently a term which is not a free variable. (term_typing:nonvar/1)

**Usage 2:** side_type(Side)
- **Description:** Gets the position of the canvas.
- **Call and exit should be compatible with:**
  Side is currently instantiated to an atom. (term_typing:atom/1)
- **The following properties should hold at call time:**
  Side is a free variable. (term_typing:var/1)
expand_value/1:
Usage 1: expand_value(Value)
  – *Description*: Specifies whether the slaves should be expanded to consume extra space in their master. *Value* may have any proper boolean value, such as 1 or 0. Defaults to 0
  – *Call and exit should be compatible with*:
    Value is an integer. (basic_props:int/1)
  – *The following properties should hold at call time*:
    Value is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: expand_value(Value)
  – *Description*: Gets the boolean value which indicates if the slaves should be expanded or no.
  – *Call and exit should be compatible with*:
    Value is an integer. (basic_props:int/1)
  – *The following properties should hold at call time*:
    Value is a free variable. (term_typing:var/1)

fill_type/1:
Usage 1: fill_type(Option)
  – *Description*: If a slave’s parcel is larger than its requested dimensions, this option may be used to stretch the slave. *Option* must have one of the following values: none (this is the default), x, y, both
  – *Call and exit should be compatible with*:
    Option is currently instantiated to an atom. (term_typing:atom/1)
  – *The following properties should hold at call time*:
    Option is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: fill_type(Option)
  – *Description*: Gets the fill value of the canvas
  – *Call and exit should be compatible with*:
    Option is currently instantiated to an atom. (term_typing:atom/1)
  – *The following properties should hold at call time*:
    Option is a free variable. (term_typing:var/1)

padx_value/1:
Usage 1: padx_value(Amount)
  – *Description*: *Amount* specifies how much horizontal external padding to leave on each side of the slave(s). Amount defaults to 0
  – *Call and exit should be compatible with*:
    Amount is an integer. (basic_props:int/1)
  – *The following properties should hold at call time*:
    Amount is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: padx_value(Amount)
Description: Gets the **Amount** which specifies how much horizontal external padding to leave on each side of the slaves.

Call and exit should be compatible with:

**Amount** is an integer.  

The following properties should hold at call time:

**Amount** is a free variable.

---

**pady_value/1:** PREDICATE

Usage 1: `pady_value(Amount)`

Description: **Amount** specifies how much vertical external padding to leave on each side of the slave(s). **Amount** defaults to 0.

Call and exit should be compatible with:

**Amount** is an integer.  

The following properties should hold at call time:

**Amount** is currently a term which is not a free variable.

Usage 2: `pady_value(Amount)`

Description: Gets the **Amount** which specifies how much vertical external padding to leave on each side of the slaves.

Call and exit should be compatible with:

**Amount** is an integer.  

The following properties should hold at call time:

**Amount** is a free variable.

---

**row_value/1:** PREDICATE

Usage 1: `row_value(Row)`

Description: Indicates the **Row** in which the widget should be allocated.

Call and exit should be compatible with:

**Row** is an integer.  

The following properties should hold at call time:

**Row** is currently a term which is not a free variable.

Usage 2: `row_value(Row)`

Description: Gets the **Row** in which the widget is allocated.

Call and exit should be compatible with:

**Row** is an integer.  

The following properties should hold at call time:

**Row** is a free variable.

---

**rowspan_value/1:** PREDICATE

Usage 1: `rowspan_value(Row)`

Description: Indicates the number of **Row** which are going to be occupied in the grid.

Call and exit should be compatible with:

**Row** is an integer.
The following properties should hold at call time:
Row is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: rowspan_value(Row)
- Description: Gets the number of Row which are occupied by the widget in the grid.
- Call and exit should be compatible with:
  Row is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Row is a free variable. (term_typing:var/1)

column_value/1:  
Usage 1: column_value(Column)
- Description: Indicates the Column in which the widget should be allocated.
- Call and exit should be compatible with:
  Column is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Column is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: column_value(Column)
- Description: Gets the Column in which the widget is allocated.
- Call and exit should be compatible with:
  Column is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Column is a free variable. (term_typing:var/1)

columnspan_value/1:  
Usage 1: columnspan_value(Column)
- Description: Indicates the number of Column which are going to be occupied in the grid.
- Call and exit should be compatible with:
  Column is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Column is currently a term which is not a free variable. (term_typing:nonvar/1)
Usage 2: columnspan_value(Column)
- Description: Gets the number of Column which are occupied by the widget in the grid.
- Call and exit should be compatible with:
  Column is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  Column is a free variable. (term_typing:var/1)
event_type_widget/1:

Usage 1: event_type_widget(EventType)

- Description: The event EventType is going to be manage by the interface.

- Call and exit should be compatible with:
  - EventType is currently instantiated to an atom. (term_typing:atom/1)

- The following properties should hold at call time:

Usage 2: event_type_widget(EventType)

- Description: Gets the event EventType which is going to be manage by the interface.

- Call and exit should be compatible with:
  - EventType is currently instantiated to an atom. (term_typing:atom/1)

- The following properties should hold at call time:

action_widget/3:

Usage 1: action_widget(Input, Output, Term)

- Description: Executes Term with Input value and Output variable.

- Call and exit should be compatible with:
  - Input is currently instantiated to an atom. (term_typing:atom/1)
  - Output is currently instantiated to an atom. (term_typing:atom/1)
  - Term is currently instantiated to an atom. (term_typing:atom/1)

- The following properties should hold at call time:
  - Input is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Output is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Term is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: action_widget(Input, Output, Term)

- Description: Term is associated to the action of the object indicated with the operation event_type_widget.

- Call and exit should be compatible with:
  - Input is currently instantiated to an atom. (term_typing:atom/1)
  - Output is currently instantiated to an atom. (term_typing:atom/1)
  - Term is currently instantiated to an atom. (term_typing:atom/1)

- The following properties should hold at call time:
  - Input is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Output is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Term is a free variable. (term_typing:var/1)

action_widget/1:

Usage 1: action_widget(Term)

- Description: Term is going to be associated to the action of the object indicated with the operacion event_type_widget.
– **Call and exit should be compatible with:**
  Term is currently instantiated to an atom. \((\text{term\_typing:atom/1})\)

– **The following properties should hold at call time:**
  Term is currently a term which is not a free variable. \((\text{term\_typing:nonvar/1})\)

**Usage 2: action_widget(Term)**

– **Description:** Term is associated to the action of the object indicated with the operation `event\_type\_widget`.
– **Call and exit should be compatible with:**
  Term is currently instantiated to an atom. \((\text{term\_typing:atom/1})\)

– **The following properties should hold at call time:**
  Term is a free variable. \((\text{term\_typing:var/1})\)

**creation_options/1:**

**PREDICATE**

**Usage:** `creation_options(OptionsList)`

– **Description:** Creates a list with the options supported by the widget.
– **Call and exit should be compatible with:**
  `OptionsList` is a list. \((\text{basic\_props:list/1})\)

– **The following properties should hold at call time:**
  `OptionsList` is a free variable. \((\text{term\_typing:var/1})\)

**creation_position/1:**

**PREDICATE**

**Usage:** `creation_position(OptionsList)`

– **Description:** Creates a list with the options supported by the pack command.
– **Call and exit should be compatible with:**
  `OptionsList` is a list. \((\text{basic\_props:list/1})\)

– **The following properties should hold at call time:**
  `OptionsList` is a free variable. \((\text{term\_typing:var/1})\)

**creation_position_grid/1:**

**PREDICATE**

**Usage:** `creation_position_grid(OptionsList)`

– **Description:** Creates a list with the options supported by the grid command.
– **Call and exit should be compatible with:**
  `OptionsList` is a list. \((\text{basic\_props:list/1})\)

– **The following properties should hold at call time:**
  `OptionsList` is a free variable. \((\text{term\_typing:var/1})\)

**creation_bind/1:**

**PREDICATE**

**Usage:** `creation_bind(BindList)`

– **Description:** Creates a list with the event to be manage and the action associated to this event.
- *Call and exit should be compatible with:*
  - `BindList` is a list. 
    - (basic.props:list/1)

- *The following properties should hold at call time:*
  - `BindList` is a free variable.
    - (term_typing:var/1)
119 menu_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#133 (2003/12/4, 17:33:46 CET)

119.1 Usage and interface (menu_class)

- Library usage:
  `- use_module(library(menu_class)).`  
- Exports:
  - Predicates:
    - `name_menu/1, menu_data/1, label_value/1, tearoff_value/1, tcl_name/1, creation_options/1, creation_options_entry/1, creation_menu_name/1`.  
- Other modules used:
  - System library modules:
    - `objects/objects_rt, tcltk_obj/window_class, tcltk_obj/menu_entry_class, tcltk/tcltk, tcltk/tcltk_low_level, lists`.

119.2 Documentation on exports (menu_class)

`name_menu/1:`  
**PREDICATE**  
**Usage:** `name_menu(Name)`  
- **Description:** Indicates the Name of the menubutton associated.  
- **Call and exit should be compatible with:**  
  `Name` is currently instantiated to an atom.  
- **The following properties should hold at call time:**  
  `Name` is currently a term which is not a free variable.  

`menu_data/1:`  
**PREDICATE**  
**Usage 1:** `menu_data(Menu)`  
- **Description:** Menu posted when cascade entry is invoked.  
- **Call and exit should be compatible with:**  
  `Menu` is currently instantiated to an atom.  
- **The following properties should hold at call time:**  
  `Menu` is currently a term which is not a free variable.  

**Usage 2:** `menu_data(Menu)`  
- **Description:** Gets the Menu associated to the cascade entry.  
- **Call and exit should be compatible with:**  
  `Menu` is currently instantiated to an atom.  
- **The following properties should hold at call time:**  
  `Menu` is a free variable.
label_value/1: PREDICATE
Usage 1: label_value(Value)
  - Description: Value specifies a string to be displayed as an identifying label in the menu entry.
  - Call and exit should be compatible with:
    Value is currently instantiated to an atom. (term_typing:atom/1)
    The following properties should hold at call time:
    Value is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: label_value(Value)
  - Description: Gets the string which identify the menu entry.
  - Call and exit should be compatible with:
    Value is currently instantiated to an atom. (term_typing:atom/1)
    The following properties should hold at call time:
    Value is a free variable. (term_typing:var/1)

tearoff_value/1: PREDICATE
Usage 1: tearoff_value(Tearoff)
  - Description: Tearoff must have a proper boolean value, which specifies whether or not the menu should include a tear-off entry at the top. Defaults to 1.
  - Call and exit should be compatible with:
    Tearoff is currently instantiated to an atom. (term_typing:atom/1)
    The following properties should hold at call time:
    Tearoff is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: tearoff_value(Tearoff)
  - Description: Gets the Tearoff value
  - Call and exit should be compatible with:
    Tearoff is currently instantiated to an atom. (term_typing:atom/1)
    The following properties should hold at call time:
    Tearoff is a free variable. (term_typing:var/1)

tcl_name/1: PREDICATE
Usage: tcl_name(Widget)
  - Description: Specifies the name of the Widget. In this case is menu.
  - Call and exit should be compatible with:
    Widget is currently instantiated to an atom. (term_typing:atom/1)
    The following properties should hold at call time:
    Widget is a free variable. (term_typing:var/1)

creation_options/1: PREDICATE
Usage: creation_options(OptionsList)
  - Description: Creates a list with the options supported by the menu.
Call and exit should be compatible with:
OptionsList is a list. (basic_props:list/1)

The following properties should hold at call time:
OptionsList is a free variable. (term_typing:var/1)

creation_options_entry/1:
Usage: creation_options_entry(OptionsList)
Description: Creates a list with the options of the menu entry.
Call and exit should be compatible with:
OptionsList is a list. (basic_props:list/1)
The following properties should hold at call time:
OptionsList is a free variable. (term_typing:var/1)

creation_menu_name/1:
Usage: creation_menu_name(OptionsList)
Description: Creates a list with the name of the menu.
Call and exit should be compatible with:
OptionsList is a list. (basic_props:list/1)
The following properties should hold at call time:
OptionsList is a free variable. (term_typing:var/1)
120 canvas_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#123 (2003/12/4, 17:32:57 CET)

120.1 Usage and interface (canvas_class)

- Library usage:
  ```prolog
  :- use_module(library(canvas_class)).
  ```
- Exports:
  - Predicates:
    ```prolog
    canvas_class/0, canvas_class/1, destructor/0, show/0, hide_/0, show_shape/1, 
    hide_shape/1, width_value/1, height_value/1, side_type/1, expand_value/1, 
    fill_type/1, padx_value/1, pady_value/1.
    ```
  - Regular Types:
    ```prolog
    shape/1, option/1.
    ```
  - Multi-files:
    ```prolog
    $class$/1, class$super/2, class$initial_ 
    state/3, class$virtual/6, class$attr_template/4, class$default_cons/1, 
    class$constructor/4, class$destructor/3, class$implements/2.
    ```
- Other modules used:
  - System library modules:
    class/class_rt, class/virtual, objects/objects_rt, tcltk_obj/window_class, 
    tcltk_obj/shape_class, system, strings, lists, tcltk/tcltk, tcltk/tcltk_low_ 
    level.

120.2 Documentation on exports (canvas_class)

shape/1: REGTYPE

Each item type is characterized in two ways: first, the form of the create command used 
to create instances of the type; and second, a set of configuration options for items of that 
type, which may be used in the create and itemconfigure widget commands.

Usage: shape(S)
- Description: S is a reference to one type of the items supported by canvas widgets.

option/1: REGTYPE

Usage: option(0)
- Description: 0 is hidden if the Shape is not visible or shown if its visible.

canvas_class/0: PREDICATE

Usage:
– *Description*: Creates a new interpreter, asserting the predicate `interp(I)` and the canvas object.

**canvas_class/1:**

**Usage:** `canvas_class(ItemsList)`

– *Description*: Adds items of the list to the canvas object.

– *Call and exit should be compatible with:*

  `ItemsList` is a list.  

– *The following properties should hold at call time:*

  `ItemsList` is currently a term which is not a free variable.

**Destructor/0:**

**Usage:**

– *Description*: Deletes the shapes of the canvas object and the object.

**Show/0:**

**Usage:**

– *Description*: Adds shapes to the canvas object.

**Hide_/0:**

**Usage:**

– *Description*: Hides shapes from the canvas object.

**Show_shape/1:**

**Usage:** `show_shape(Shape)`

– *Description*: Shows `Shape` into the canvas object.

– *Call and exit should be compatible with:*

  `Shape` is a reference to one type of the items supported by canvas widgets.

– *The following properties should hold at call time:*

  `Shape` is currently a term which is not a free variable.

**Hide_shape/1:**

**Usage:** `hide_shape(Shape)`

– *Description*: Hides `Shape` from the canvas object.

– *Call and exit should be compatible with:*

  `Shape` is a reference to one type of the items supported by canvas widgets.

– *The following properties should hold at call time:*

  `Shape` is currently a term which is not a free variable.
width_value/1:
    
    Usage 1: width_value(Width)
    - Description: Sets the Width of the canvas. Default 100.
    - Call and exit should be compatible with:
      Width is a number. (basic_props:num/1)
    - The following properties should hold at call time:
      Width is currently a term which is not a free variable. (term_typing:nonvar/1)

    Usage 2: width_value(Width)
    - Description: Gets the Width of the canvas.
    - Call and exit should be compatible with:
      Width is a number. (basic_props:num/1)
    - The following properties should hold at call time:
      Width is a free variable. (term_typing:var/1)

height_value/1:
    
    Usage 1: height_value(Height)
    - Description: Sets the Height of the canvas. Default 50.
    - Call and exit should be compatible with:
      Height is a number. (basic_props:num/1)
    - The following properties should hold at call time:
      Height is currently a term which is not a free variable. (term_typing:nonvar/1)

    Usage 2: height_value(Height)
    - Description: Gets the Height of the canvas.
    - Call and exit should be compatible with:
      Height is a number. (basic_props:num/1)
    - The following properties should hold at call time:
      Height is a free variable. (term_typing:var/1)

side_type/1:
    
    Usage 1: side_type(Side)
    - Description: Specifies which Side of the master, the slave(s) will be packed against.
      Must be left, right, top or bottom. Defaults to top.
    - Call and exit should be compatible with:
      Side is currently instantiated to an atom. (term_typing:atom/1)
    - The following properties should hold at call time:
      Side is currently a term which is not a free variable. (term_typing:nonvar/1)

    Usage 2: side_type(Side)
    - Description: Gets the Side of the canvas.
    - Call and exit should be compatible with:
      Side is currently instantiated to an atom. (term_typing:atom/1)
    - The following properties should hold at call time:
      Side is a free variable. (term_typing:var/1)
expand_value/1:
Usage 1: expand_value(Value)

  Description: Specifies whether the slaves should be expanded to consume extra space in their master. Value may have any proper boolean value, such as 1 or no. Defaults to 0.

  Call and exit should be compatible with:
  Value is currently instantiated to an atom. (term_typing:atom/1)

Usage 2: expand_value(Value)

  Description: Gets the boolean Value which indicates if the slaves should be expanded or no.

  Call and exit should be compatible with:
  Value is currently instantiated to an atom. (term_typing:atom/1)

fill_type/1:
Usage 1: fill_type(Option)

  Description: If a slave’s parcel is larger than its requested dimensions, this option may be used to stretch the slave. Style must have one of the following values: none (this is the default), x, y, both.

  Call and exit should be compatible with:
  Option is currently instantiated to an atom. (term_typing:atom/1)

Usage 2: fill_type(Option)

  Description: Gets the fill value of the canvas

  Call and exit should be compatible with:
  Option is currently instantiated to an atom. (term_typing:atom/1)

padx_value/1:
Usage 1: padx_value(Amount)

  Description: Amount specifies how much horizontal external padding to leave on each side of the slave(s). Amount defaults to 0.

  Call and exit should be compatible with:
  Amount is currently instantiated to an atom. (term_typing:atom/1)

Usage 2: padx_value(Amount)
– **Description:** Gets the `Amount` which specifies how much horizontal external padding to leave on each side of the slaves.

– **Call and exit should be compatible with:**
  
  `Amount` is currently instantiated to an atom.  
  
  (term_typing:atom/1)

– **The following properties should hold at call time:**
  
  `Amount` is a free variable.
  
  (term_typing:var/1)

### pady_value/1:

**Usage 1:** `pady_value(Amount)`

– **Description:** `Amount` specifies how much vertical external padding to leave on each side of the slave(s). `Amount` defaults to 0.

– **Call and exit should be compatible with:**
  
  `Amount` is currently instantiated to an atom.  
  
  (term_typing:atom/1)

– **The following properties should hold at call time:**
  
  `Amount` is currently a term which is not a free variable.  
  
  (term_typing:nonvar/1)

**Usage 2:** `pady_value(Amount)`

– **Description:** Gets the `Amount` which specifies how much vertical external padding to leave on each side of the slaves.

– **Call and exit should be compatible with:**
  
  `Amount` is currently instantiated to an atom.  
  
  (term_typing:atom/1)

– **The following properties should hold at call time:**
  
  `Amount` is a free variable.  
  
  (term_typing:var/1)

### 120.3 Documentation on multifiles (canvas_class)

#### $class$/1:

**PREDICATE**

No further documentation available for this predicate.

The predicate is *multifile*.

#### class$super/2:

**PREDICATE**

No further documentation available for this predicate.

The predicate is *multifile*.

#### class$initial_state/3:

**PREDICATE**

No further documentation available for this predicate.

The predicate is *multifile*.

#### class$virtual/6:

**PREDICATE**

No further documentation available for this predicate.

The predicate is *multifile*. 
class$attr_template/4:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$default_cons/1:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$constructor/4:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$destructor/3:
   No further documentation available for this predicate.
   The predicate is *multifile*.

class$implements/2:
   No further documentation available for this predicate.
   The predicate is *multifile*. 
121 button_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#122 (2003/12/4, 17:32:36 CET)

121.1 Usage and interface (button_class)

- Library usage:
  
  ```
  :- use_module(library(button_class)).
  ```

- Exports:
  - Predicates:
    ```
    command_button/1.
    ```

- Other modules used:
  - System library modules:
    ```
    objects/objects_rt.
    ```

121.2 Documentation on exports (button_class)

command_button/1:  

**PREDICATE**

**Usage 1:** command_button(Command)

- **Description:** Sets a Tcl Command to be associated with the button. This Command is typically invoked when mouse button 1 is released over the button window.

- **Call and exit should be compatible with:**
  ```
  Command is currently instantiated to an atom.  
  ```

- **The following properties should hold at call time:**
  ```
  Command is currently a term which is not a free variable. 
  ```

**Usage 2:** command_button(Command)

- **Description:** Gets the Tcl Command associated with the button.

- **Call and exit should be compatible with:**
  ```
  Command is currently instantiated to an atom. 
  ```

- **The following properties should hold at call time:**
  ```
  Command is a free variable. 
  ```
122 checkbutton_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#125 (2003/12/4, 17:33:7 CET)

122.1 Usage and interface (checkbutton_class)

- Library usage:
  :- use_module(library(checkbutton_class)).
- Exports:
  - Predicates:
    variable_value/1.
- Other modules used:
  - System library modules:
    objects(objects_rt).

122.2 Documentation on exports (checkbutton_class)

variable_value/1:

Usage 1: variable_value(Variable)
  - Description: Sets the value of global Variable to indicate whether or not this button is selected.
  - Call and exit should be compatible with:
    Variable is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Variable is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: variable_value(Variable)
  - Description: Gets the value of global Variable which indicates if the button is selected.
  - Call and exit should be compatible with:
    Variable is currently instantiated to an atom. (term_typing:atom/1)
  - The following properties should hold at call time:
    Variable is a free variable. (term_typing:var/1)
Chapter 123: radiobutton_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#143 (2003/12/4, 17:34:38 CET)

123.1 Usage and interface (radiobutton_class)

- Library usage:
  :- use_module(library(radiobutton_class)).

- Exports:
  - Predicates:
    variable_value/1.

- Other modules used:
  - System library modules:
    objects/objects_rt.

123.2 Documentation on exports (radiobutton_class)

**variable_value/1:**

- **Usage 1:** variable_value(Variable)
  - **Description:** Specifies the value of global Variable to set whenever this button is selected.
  - **Call and exit should be compatible with:**
    Variable is currently instantiated to an atom. (term_typing:atom/1)
  - **The following properties should hold at call time:**
    Variable is currently a term which is not a free variable. (term_typing:nonvar/1)

- **Usage 2:** variable_value(Variable)
  - **Description:** Gets the value of global Variable which indicates if this button is selected.
  - **Call and exit should be compatible with:**
    Variable is currently instantiated to an atom. (term_typing:atom/1)
  - **The following properties should hold at call time:**
    Variable is a free variable. (term_typing:var/1)
124 entry_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#127 (2003/12/4, 17:33:22 CET)

124.1 Usage and interface (entry_class)

- Library usage:
  \[\text{:- use_module(library(entry_class)).}\]
- Exports:
  - Predicates:
    \[\text{textvariable_entry/1, textvariablevalue_string/1, textvariablevalue_number/1, justify_entry/1.}\]
- Other modules used:
  - System library modules:
    \[\text{objects/objects_rt, lists, tcltk/examples/tk_test_aux, tcltk/tcltk.}\]

124.2 Documentation on exports (entry_class)

textvariable_entry/1:  PREDICATE
Usage 1: textvariable_entry(Variable)
  - Description: Variable specifies the name of the Tcl variable
  - Call and exit should be compatible with:
    Variable is currently instantiated to an atom. \(\text{term_typing:atom/1}\)
  - The following properties should hold at call time:
    Variable is currently a term which is not a free variable. \(\text{term_typing:nonvar/1}\)
Usage 2: textvariable_entry(Variable)
  - Description: Gets the name of the Tcl Variable associated to the entry
  - Call and exit should be compatible with:
    Variable is currently instantiated to an atom. \(\text{term_typing:atom/1}\)
  - The following properties should hold at call time:
    Variable is a free variable. \(\text{term_typing:var/1}\)

textvariablevalue_string/1:  PREDICATE
Usage 1: textvariablevalue_string(Value)
  - Description: Specifies the Value of the Tcl variable associated to the entry.
  - Call and exit should be compatible with:
    Value is a number. \(\text{basic_props:num/1}\)
  - The following properties should hold at call time:
    Value is currently a term which is not a free variable. \(\text{term_typing:nonvar/1}\)
Usage 2: textvariablevalue_string(Value)
- Description: Value is the value of the Tcl variable associated to the entry.
- Call and exit should be compatible with:
  Value is a number.
  The following properties should hold at call time:
  Value is a free variable.

Usage 1: textvariablevalue_number(Value)
- Description: Specifies the Value of the Tcl variable associated to the entry.
- Call and exit should be compatible with:
  Value is a number.
  The following properties should hold at call time:
  Value is currently a term which is not a free variable.

Usage 2: textvariablevalue_number(Value)
- Description: Value is the value of the Tcl variable associated to the entry.
- Call and exit should be compatible with:
  Value is a number.
  The following properties should hold at call time:
  Value is a free variable.

justfy_entry/1:
Usage 1: justify_entry(How)
- Description: How specifies how to justify the text in the entry. How must be one of the values left, right or center. This option defaults to left.
- Call and exit should be compatible with:
  How is currently instantiated to an atom.
  The following properties should hold at call time:
  How is currently a term which is not a free variable.

Usage 2: justify_entry(How)
- Description: Gets How is justified the text.
- Call and exit should be compatible with:
  How is currently instantiated to an atom.
Chapter 125: label_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#129 (2003/12/4, 17:33:30 CET)

125.1 Usage and interface (label_class)

- Library usage:
  ```prolog```
  ```
  :- use_module(library(label_class)).
  ```
- Exports:
  - Predicates:
    ```prolog```
    ```
    textvariable_label/1.
    ```
- Other modules used:
  - System library modules:
    ```prolog```
    ```
    objects/objects_rt.
    ```

125.2 Documentation on exports (label_class)

textvariable_label/1:  
No further documentation available for this predicate.
126 menubutton_class (library)

Author(s): Montserrat Urraca.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#137 (2003/12/4, 17:34:7 CET)

126.1 Usage and interface (menubutton_class)

- Library usage:
  
  ```prolog
  :- use_module(library(menubutton_class)).
  ```

- Exports:
  
  - Predicates:
    
    ```prolog
    menu_name/1.
    ```

- Other modules used:
  
  - System library modules:
    
    ```prolog
    objects/objects_rt, lists.
    ```

126.2 Documentation on exports (menubutton_class)

**menu_name/1:**

**PREDICATE**

Usage 1: `menu_name(\text{Menu})`

- *Description:* Menu posted when menubutton is clicked.
- *Call and exit should be compatible with:*
  
  ```prolog
  Menu is currently instantiated to an atom. \text{(term_typing:atom/1)}
  ```

- *The following properties should hold at call time:*
  
  ```prolog
  Menu is currently a term which is not a free variable. \text{(term_typing:nonvar/1)}
  ```

Usage 2: `menu_name(\text{Menu})`

- *Description:* Gets the name of the \text{Menu} asociated to the menubutton.
- *Call and exit should be compatible with:*
  
  ```prolog
  Menu is currently instantiated to an atom. \text{(term_typing:atom/1)}
  ```

- *The following properties should hold at call time:*
  
  ```prolog
  Menu is a free variable. \text{(term_typing:var/1)}
  ```
127 menu_entry_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#135 (2003/12/4, 17:33:57 CET)

127.1 Usage and interface (menu_entry_class)

- Library usage:
  \[-\text{use\_module(library(menu\_entry\_class))}.\]
- Exports:
  - Predicates:
    set_name/1, set_action/1, label_value/1, menu_name/1.
- Other modules used:
  - System library modules:
    objects/objects\_rt, tcltk\_obj/menu\_class, lists.

127.2 Documentation on exports (menu_entry_class)

set_name/1: PREDICATE

Usage: set_name(Name)
- Description: Name of the menubutton associated.
- Call and exit should be compatible with:
  Name is currently instantiated to an atom. \(\text{(term\_typing:atom/1)}\)
- The following properties should hold at call time:
  Name is currently a term which is not a free variable. \(\text{(term\_typing:nonvar/1)}\)

set_action/1: PREDICATE

Usage: set_action(Predicate)
- Description: Specifies Predicate associated to the menu entry.
- Call and exit should be compatible with:
  Predicate is currently instantiated to an atom. \(\text{(term\_typing:atom/1)}\)
- The following properties should hold at call time:
  Predicate is currently a term which is not a free variable. \(\text{(term\_typing:nonvar/1)}\)

label_value/1: PREDICATE

Usage 1: label_value(Value)
- Description: Value specifies a value to be displayed as an identifying label in the menu entry.
- Call and exit should be compatible with:
  Value is currently instantiated to an atom. \(\text{(term\_typing:atom/1)}\)
The following properties should hold at call time:
Value is currently a term which is not a free variable.  

Usage 2: label_value(Value)
- Description: Gets the string which identify label in the menu entry.
- Call and exit should be compatible with:
  Value is currently instantiated to an atom.  
- The following properties should hold at call time:
  Value is a free variable.

menu_name/1:

Usage 1: menu_name(Menu)
- Description: Menu posted when cascade entry is invoked.
- Call and exit should be compatible with:
  Menu is currently instantiated to an atom.  
- The following properties should hold at call time:
  Menu is currently a term which is not a free variable.

Usage 2: menu_name(Menu)
- Description: Gets the Menu asociated to the cascade entry.
- Call and exit should be compatible with:
  Menu is currently instantiated to an atom.  
- The following properties should hold at call time:
  Menu is a free variable.
Chapter 128: shape_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#146 (2003/12/4, 17:34:49 CET)

128.1 Usage and interface (shape_class)

- Library usage:
  
  ```prolog
  :- use_module(library(shape_class)).
  ```

- Other modules used:
  
  System library modules:
  
  class/class_rt, class/virtual, objects/objects_rt, tcltk_obj/canvas_class.

128.2 Documentation on multifiles (shape_class)

$\text{class$/1:}$

No further documentation available for this predicate.
The predicate is \textit{multifile}.

\text{class$/super/2:}$

No further documentation available for this predicate.
The predicate is \textit{multifile}.

\text{class$/initial\_state/3:}$

No further documentation available for this predicate.
The predicate is \textit{multifile}.

\text{class$/virtual/6:}$

No further documentation available for this predicate.
The predicate is \textit{multifile}.

\text{class$/attr\_template/4:}$

No further documentation available for this predicate.
The predicate is \textit{multifile}.

\text{class$/default\_cons/1:}$

No further documentation available for this predicate.
The predicate is \textit{multifile}. 
class$constructor/4:
No further documentation available for this predicate.
The predicate is *multifile*.

class$destructor/3:
No further documentation available for this predicate.
The predicate is *multifile*.

class$implements/2:
No further documentation available for this predicate.
The predicate is *multifile*.
129 arc_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#120 (2003/12/4, 17:32:15 CET)

129.1 Usage and interface (arc_class)

- Library usage:
  ```prolog
  :- use_module(library(arc_class)).
  ```
- Exports:
  - Predicates:
    - coord/4, width/1, height/1, center/2, angle_start/1, style_type/1, outline_color/1.
- Other modules used:
  - System library modules:
    - objects/objects_rt.

129.2 Documentation on exports (arc_class)

coord/4: PREDICATE
Usage: coord(X1, Y1, X2, Y2)
- Description: X1, Y1, X2, and Y2 give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval that defines the arc.
- Call and exit should be compatible with:
  - X1 is an integer. (basic_props:int/1)
  - Y1 is an integer. (basic_props:int/1)
  - X2 is an integer. (basic_props:int/1)
  - Y2 is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  - X1 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Y1 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - X2 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Y2 is currently a term which is not a free variable. (term_typing:nonvar/1)

width/1: PREDICATE
Usage 1: width(Width)
- Description: Specifies shape’s Width.
- Call and exit should be compatible with:
  - Width is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  - Width is currently a term which is not a free variable. (term_typing:nonvar/1)
**Usage 2: width(Width)**

- *Description:* Gets shape’s Width.
- *Call and exit should be compatible with:*
  
  _Width_ is an integer.  

- *The following properties should hold at call time:*
  
  _Width_ is a free variable.

**height/1:**

**PREDICATE**

**Usage 1: height(Height)**

- *Description:* Specifies shape’s Height.
- *Call and exit should be compatible with:*
  
  _Height_ is an integer.  

- *The following properties should hold at call time:*
  
  _Height_ is currently a term which is not a free variable.

**Usage 2: height(Height)**

- *Description:* Gets shape’s Height.
- *Call and exit should be compatible with:*
  
  _Height_ is an integer.  

- *The following properties should hold at call time:*
  
  _Height_ is a free variable.

**center/2:**

**PREDICATE**

**Usage 1: center(X, Y)**

- *Description:* Specifies shape’s center with _X_ and _Y_.
- *Call and exit should be compatible with:*
  
  _X_ is an integer.  

  _Y_ is an integer.  

- *The following properties should hold at call time:*
  
  _X_ is currently a term which is not a free variable.  

  _Y_ is currently a term which is not a free variable.

**Usage 2: center(X, Y)**

- *Description:* Gets shape’s center with _X_ and _Y_.
- *Call and exit should be compatible with:*
  
  _X_ is an integer.  

  _Y_ is an integer.  

- *The following properties should hold at call time:*
  
  _X_ is a free variable.  

  _Y_ is a free variable.
**angle_start/1:**

Usage 1: angle_start(Angle)
- Description: Angle specifies the beginning of the angular range occupied by the arc. Degrees are given in units of degrees measured counter-clockwise from the 3-o’clock position; it may be either positive or negative.
- Call and exit should be compatible with:
  - Angle is an integer.  
- The following properties should hold at call time:
  - Angle is currently a term which is not a free variable.

Usage 2: angle_start(Angle)
- Description: Gets the value of the Angle.
- Call and exit should be compatible with:
  - Angle is an integer.  
- The following properties should hold at call time:
  - Angle is a free variable.

**style_type/1:**

Usage 1: style_type(Style)
- Description: Style specifies how to draw the arc. If type is pieslice (the default) then the arc’s region is defined by a section of the oval’s perimeter plus two line segments, one between the center of the oval and each end of the perimeter section. If type is chord then the arc’s region is defined by a section of the oval’s perimeter plus a single line segment connecting the two end points of the perimeter section. If type is arc then the arc’s region consists of a section of the perimeter alone. In this last case the -fill option is ignored.
- Call and exit should be compatible with:
  - Style is currently instantiated to an atom.  
- The following properties should hold at call time:
  - Style is currently a term which is not a free variable.

Usage 2: style_type(Style)
- Description: Gets the Style of the arc.
- Call and exit should be compatible with:
  - Style is currently instantiated to an atom.  
- The following properties should hold at call time:
  - Style is a free variable.

**outline_color/1:**

Usage 1: outline_color(Color)
- Description: Color specifies the color used for drawing the arc’s outline. This option defaults to black.
- Call and exit should be compatible with:
  - Color is currently instantiated to an atom.  
- The following properties should hold at call time:
  - Color is currently a term which is not a free variable.
Usage 2: outline_color(Color)

- Description: It gets arc's outline Color.
- Call and exit should be compatible with:
  Color is currently instantiated to an atom. \(\text{(term_typing:atom/1)}\)
- The following properties should hold at call time:
  Color is a free variable. \(\text{(term_typing:var/1)}\)
130 oval_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#139 (2003/12/4, 17:34:21 CET)

130.1 Usage and interface (oval_class)

- Library usage:
  :- use_module(library(oval_class)).
- Exports:
  - Predicates:
    coord/4, width/1, height/1, center/2, outline_color/1.
- Other modules used:
  - System library modules:
    objects/objects_rt.

130.2 Documentation on exports (oval_class)

coord/4:  
Usage: coord(X1, Y1, X2, Y2)  
- Description: X1, Y1, X2, and Y2 give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval.  
- Call and exit should be compatible with:
  X1 is an integer.  
  Y1 is an integer.  
  X2 is an integer.  
  Y2 is an integer.  
- The following properties should hold at call time:
  X1 is currently a term which is not a free variable.  
  Y1 is currently a term which is not a free variable.  
  X2 is currently a term which is not a free variable.  
  Y2 is currently a term which is not a free variable.

width/1:  
Usage 1: width(Width)  
- Description: Specifies shape’s Width.  
- Call and exit should be compatible with:
  Width is an integer.  
- The following properties should hold at call time:
  Width is currently a term which is not a free variable.
Usage 2: width(Width)
   - Description: Gets shape’s Width.
   - Call and exit should be compatible with:
     Width is an integer. (basic_props:int/1)
   - The following properties should hold at call time:
     Width is a free variable. (term_typing:var/1)

height/1:
   PREDICATE
   Usage 1: height(Height)
   - Description: Specifies shape’s Height.
   - Call and exit should be compatible with:
     Height is an integer. (basic_props:int/1)
   - The following properties should hold at call time:
     Height is currently a term which is not a free variable. (term_typing:nonvar/1)
   Usage 2: height(Height)
   - Description: Gets shape’s Height.
   - Call and exit should be compatible with:
     Height is an integer. (basic_props:int/1)
   - The following properties should hold at call time:
     Height is a free variable. (term_typing:var/1)

center/2:
   PREDICATE
   Usage 1: center(X, Y)
   - Description: Specifies shape’s center with X and Y.
   - Call and exit should be compatible with:
     X is an integer. (basic_props:int/1)
     Y is an integer. (basic_props:int/1)
   - The following properties should hold at call time:
     X is currently a term which is not a free variable. (term_typing:nonvar/1)
     Y is currently a term which is not a free variable. (term_typing:nonvar/1)
   Usage 2: center(X, Y)
   - Description: Gets shape’s center with X and Y.
   - Call and exit should be compatible with:
     X is an integer. (basic_props:int/1)
     Y is an integer. (basic_props:int/1)
   - The following properties should hold at call time:
     X is a free variable. (term_typing:var/1)
     Y is a free variable. (term_typing:var/1)
outline_color/1:

Usage 1: outline_color(Color)
- *Description:* Color specifies the color to be used for drawing the oval’s outline. This option defaults to black.
- *Call and exit should be compatible with:*
  *Color* is currently instantiated to an atom. (term_typing:atom/1)
- *The following properties should hold at call time:*
  *Color* is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: outline_color(Color)
- *Description:* Gets oval’s outline Color.
- *Call and exit should be compatible with:*
  *Color* is currently instantiated to an atom. (term_typing:atom/1)
- *The following properties should hold at call time:*
  *Color* is a free variable. (term_typing:var/1)
131 poly_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#187 (2003/12/10, 21:19:39 CET)

131.1 Usage and interface (poly_class)

- Library usage:
  ```prolog```
  ```
  :- use_module(library(poly_class)).
  ```
- Other modules used:
  - System library modules:
    ```prolog```
    ```
    class/class_rt, class/virtual, objects/objects_rt, lists.
    ```

131.2 Documentation on multifiles (poly_class)

$\textit{class}$/1: \hspace{1cm} \textbf{PREDICATE}

No further documentation available for this predicate.
The predicate is \textit{multifile}.

class$\textit{super}$/2: \hspace{1cm} \textbf{PREDICATE}

No further documentation available for this predicate.
The predicate is \textit{multifile}.

class$\textit{initial\_state}$/3: \hspace{1cm} \textbf{PREDICATE}

No further documentation available for this predicate.
The predicate is \textit{multifile}.

class$\textit{virtual}$/6: \hspace{1cm} \textbf{PREDICATE}

No further documentation available for this predicate.
The predicate is \textit{multifile}.

class$\textit{attr\_template}$/4: \hspace{1cm} \textbf{PREDICATE}

No further documentation available for this predicate.
The predicate is \textit{multifile}.

class$\textit{default\_cons}$/1: \hspace{1cm} \textbf{PREDICATE}

No further documentation available for this predicate.
The predicate is \textit{multifile}. 

class$constructor/4:
No further documentation available for this predicate.
The predicate is multife.

class$destructor/3:
No further documentation available for this predicate.
The predicate is multife.

class$implements/2:
No further documentation available for this predicate.
The predicate is multife.
132 line_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#131 (2003/12/4, 17:33:38 CET)

132.1 Usage and interface (line_class)

- Library usage:
  :- use_module(library(line_class)).
- Other modules used:
  - System library modules:
    class/class_rt, class/virtual, objects/objects_rt, lists.

132.2 Documentation on multifiles (line_class)

$\text{class}/1$: No further documentation available for this predicate.
The predicate is multi\text{file}.

class$\text{super}/2$: No further documentation available for this predicate.
The predicate is multi\text{file}.

class$\text{initial\_state}/3$: No further documentation available for this predicate.
The predicate is multi\text{file}.

class$\text{virtual}/6$: No further documentation available for this predicate.
The predicate is multi\text{file}.

class$\text{attr\_template}/4$: No further documentation available for this predicate.
The predicate is multi\text{file}.

class$\text{default\_cons}/1$: No further documentation available for this predicate.
The predicate is multi\text{file}.
class$constructor/4:
No further documentation available for this predicate.
The predicate is *multifile*.

PREDICATE

class$destructor/3:
No further documentation available for this predicate.
The predicate is *multifile*.

PREDICATE

class$implements/2:
No further documentation available for this predicate.
The predicate is *multifile*.

PREDICATE
133 text_class (library)

Author(s): Montserrat Urraca.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#148 (2003/12/4, 17:34:58 CET)

133.1 Usage and interface (text_class)

- Library usage:
  :- use_module(library(text_class)).
- Exports:
  - Predicates:
    coord/2, point/2, text_characters/1, anchor/1, font_type/1, justify_text/1.
- Other modules used:
  - System library modules:
    objects/objects_rt.

133.2 Documentation on exports (text_class)

coord/2: PREDICATE
Usage: coord(X, Y)
  - Description: X and Y specify the coordinates of a point used to position the text on the display.
  - Call and exit should be compatible with:
    X is an integer. (basic_props:int/1)
    Y is an integer. (basic_props:int/1)
  - The following properties should hold at call time:
    X is currently a term which is not a free variable. (term_typing:nonvar/1)
    Y is currently a term which is not a free variable. (term_typing:nonvar/1)

point/2: PREDICATE
Usage: point(X, Y)
  - Description: X and Y change the coordinates of a point used to position the text on the display.
  - Call and exit should be compatible with:
    X is an integer. (basic_props:int/1)
    Y is an integer. (basic_props:int/1)
  - The following properties should hold at call time:
    X is currently a term which is not a free variable. (term_typing:nonvar/1)
    Y is currently a term which is not a free variable. (term_typing:nonvar/1)
**text_characters/1:**

**Usage 1:** text_characters(Text)
- **Description:** Text specifies the characters to be displayed in the text item. This option defaults to an empty string.
- **Call and exit should be compatible with:**
  Text is currently instantiated to an atom.
- **The following properties should hold at call time:**
  Text is currently a term which is not a free variable.

**Usage 2:** text_characters(Text)
- **Description:** Gets the text displayed in the text item.
- **Call and exit should be compatible with:**
  Text is currently instantiated to an atom.
- **The following properties should hold at call time:**
  Text is a free variable.

**anchor/1:**

**Usage 1:** anchor(AnchorPos)
- **Description:** AnchorPos tells how to position the text relative to the positioning point for the text. This option defaults to center.
- **Call and exit should be compatible with:**
  AnchorPos is currently instantiated to an atom.
- **The following properties should hold at call time:**
  AnchorPos is currently a term which is not a free variable.

**Usage 2:** anchor(AnchorPos)
- **Description:** Gets the position of the text relative to the positioning point.
- **Call and exit should be compatible with:**
  AnchorPos is currently instantiated to an atom.
- **The following properties should hold at call time:**
  AnchorPos is a free variable.

**font_type/1:**

**Usage 1:** font_type(Font)
- **Description:** Font specifies the font to use for the text item. This option defaults to arial.
- **Call and exit should be compatible with:**
  Font is currently instantiated to an atom.
- **The following properties should hold at call time:**
  Font is currently a term which is not a free variable.

**Usage 2:** font_type(Font)
- **Description:** Gets the value of the Font used for the text item.
- **Call and exit should be compatible with:**
  Font is currently instantiated to an atom.
- **The following properties should hold at call time:**
  Font is a free variable.
justify_text/1:

Usage 1: justify_text(How)

- **Description**: How specifies how to justify the text within its bounding region. How must be one of the values left, right or center. This option defaults to left.

- **Call and exit should be compatible with**:
  - How is currently instantiated to an atom. (term_typing:atom/1)

- **The following properties should hold at call time**:
  - How is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: justify_text(How)

- **Description**: Gets How is justified the text.

- **Call and exit should be compatible with**:
  - How is currently instantiated to an atom. (term_typing:atom/1)

- **The following properties should hold at call time**:
  - How is a free variable. (term_typing:var/1)
134 The PiLLoW Web programming library

Author(s): Daniel Cabeza, Manuel Hermenegildo, clip@clip.dia.fi.upm.es, http://www.clip.dia.fi.upm.es/, The CLIP Group, School of Computer Science, Technical University of Madrid.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#107 (2003/12/22, 17:59:44 CET)

This package implements the PiLLoW library [CHV96a]. The following three chapters document, respectively, the predicates for HTML/XML/CGI programming, the predicate for HTTP connectivity, and the types used in the definition of the predicates (key for fully understanding the other predicates). You can find a paper and some additional information in the library/pillow/doc directory of the distribution, and in the WWW at http://clip.dia.fi.upm.es/Software/pillow/pillow.html. There is also a PiLLoW on-line tutorial (slides) at http://clip.dia.fi.upm.es/logalg/slides/C_pillow/C_pillow.html which illustrates the basic features and provides a number of examples of PiLLoW use.

134.1 Installing PiLLoW

To correctly install PiLLoW, first, make sure you downloaded the right version of PiLLoW (there are different versions for different LP/CLP systems; the version that comes with Ciao is of course the right one for Ciao). Then, please follow these steps:

1. Copy the files in the images directory to a WWW accessible directory in your server.
2. Edit the file icon_address.pl and change the fact to point to the URL to be used to access the images above.
3. In the Ciao system the files are in the correct place, in other systems copy the files pillow.pl and icon_address.pl to a suitable directory so that your Prolog system will find them.

134.2 Usage and interface (pillow)

- Library usage:
  := use_package(pillow).
  
or
  := module(...,...,[pillow]).
- New operators defined:
  $/2 [150,xfx], $/1 [150,fx].
- Other modules used:
  - System library modules:
    pillow/http, pillow/html.
Chapter 135: HTML/XML/CGI programming

Author(s): Daniel Cabeza, Manuel Hermenegildo, Sacha Varma.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

This module implements the predicates of the PiLloW package related to HTML/XML generation and parsing, CGI and form handlers programming, and in general all the predicates which do not imply the use of the HTTP protocol.

135.1 Usage and interface (html)

- Library usage:
  
  :- use_module(library(html)).

- Exports:
  - Predicates:
    
    output_html/1, html2terms/2, xml2terms/2, html_template/3, html_report_error/1, get_form_input/1, get_form_value/3, form_empty_value/1, form_default/3, set_cookie/2, get_cookies/1, url_query/2, url_query_values/2, my_url/1, url_info/2, url_info_relative/3, form_request_method/1, icon_address/2, html_protect/1, http_lines/3.

  - Multifiles:
    
    define_flag/3, html_expansion/2.

- Other modules used:

  - System library modules:
    
    strings, lists, system, pillow/pillow_aux, pillow/pillow_types.

135.2 Documentation on exports (html)

**output_html/1:**

PREDICATE

output_html(HTMLTerm)

Outputs HTMLTerm, interpreted as an html_term/1, to current output stream.

Usage:

- *The following properties should hold at call time:*
  
  HTMLTerm is a term representing HTML code. (pillow_types:html_term/1)

**html2terms/2:**

PREDICATE

html2terms(String, Terms)

String is a character list containing HTML code and Terms is its prolog structured representation.

Usage 1:

- *Description:* Translates an HTML-term into the HTML code it represents.
The following properties should hold at call time:
- **String** is a free variable. (*term_typing*:var/1)
- **Terms** is a term representing HTML code. (*pillow_types*:html_term/1)

The following properties hold upon exit:
- **String** is a string (a list of character codes). (*basic_props*:string/1)
- **Terms** is a term representing HTML code in canonical form. (*pillow_types*:canonic_html_term/1)

**Usage 2:**
- **Description:** Translates HTML code into a structured HTML-term.
- **Calls should, and exit will be compatible with:**
  - **Terms** is a term representing HTML code in canonical form. (*pillow_types*:canonic_html_term/1)

xml2terms/2:  
xml2terms(String, Terms)

String is a character list containing XML code and Terms is its prolog structured representation.

**Usage 1:**
- **Description:** Translates a XML-term into the XML code it represents.
- **The following properties should hold at call time:**
  - **String** is a free variable. (*term_typing*:var/1)
  - **Terms** is a term representing HTML code. (*pillow_types*:html_term/1)

html_template/3:  
html_template(Chars, Terms, Dict)

Interprets Chars as an HTML template returning in Terms the corresponding structured HTML-term, which includes variables, and unifying Dict with a dictionary of those variables (an incomplete list of name=Var pairs). An HTML template is standard HTML
code, but in which “slots” can be defined and given an identifier. These slots represent parts of the HTML code in which other HTML code can be inserted, and are represented in the HTML-term as free variables. There are two kinds of variables in templates:

- Variables representing page contents. A variable with name `name` is defined with the special tag `<V>name</V>`.
- Variables representing tag attributes. They occur as an attribute or an attribute value starting with `_`, followed by its name, which must be formed by alphabetic characters.

As an example, suppose the following HTML template:

```html
<html>
<body bgcolor=_bgcolor>
<v>content</v>
</body>
</html>
```

The following query in the Ciao toplevel shows how the template is parsed, and the dictionary returned:

```ciao
?- file_to_string('template.html',_S), html_template(_S,Terms,Dict).

Dict = [bgcolor=_A,content=_B|_],
Terms = [env(html,[],['"',env(body,[bgcolor=_A],[" ",B," "]),'"']),'"'] ?
```

Yes

If a dictionary with values is supplied at call time, then variables are unified accordingly inside the template:

```ciao
?- file_to_string('template.html',_S), html_template(_S,Terms,[content=b("hello world!"),bgcolor="white"]).

Terms = [env(html,[],['" ",env(body,[bgcolor="white"],[" ",b("hello world!")," "]),'"']),'"'] ?
```

Yes

Usage:

- **Calls should, and exit will be compatible with:**
  - `Terms` is a term representing HTML code in canonical form. (pillow_types:canonic_html_term/1)
  - `Dict` is a list. (basic_props:list/1)
- **The following properties should hold at call time:**
  - `Chars` is a string (a list of character codes). (basic_props:string/1)
html_report_error/1:

Usage: html_report_error(Error)
Description: Outputs error Error as a standard HTML page.

get_form_input/1:

get_form_input(Dict)

Translates input from the form (with either the POST or GET methods, and even with CONTENT_TYPE multipart/form-data) to a dictionary Dict of attribute=value pairs. If the flag raw_form_values is off (which is the default state), it translates empty values (which indicate only the presence of an attribute) to the atom "$empty", values with more than one line (from text areas or files) to a list of lines as strings, the rest to atoms or numbers (using name/2). If the flag on, it gives all values as atoms, without translations.

Usage:

The following properties should hold at call time:

Dict is a free variable.

The following properties hold upon exit:

Dict is a dictionary of values of the attributes of a form. It is a list of form_assignment.

get_form_value/3:

get_form_value(Dict, Var, Val)

Unifies Val with the value for attribute Var in dictionary Dict. Does not fail: value is '' if not found (this simplifies the programming of form handlers when they can be accessed directly).

Usage:

Calls should, and exit will be compatible with:

Val is a value of an attribute of a form.

The following properties should hold at call time:

Dict is a dictionary of values of the attributes of a form. It is a list of form_assignment.

Var is an atom.

The following properties hold upon exit:

Val is a value of an attribute of a form.

form_empty_value/1:

Usage: form_empty_value(Term)
Description: Checks that Term, a value coming from a text area is empty (can have spaces, newlines and linefeeds).
Chapter 135: HTML/XML/CGI programming

form_default/3:  
Usage: form_default(Val, Default, NewVal)  
Description: Useful when a form is only partially filled, or when the executable can be invoked either by a link or by a form, to set form defaults. If the value of Val is empty then NewVal=Default, else NewVal=Val.  
The following properties should hold at call time:  
Val is currently a term which is not a free variable.  
Default is currently a term which is not a free variable.  
NewVal is a free variable.

set_cookie/2:  
set_cookie(Name, Value)  
Sets a cookie of name Name and value Value. Must be invoked before outputting any data, including the cgi_reply html-term.  
Usage:  
The following properties should hold at call time:  
Name is an atom.  
Value is an atomic term (an atom or a number).

get_cookies/1:  
get_cookies(Cookies)  
Unifies Cookies with a dictionary of attribute=value pairs of the active cookies for this URL. If the flag raw_form_values is on, values are always atoms even if they could be interpreted as numbers.  
Usage:  
The following properties should hold at call time:  
Cookies is a dictionary of values. It is a list of pairs atom=constant.

url_query/2:  
url_query(Dict, URLArgs)  
(Deprecated, see url_query_values/2) Translates a dictionary Dict of parameter values into a string URLArgs for appending to a URL pointing to a form handler.  
Usage 1:  
The following properties should hold at call time:  
Dict is a dictionary of values. It is a list of pairs atom=constant.

Usage 2:
The following properties should hold at call time:

Dict is a dictionary of values. It is a list of pairs atom=constant. (pillow_types:value_dict/1)

URLArgs is a free variable. (term_typing:var/1)

The following properties hold upon exit:

URLArgs is a string (a list of character codes). (basic_props:string/1)

Usage 3:

The following properties should hold at call time:

Dict is a free variable. (term_typing:var/1)

URLArgs is a string (a list of character codes). (basic_props:string/1)

The following properties hold upon exit:

Dict is a dictionary of values. It is a list of pairs atom=constant. (pillow_types:value_dict/1)

url_query_values/2: PREDICATE

url_query_values(Dict, URLArgs)

Dict is a dictionary of parameter values and URLArgs is the URL-encoded string of those assignments, which may appear after an URL pointing to a CGI script preceded by a '?'.

Dict is computed according to the raw_form_values flag. The use of this predicate is reversible.

my_url/1: PREDICATE

my_url(URL)

Unifies URL with the Uniform Resource Locator (WWW address) of this cgi executable.

Usage:

Calls should, and exit will be compatible with:

URL is a string (a list of character codes). (basic_props:string/1)

The following properties hold upon exit:

URL is a string (a list of character codes). (basic_props:string/1)

url_info/2: PREDICATE

url_info(URL, URLTerm)

Translates a URL URL to a Prolog structure URLTerm which details its various components, and vice-versa. For now non-HTTP URLs make the predicate fail.

Usage 1:

Calls should, and exit will be compatible with:

URLTerm specifies a URL. (pillow_types:url_term/1)

The following properties should hold at call time:

URL is an atom. (basic_props:atm/1)

The following properties hold upon exit:

URLTerm specifies a URL. (pillow_types:url_term/1)

Usage 2:
Chapter 135: HTML/XML/CGI programming

-- Calls should, and exit will be compatible with:
URLTerm specifies a URL. (pillow_types:url_term/1)
-- The following properties should hold at call time:
URL is a string (a list of character codes). (basic_props:string/1)
-- The following properties hold upon exit:
URLTerm specifies a URL. (pillow_types:url_term/1)

Usage 3:
-- The following properties should hold at call time:
URL is a free variable. (term_typing:var/1)
URLTerm specifies a URL. (pillow_types:url_term/1)
-- The following properties hold upon exit:
URL is a string (a list of character codes). (basic_props:string/1)

url_info_relative/3:
PREDICATE
url_info_relative(URL, BaseURLTerm, URLTerm)
Translates a relative URL URL which appears in the HTML page referred to by BaseURLTerm into URLTerm, a Prolog structure containing its absolute parameters. Absolute URLs are translated as with url_info/2. E.g.
url_info_relative("dadu.html", http('www.foo.com',80,"/bar/scoob.html"), Info)
gives Info = http('www.foo.com',80,"/bar/dadu.html").
Usage 1:
-- Calls should, and exit will be compatible with:
URLTerm specifies a URL. (pillow_types:url_term/1)
-- The following properties should hold at call time:
URL is an atom. (basic_props:atm/1)
BaseURLTerm specifies a URL. (pillow_types:url_term/1)
-- The following properties hold upon exit:
URLTerm specifies a URL. (pillow_types:url_term/1)

Usage 2:
-- Calls should, and exit will be compatible with:
URLTerm specifies a URL. (pillow_types:url_term/1)
-- The following properties should hold at call time:
URL is a string (a list of character codes). (basic_props:string/1)
BaseURLTerm specifies a URL. (pillow_types:url_term/1)
-- The following properties hold upon exit:
URLTerm specifies a URL. (pillow_types:url_term/1)

form_request_method/1:
PREDICATE
Usage: form_request_method(Method)
-- Description: Unifies Method with the method of invocation of the form handler (GET or POST).
-- The following properties hold upon exit:
Method is an atom. (basic_props:atm/1)
icon_address/2:  
icon_address(Img, IAddress)

The PiLLoW image Img has URL IAddress.

Usage:
- *Calls should, and exit will be compatible with:*
  - Img is an atom. (basic_props:atm/1)
  - IAddress is an atom. (basic_props:atm/1)
- *The following properties hold upon exit:*
  - Img is an atom. (basic_props:atm/1)
  - IAddress is an atom. (basic_props:atm/1)

html_protect/1:  
html_protect(Goal)

Calls Goal. If an error occurs during its execution, or it fails, an HTML page is output informing about the incident. Normally the whole execution of a CGI is protected thus.

*Meta-predicate* with arguments: html_protect(goal).

http_lines/3:  
Usage: http_lines(Lines, String, Tail)
- *Description:* Lines is a list of the lines with occur in String until Tail. The lines may end UNIX-style or DOS-style in String, in Lines they have not end of line characters. Suitable to be used in DCGs.
- *Calls should, and exit will be compatible with:*
  - Lines is a list of strings. (basic_props:list/2)
  - String is a string (a list of character codes). (basic_props:string/1)
  - Tail is a string (a list of character codes). (basic_props:string/1)

135.3 Documentation on multfiles (html)

define_flag/3:  
Defines a flag as follows:

```
define_flag(raw_form_values,[on,off],off).
```

(See Chapter 24 [Changing system behaviour and various flags], page 131).

If flag is on, values returned by get_form_input/1 are always atoms, unchanged from its original value.

The predicate is *multifile.*

html_expansion/2:  
The predicate is *multifile.*

Usage: html_expansion(Term, Expansion)
- *Description:* Hook predicate to define macros. Expand occurrences of Term into Expansion, in output_html/1. Take care to not transform something into itself!
135.4 Other information (html)

The code uses input from L. Naish’s forms and F. Bueno’s previous Chat interface. Other people who have contributed is (please inform us if we leave out anybody): Markus Fromherz, Samir Genaim.
136 HTTP connectivity

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.3#114 (1999/11/24, 0:57:16 MET)

This module implements the HTTP protocol, which allows retrieving data from HTTP servers.

136.1 Usage and interface (http)

- Library usage:
  :- use_module(library(http)).
- Exports:
  - Predicates:
    fetch_url/3.
- Other modules used:
  - System library modules:
    strings, lists, pillow/pillow_aux, pillow/pillow_types, pillow/http_ll.

136.2 Documentation on exports (http)

fetch_url/3: 
fetch_url(URL, Request, Response)

Fetches the document pointed to by URL from Internet, using request parameters Request, and unifies Response with the parameters of the response. Fails on timeout. Note that redirections are not handled automatically, that is, if Response contains terms of the form status(redirection,301,_), and location(NewURL), the program should in most cases access location NewURL.

Usage: fetch_url(URL, Request, Response)
  - The following properties should hold at call time:
    URL specifies a URL. (pillow_types:url_term/1)
    Request is a list of http_request_params. (basic_props:list/2)
  - The following properties hold upon exit:
    Response is a list of http_response_params. (basic_props:list/2)
137 PiLLoW types

Author(s): Daniel Cabeza.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#131 (2003/12/30, 22:44:12 CET)

Here are defined the regular types used in the documentation of the predicates of the PiLLoW package.

137.1 Usage and interface (pillow_types)

- Library usage:
  :- use_module(library(pillow_types)).
- Exports:
  - Regular Types:
    canonic_html_term/1, canonic_xml_term/1, html_term/1, form_dict/1, form_assignment/1, form_value/1, value_dict/1, url_term/1, http_request_param/1, http_response_param/1, http_date/1, weekday/1, month/1, hms_time/1.

137.2 Documentation on exports (pillow_types)

canonic_html_term/1: REGTYPE
A term representing HTML code in canonical, structured way. It is a list of terms defined by the following predicate:

\[
\text{canonic_html_item}(\text{comment}(S)) :- \\
\quad \text{string}(S).
\]

\[
\text{canonic_html_item}(\text{declare}(S)) :- \\
\quad \text{string}(S).
\]

\[
\text{canonic_html_item}(\text{env}(\text{Tag}, \text{Atts}, \text{Terms})) :- \\
\quad \text{atm}(\text{Tag}), \\
\quad \text{list}(\text{Atts}, \text{tag_attrib}), \\
\quad \text{canonic_html_term}(\text{Terms}).
\]

\[
\text{canonic_html_item}($(\text{Tag}, \text{Atts})) :- \\
\quad \text{atm}(\text{Tag}), \\
\quad \text{list}(\text{Atts}, \text{tag_attrib}).
\]

\[
\text{canonic_html_item}(\text{S}) :- \\
\quad \text{string}(\text{S}).
\]

\[
\text{tag_attrib}(\text{Att}) :- \\
\quad \text{atm}(\text{Att}).
\]

\[
\text{tag_attrib}(\text{Att}=\text{Val}) :- \\
\quad \text{atm}(\text{Att}), \\
\quad \text{string}(\text{Val}).
\]

Each structure represents one HTML construction:

\[
\text{env}(\text{tag}, \text{attribs}, \text{terms})
\]
An HTML environment, with name \text{tag}, list of attributes \text{attribs} and contents \text{terms}.
$(tag,attribs)$
An HTML element of name $tag$ and list of attributes $attribs$. $(\$)/2$ is defined by the pillow package as an infix, binary operator.

comment(string)
An HTML comment (translates to/from $<!--string-->$).

decclare(string)
An HTML declaration, they are used only in the header (translates to/from $<!string>$).

string Normal text is represented as a list of character codes.

For example, the term
$$\text{env}(a,[\text{href}="\text{www.therainforestsite.com}"],["Visit ",\text{img}[\text{src}="\text{TRFS.gif}"]])$$
is output to (or parsed from):
$$<a \text{ href}="\text{www.therainforestsite.com}">\text{Visit } <\text{img} \text{ src}="\text{TRFS.gif}"</a>$$

Usage: canonic_html_term(HTMLTerm)
$\begin{array}{l}
\quad \text{Description: HTMLTerm is a term representing HTML code in canonical form.}
\end{array}$

canonic_xml_term/1:
A term representing XML code in canonical, structured way. It is a list of terms defined by the following predicate (see tag_attrib/1 definition in canonic_html_term/1):
$$\text{canonic_xml_item}(\text{Term}) :-$$
$$\text{canonic_html_item}(\text{Term}).$$
$$\text{canonic_xml_item}(\text{xmldecl}(\text{Atts})) :-$$
$$\text{list}((\text{Atts}, \text{tag_attrib})).$$
$$\text{canonic_xml_item}(\text{env}(\text{Tag}, \text{Atts}, \text{Terms})) :-$$
$$\text{atm}(\text{Tag}),$$
$$\text{list}((\text{Atts}, \text{tag_attrib})),$$
$$\text{canonic_xml_term}(\text{Terms}).$$
$$\text{canonic_xml_item}(\text{elem}(\text{Tag}, \text{Atts})) :-$$
$$\text{atm}(\text{Tag}),$$
$$\text{list}((\text{Atts}, \text{tag_attrib})).$$

In addition to the structures defined by canonic_html_term/1 (the $(\$)/2$ structure appears only in malformed XML code), the following structures can be used:

elem(tag,atts)
Specifies an XML empty element of name $tag$ and list of attributes $atts$. For example, the term
$$\text{elem}(\text{arc},[\text{weigh}="3", \text{begin}="n1", \text{end}="n2"])$$
is output to (or parsed from):
$$<\text{arc} \text{ weigh}="3" \text{ begin}="n1" \text{ end}="n2"/>$$

xmldecl(atts)
Specifies an XML declaration with attributes $atts$ (translates to/from $<?xml atts?>$)
Chapter 137: PiLLoW types

Usage: canonic_xml_term(XMLTerm)
- Description: XMLTerm is a term representing XML code in canonical form.

**html_term/1:**

A term which represents HTML or XML code in a structured way. In addition to the structures defined by canonic_html_term/1 or canonic_xml_term/1, the following structures can be used:

**begin(tag,atts)**

It translates to the start of an HTML environment of name `tag` and attributes `atts`. There exists also a `begin(tag)` structure. Useful, in conjunction with the next structure, when including in a document output generated by an existing piece of code (e.g. `tag = pre`). Its use is otherwise discouraged.

**end(tag)**

Translates to the end of an HTML environment of name `tag`.

**start**

Used at the beginning of a document (translates to `<html>`).

**end**

Used at the end of a document (translates to `</html>`).

**\**

Produces a horizontal rule (translates to `<hr>`).

**\**

Produces a line break (translates to `<br>`).

**\**

Produces a paragraph break (translates to `<p>`).

**image(address)**

Used to include an image of address (URL) `address` (equivalent to `img[src=address]`).

**image(address,atts)**

As above with the list of attributes `atts`.

**ref(address,text)**

Produces a hypertext link, `address` is the URL of the referenced resource, `text` is the text of the reference (equivalent to `a(href=address),text`).

**label(name,text)**

Labels `text` as a target destination with label `name` (equivalent to `a([name=name],text)`).

**heading(n,text)**

-produces a heading of level `n` (between 1 and 6), `text` is the text to be used as heading. Useful when one wants a heading level relative to another heading (equivalent to `hn(text)`).

**itemize(items)**

Produces a list of bulleted items, `items` is a list of corresponding HTML terms (translates to a `<ul>` environment).

**enumerate(items)**

Produces a list of numbered items, `items` is a list of corresponding HTML terms (translates to a `<ol>` environment).

**description(defs)**

Produces a list of defined items, `defs` is a list whose elements are definitions, each of them being a Prolog sequence (composed by `','/2 operators). The last element of the sequence is the definition, the other (if any) are the defined terms (translates to a `<dl>` environment).
nice_itemize(img,items)
    Produces a list of bulleted items, using the image img as bullet. The predicate icon_address/2 provides a colored bullet.

preformatted(text)
    Used to include preformatted text, text is a list of HTML terms, each element of the list being a line of the resulting document (translates to a <pre> environment).

verbatim(text)
    Used to include text verbatim, special HTML characters (<,>,&,", and space) are translated into its quoted HTML equivalent.

prolog_term(term)
    Includes any prolog term term, represented in functional notation. Variables are output as _.

nl
    Used to include a newline in the HTML source (just to improve human readability).

entity(name)
    Includes the entity of name name (ISO-8859-1 special character).

start_form(addr,atts)
    Specifies the beginning of a form. addr is the address (URL) of the program that will handle the form, and atts other attributes of the form, as the method used to invoke it. If atts is not present (there is only one argument) the method defaults to POST.

start_form
    Specifies the beginning of a form without assigning address to the handler, so that the form handler will be the cgi-bin executable producing the form.

donstart_form
    Specifies the end of a form.

checkbox(name,state)
    Specifies an input of type checkbox with name name, state is on if the checkbox is initially checked.

radio(name,value,selected)
    Specifies an input of type radio with name name (several radio buttons which are interlocked must share their name), value is the the value returned by the button, if selected=value the button is initially checked.

input(type,atts)
    Specifies an input of type type with a list of attributes atts. Possible values of type are text, hidden, submit, reset, Idots

textInput(name,atts,text)
    Specifies an input text area of name name. text provides the default text to be shown in the area, atts a list of attributes.

option(name,val,options)
    Specifies a simple option selector of name name, options is the list of available options and val is the initial selected option (if val is not in options the first item is selected by default) (translates to a <select> environment).

menu(name,atts,items)
    Specifies a menu of name name, list of attributes atts and list of options items. The elements of the list items are marked with the prefix operator $ to indicate that they are selected (translates to a <select> environment).

form_reply
cgi_reply  This two are equivalent, they do not generate HTML, rather, the CGI protocol requires this content descriptor to be used at the beginning by CGI executables (including form handlers) when replying (translates to Content-type: text/html).

pr  Includes in the page a graphical logo with the message “Developed using the PiLLoW Web programming library”, which points to the manual and library source.

name(text)  A term with functor name/1, different from the special functors defined herein, represents an HTML environment of name name and included text text. For example, the term

address('clip@clip.dia.fi.upm.es')

is translated into the HTML source

<address>clip@clip.dia.fi.upm.es</address>

name(atts,text)  A term with functor name/2, different from the special functors defined herein, represents an HTML environment of name name, attributes atts and included text text. For example, the term

a([href='http://www.clip.dia.fi.upm.es/'],"Clip home")

represents the HTML source

<a href="http://www.clip.dia.fi.upm.es/">Clip home</a>

Usage: html_term(HTMLTerm)

– Description: HTMLTerm is a term representing HTML code.

form_dict/1:  REGTYPE

Usage: form_dict(Dict)

– Description: Dict is a dictionary of values of the attributes of a form. It is a list of form_assignment

form_assignment/1:  REGTYPE

Usage: form_assignment(Eq)

– Description: Eq is an assignment of value of an attribute of a form. It is defined by:

form_assignment(A=V) :-
    atm(A),
    form_value(V).
form_value(A) :-
    atm(A).
form_value(N) :-
    num(N).
form_value(L) :-
    list(L,string).
form_value/1:  
Usage: form_value(V) 
   – Description: V is a value of an attribute of a form.

value_dict/1:  
Usage: value_dict(Dict) 
   – Description: Dict is a dictionary of values. It is a list of pairs atom=constant.

url_term/1:  
A term specifying an Internet Uniform Resource Locator. Currently only HTTP URLs are supported. Example: http('www.clip.dia.fi.upm.es',80,'/Software/Ciao/'). Defined as

   url_term(http(Host,Port,Document)) :-
   atm(Host),
   int(Port),
   string(Document).

Usage: url_term(URL) 
   – Description: URL specifies a URL.

http_request_param/1:  
A parameter of an HTTP request:
   • head: Specify that the document content is not wanted.
   • timeout(T): T specifies the time in seconds to wait for the response. Default is 300 seconds.
   • if_modified_since(Date): Get document only if newer than Date. Date has the format defined by http_date/1.
   • user_agent(Agent): Provides a user-agent field, Agent is an atom. The string "PiLLoW/1.1" (or whatever version of PiLLoW is used) is appended.
   • authorization(Scheme,Params): To provide credentials. See RFC 1945 for details.
   • option(Value): Any unary term, being Value an atom, can be used to provide another valid option (e.g. from('user@machine')).

Usage: http_request_param(Request) 
   – Description: Request is a parameter of an HTTP request.

http_response_param/1:  
A parameter of an HTTP response:
   • content(String): String is the document content (list of bytes). If the head parameter of the HTTP request is used, an empty list is get here.
   • status(Type,Code,Reason): Type is an atom denoting the response type, Code is the status code (an integer), and Reason is a string holding the reason phrase.
   • message_date(Date): Date is the date of the response, with format defined by http_date/1.
- **location**(*Loc*): This parameter appears when the document has moved, *Loc* is an atom holding the new location.
- **http_server**(*Server*): *Server* is the server responding, as a string.
- **authenticate**(*Params*): Returned if document is protected, *Params* is a list of challenges. See RFC 1945 for details.
- **allow**(*Methods*): *Methods* are the methods allowed by the server, as a list of atoms.
- **content_encoding**(*Encoding*): *Encoding* is an atom defining the encoding.
- **content_length**(*Length*): *Length* is the length of the document (an integer).
- **content_type**(*Type*, *Subtype*, *Params*): Specifies the document content type, *Type* and *Subtype* are atoms, *Params* a list of parameters (e.g. `content_type(text, html, [])`).
- **expires**(*Date*): *Date* is the date after which the entity should be considered stale. Format defined by http_date/1.
- **last_modified**(*Date*): *Date* is the date at which the sender believes the resource was last modified. Format defined by http_date/1.
- **pragma**(*String*): Miscellaneous data.
- **header**(*String*): Any other functor header/1 is an extension header.

**Usage**: http_response_param(Response)

- **Description**: Response is a parameter of an HTTP response.

**http_date/1**: REGTYPE

http_date(Date)

Date is a term defined as

```
http_date(date(WeekDay, Day, Month, Year, Time)) :-
    weekday(WeekDay),
    int(Day),
    month(Month),
    int(Year),
    hms_time(Time).
```

**Usage**: http_date(Date)

- **Description**: Date is a term denoting a date.

**weekday/1**: REGTYPE

Usage: weekday(WeekDay)

- **Description**: WeekDay is a term denoting a weekday.

**month/1**: REGTYPE

Usage: month(Month)

- **Description**: Month is a term denoting a month.

**hms_time/1**: REGTYPE

Usage: hms_time(Time)

- **Description**: Time is an atom of the form hh:mm:ss
138 Persistent predicate database

Author(s): J.M. Gomez, D. Cabeza, and M. Hermenegildo, clip@dia.fi.upm.es, http://www.clip.dia.fi.upm.es/, The CLIP Group, Facultad de Informática, Universidad Politécnica de Madrid.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

138.1 Introduction to persistent predicates

This library implements a generic persistent predicate database. The basic notion implemented by the library is that of a persistent predicate. The persistent predicate concept provides a simple, yet powerful generic persistent data access method [CHGT98, Par97]. A persistent predicate is a special kind of dynamic, data predicate that “resides” in some persistent medium (such as a set of files, a database, etc.) that is typically external to the program using such predicates. The main effect is that any changes made to a persistent predicate from a program “survive” across executions. I.e., if the program is halted and restarted the predicate that the new process sees is in precisely the same state as it was when the old process was halted (provided no change was made in the meantime to the storage by other processes or the user).

Persistent predicates appear to a program as ordinary predicates, and calls to these predicates can appear in clause bodies in the usual way. However, the definitions of these predicates do not appear in the program. Instead, the library maintains automatically the definitions of predicates which have been declared as persistent in the persistent storage.

Updates to persistent predicates can be made using enhanced versions of asserta_fact/1, assertz_fact/1 and retract_fact/1. The library makes sure that each update is a transactional update, in the sense that if the update terminates, then the permanent storage has definitely been modified. For example, if the program making the updates is halted just after the update and then restarted, then the updated state of the predicate will be seen. This provides security against possible data loss due to, for example, a system crash. Also, due to the atomicity of the transactions, persistent predicates allow concurrent updates from several programs.

138.2 Persistent predicates, files, and relational databases

The concept of persistent predicates provided by this library essentially implements a lightweight, simple, and at the same time powerful form of relational database (a deductive database), and which is standalone, in the sense that it does not require external support, other than the file management capabilities provided by the operating system. This is due to the fact that the persistent predicates are in fact stored in one or more auxiliary files below a given directory.

This type of database is specially useful when building small to medium-sized standalone applications in Prolog which require persistent storage. In many cases it provides a much easier way of implementing such storage than using files under direct program control. For example, interactive applications can use persistent predicates to represent their internal state in a way that is close to the application. The persistence of such predicates then allows automatically restoring the state to that at the end of a previous session. Using persistent predicates amounts to simply declaring some predicates as such and eliminates having to worry about opening files, closing them, recovering from system crashes, etc.

In other cases, however, it may be convenient to use a relational database as persistent storage. This may be the case, for example, when the data already resides in such a database (where it is perhaps accessed also by other applications) or the volume of data is very large. persdb_sql [CCG98] is a companion library which implements the same notion of persistent
predicates used herein, but keeping the storage in a relational database. This provides a very
natural and transparent way to access SQL database relations from a Prolog program. In that
library, facilities are also provided for reflecting more complex views of the database relations
as predicates. Such views can be constructed as conjunctions, disjunctions, projections, etc. of
database relations, and may include SQL-like aggregation operations.

A nice characteristic of the notion of persistent predicates used in both of these libraries is
that it abstracts away how the predicate is actually stored. Thus, a program can use persistent
predicates stored in files or in external relational databases interchangeably, and the type of
storage used for a given predicate can be changed without having to modify the program (except
for replacing the corresponding persistent/2 declarations).

An example application of the persdb and persdb_sql libraries (and also the pillow library
[CH97]), is WebDB [GCH98]. WebDB is a generic, highly customizable deductive database engine
with an html interface. WebDB allows creating and maintaining Prolog-based databases as well
as relational databases (residing in conventional relational database engines) using any standard
WWW browser.

138.3 Using file-based persistent predicates

Persistent predicates can be declared statically, using persistent/2 declarations (which is
the preferred method, when possible), or dynamically via calls to make_persistent/2. Currently, persistent predicates may only contain facts, i.e., they are dynamic predicates of type
data/1.

Predicates declared as persistent are linked to directory, and the persistent state of the
predicate will be kept in several files below that directory. The files in which the persistent
predicates are stored are in readable, plain ASCII format, and in Prolog syntax. One advantage
of this approach is that such files can also be created or edited by hand, in a text editor, or even
by other applications.

An example definition of a persistent predicate implemented by files follows:

```prolog
:- persistent(p/3, dbdir).

persistent_dir(dbdir, '/home/clip/public_html/db').
```

The first line declares the predicate p/3 persistent. The argument dbdir is a key used to
index into a fact of the relation persistent_dir/2, which specifies the directory where the
corresponding files will be kept. The effect of the declaration, together with the persistent_dir/2 fact, is that, although the predicate is handled in the same way as a normal data predicate,
in addition the system will create and maintain efficiently a persistent version of p/3 via files in
the directory /home/clip/public_html/db.

The level of indirection provided by the dbdir argument makes it easy to place the storage of
several persistent predicates in a common directory, by specifying the same key for all of them.
It also allows changing the directory for several such persistent predicates by modifying only
one fact in the program. Furthermore, the persistent_dir/2 predicate can even be dynamic
and specified at run-time.

138.4 Implementation Issues

We outline the current implementation approach. This implementation attempts to provide
at the same time efficiency and security. To this end, up to three files are used for each predicate
(the persistence set): the data file, the operations file, and the backup file. In the updated state
the facts (tuples) that define the predicate are stored in the data file and the operations file is
empty (the backup file, which contains a security copy of the data file, may or may not exist).
While a program using a persistent predicate is running, any insertion (assert) or deletion (retract) operations on the predicate are performed on both the program memory and on the persistence set. However, in order to incur only a small overhead in the execution, rather than changing the data file directly, a record of each of the insertion and deletion operations is appended to the operations file. The predicate is then in a transient state, in that the contents of the data file do not reflect exactly the current state of the corresponding predicate. However, the complete persistence set does.

When a program starts, all pending operations in the operations file are performed on the data file. A backup of the data file is created first to prevent data loss if the system crashes during this operation. The order in which this updating of files is done ensures that, if at any point the process dies, on restart the data will be completely recovered. This process of updating the persistence set can also be triggered at any point in the execution of the program (for example, when halting) by calling \texttt{update\_files}.

138.5 Defining an initial database

It is possible to define an initial database by simply including in the program code facts of persistent predicates. They will be included in the persistent database when it is created. They are ignored in successive executions.

138.6 Using persistent predicates from the top level

Special care must be taken when loading into the top level modules or user files which use persistent predicates. Beforehand, a goal \texttt{use\_module(library('persdb/persdbrt'))} must be issued. Furthermore, since persistent predicates defined by the loaded files are in this way defined dynamically, a call to \texttt{initialize\_db/0} is commonly needed after loading and before calling predicates of these files.
138.7 Usage and interface (persdbrt)

- **Library usage:**
  There are two packages which implement persistence: `persdb` and `persdb/ll` (for low level). In the first, the standard builtins `asserta_fact/1`, `assertz_fact/1`, and `retract_fact/1` are replaced by new versions which handle persistent data predicates, behaving as usual for normal data predicates. In the second package, predicates with names starting with `p` are defined, so that there is no overhead in calling the standard builtins. In any case, each package is used as usual: including it in the package list of the module, or using the `use_package/1` declaration.

- **Exports:**
  - **Predicates:**
    - `passerta_fact/1`, `passertz_fact/1`, `pretract_fact/1`, `pretractall_fact/1`, `asserta_fact/1`, `assertz_fact/1`, `retract_fact/1`, `initialize_db/0`, `make_persistent/2`, `update_files/0`, `update_files/1`.
  - **Multi-files:**
    - `persistent_dir/2`, `$is_persistent/2`.

- **Other modules used:**
  - **System library modules:**
    - `lists`, `streams`, `read`, `aggregates`, `system`, `file_locks/file_locks`, `persdb/persdbcache`.

138.8 Documentation on exports (persdbrt)

**passerta_fact/1:**

*Meta-predicate* with arguments: `passerta_fact(fact)`.

**Usage:** `passerta_fact(Fact)`

- **Description:** Persistent version of `asserta_fact/1`: the current instance of `Fact` is interpreted as a fact (i.e., a relation tuple) and is added at the beginning of the definition of the corresponding predicate. The predicate concerned must be declared `persistent`. Any uninstantiated variables in the `Fact` will be replaced by new, private variables. Defined in the `persdb/ll` package.

- **The following properties should hold at call time:**

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

**passertz_fact/1:**

*Meta-predicate* with arguments: `passertz_fact(fact)`.

**Usage:** `passertz_fact(Fact)`

- **Description:** Persistent version of `assertz_fact/1`: the current instance of `Fact` is interpreted as a fact (i.e., a relation tuple) and is added at the end of the definition of the corresponding predicate. The predicate concerned must be declared `persistent`. Any uninstantiated variables in the `Fact` will be replaced by new, private variables. Defined in the `persdb/ll` package.
The following properties should hold at call time:
Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

pretract_fact/1:
pretract_fact(P)
Retracts a predicate in both, the dynamic and the persistent databases.
Meta-predicate with arguments: pretract_fact(fact).
Usage: pretract_fact(Fact)
  Description: Persistent version of retract_fact/1: deletes on backtracking all the facts which unify with Fact. The predicate concerned must be declared persistent. Defined in the 'persdb/1' package.
  The following properties should hold at call time:
  Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

pretractall_fact/1:
pretractall_fact(P)
Retracts all the instances of a predicate in both, the dynamic and the persistent databases.
Meta-predicate with arguments: pretractall_fact(fact).

asserta_fact/1:
Meta-predicate with arguments: asserta_fact(fact).
Usage: asserta_fact(Fact)
  Description: Same as passerta_fact/1, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the persdb package.
  The following properties should hold at call time:
  Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

assertz_fact/1:
Meta-predicate with arguments: assertz_fact(fact).
Usage: assertz_fact(Fact)
  Description: Same as passertz_fact/1, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the persdb package.
  The following properties should hold at call time:
  Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
retract_fact/1:

Meta-predicate with arguments: retract_fact(fact).

Usage 1: retract_fact(Fact)

- Description: Same as retractall/1, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the persdb package.
- The following properties should hold at call time:
  Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 2: retract_fact(Fact)

- Description: Same as retractall/1, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the persdb package.
- The following properties should hold at call time:
  Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retractall_fact/1:

No further documentation available for this predicate.

Meta-predicate with arguments: retractall_fact(fact).

initialize_db/0:

Usage:

- Description: Initializes the whole database, updating the state of the declared persistent predicates. Must be called explicitly after dynamically defining clauses for persistent_dir/2.

make.persistent/2:

Meta-predicate with arguments: make.persistent(spec,?).

Usage: make.persistent(PredDesc, Keyword)

- Description: Dynamic version of the persistent declaration.
- The following properties should hold at call time:
  PredDesc is a Name/Arity structure denoting a predicate name:
  
  predname(P/A) :-
  atm(P),
  nnegint(A).

  (basic_props:predname/1)

  Keyword is an atom corresponding to a directory identifier. (persdbcache:keyword/1)

update_files/0:

Usage:

- Description: Updates the files comprising the persistence set of all persistent predicates defined in the application.
update_files/1:  
*Meta-predicate* with arguments: update_files(list(spec)).

**Usage:** update_files(PredSpecList)
- **Description:** Updates the files comprising the persistence set of the persistent predicates in PredSpecList.
- **Call and exit should be compatible with:**
  PredSpecList is a list of prednames. (basic_props:list/2)

### 138.9 Documentation on multifiles (persdbrt)

**persistent_dir/2:**  
The predicate is *multifile.*
The predicate is of type *data.*

**Usage:** persistent_dir(Keyword, Location_Path)
- **Description:** Relates identifiers of locations (the *Keywords*) with descriptions of such locations (*Location_Paths*). *Location_Path* is a directory and it means that the definition for the persistent predicates associated with *Keyword* is kept in files below that directory (which must previously exist). These files, in the updated state, contain the actual definition of the predicate in Prolog syntax (but with module names resolved).
- **The following properties should hold at call time:**
  - *Keyword* is an atom corresponding to a directory identifier. (persdbcache:keyword/1)
  - *Location_Path* is an atom, the name of a directory. (persdb:directoryname/1)

**$is_persistent/2:**  
$is_persistent(Spec, Key)
Predicate *Spec* persists within database *Key*. Programmers should not define this predicate directly in the program.
The predicate is *multifile.*
The predicate is of type *data.*

### 138.10 Documentation on internals (persdbrt)

**persistent/2:**  
**Declaration**

**Usage:** :- persistent(PredDesc, Keyword).
- **Description:** Declares the predicate *PredDesc* as persistent. *Keyword* is the identifier of a location where the persistent storage for the predicate is kept. The location *Keyword* is described in the *persistent_dir* predicate, which must contain a fact in which the first argument unifies with *Keyword*.
- **The following properties should hold upon exit:**
  - *PredDesc* is a Name/Arity structure denoting a predicate name:
predname(P/A) :-
    atm(P),
    nnegint(A).

Keyword is an atom corresponding to a directory identifier.

**keyword/1:**

**PREDICATE**

An atom which identifies a fact of the **persistent_dir/2** relation. This fact relates this atom to a directory in which the persistent storage for one or more persistent predicates is kept. Storage is expected under a subdirectory by the name of the module and in a file by the name of the predicate.

**directoryname/1:**

**REGTYPE**

- **Usage:** directoryname(X)
- **Description:** X is an atom, the name of a directory.

138.11 Known bugs and planned improvements (**persdbrt**)

- To load in the toplevel a file which uses this package, module `library('persdb/persdbrt')` has to be previously loaded.
Chapter 139: Using the persdb library

**Author(s):** The CLIP Group.

**Version:** 1.11#222 (2004/5/24, 13:8:7 CEST)

**Version of last change:** 1.11#108 (2003/12/22, 18:3:39 CET)

Through the following examples we will try to illustrate the two mains ways of declaring and using persistent predicates: statically (the preferred method) and dynamically (necessary when the new persistent predicates have to be defined at run-time). The final example is a small application implementing a simple persistent queue.

### 139.1 An example of persistent predicates (static version)

```prolog
:- use_package(iso).
:- use_package(persdb).

%% Declare the directory associated to the key "db" where the
%% persistence sets of the persistent predicates are stored:
persistent_dir(db,'./').

%% Declare a persistent predicate:
:- persistent(bar/1, db).

%% Read a term, storing it in a new fact of the persistent predicate
%% and list all the current facts of that predicate
main:-
    read(X),
    assertz_fact(bar(X)),
    findall(Y,bar(Y),L),
    write(L).

erase_one :-
    retract_fact(bar(_)).
erase_all :-
    retractall_fact(bar(_)).
```

### 139.2 An example of persistent predicates (dynamic version)

```prolog
:- use_package(iso).
:- use_package(persdb).

main([X]):-
    %% Declare the directory associated to the key "db"
    asserta_fact(persistent_dir(db,'./')),
    %% Declare the predicate bar/1 as dynamic (and data) at run-time
    data(bar/1),
    %% Declare the predicate bar/1 as persistent at run-time
    make_persistent(bar/1, db),
```

assertz_fact(bar(X)),
findall(Y, bar(Y), L),
write(L).

139.3 A simple application / a persistent queue

:- module(queue, [main/0], [persdb]).

:- use_package(iso).

:- use_module(library(read)).
:- use_module(library(write)).
:- use_module(library( aggregates)).
persistent_dir(queue_dir,'./pers').

:- persistent(queue/1, queue_dir).
queue(first).
queue(second).

main:-
    write('Action ( in(Term). | slip(Term) | out. | list. | halt. ): '),
    read(A),
    ( handle_action(A)
    -> main
    ; write('Unknown command.'), nl, main).

handle_action(end_of_file) :-
    halt.
handle_action(halt) :-
    halt.
handle_action(in(Term)) :-
    assertz_fact(queue(Term)).
handle_action(slip(Term)) :-
    asserta_fact(queue(Term)).
handle_action(out) :-
    ( retract_fact(queue(Term))
    -> write('Out '), write(Term)
    ; write('FIFO empty.') ),
    nl.
handle_action(list) :-
    findall(Term, queue(Term), Terms),
    write('Contents: '), write(Terms), nl.
140 Filed predicates

Author(s): Francisco Bueno.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#10 (2002/5/30, 19:46:8 CEST)

This package allows using files as a “cache” for predicates defined by facts. This is useful for huge tables of facts that may push the memory limits of the system too far. Goals of a filed predicate are executed simply by reading from the corresponding file.

Anything in the DB file used for the predicate that is different from a fact for the corresponding predicate is ignored. Each call to a filed predicate forces opening the file, so the use of this package is subject to the limit on the number of open files that the system can support.

Dynamic modification of the filed predicates is also allowed during execution of the program. Thus filed predicates are regarded as dynamic, data predicates residing in a file. However, dynamic modifications to the predicates do not affect the file, unless the predicate is also declared persistent.

The package is compatible with persdb in the sense that a predicate can be made both filed and persistent. In this way, the predicate can be used in programs, but it will not be loaded (saving memory), can also be modified during execution, and modifications will persist in the file. Thus, the user interface to both packages is the same (so the DB file must be one for both filing and persistency).

140.1 Usage and interface (factsdb_rt)

- Library usage:
  This facility is used as a package, thus either including factsdb in the package list of the module, or by using the use_package/1 declaration. The facility predicates are defined in library module factsdb_rt.

- Exports:
  - Predicates:
    asserta_fact/1, assertz_fact/1, call/1, current_fact/1, retract_fact/1.
  - Multi-files:
    $factsdb$cached_goal/3, persistent_dir/2, file_alias/2.

- Other modules used:
  - System library modules:
    counters, read, persdb/persdbcache.

140.2 Documentation on exports (factsdb_rt)

asserta_fact/1:

Meta-predicate with arguments: asserta_fact(fact).

Usage: asserta_fact(Fact)

- Description: Version of data_facts:asserta_fact/1 for filed predicates. The current instance of Fact is interpreted as a fact and is added at the beginning of the definition of the corresponding predicate. Therefore, before all the facts filed in the DB file for the predicate. The predicate concerned must be declared as facts; if it is not, then data_facts:asserta_fact/1 is used.
The following properties should hold at call time:

Fact is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

**assertz_fact/1:**

*Meta-predicate* with arguments: `assertz_fact(fact)`.

*Usage:* `assertz_fact(Fact)`

- *Description:* Version of `data_facts:assertz_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is added at the end of the definition of the corresponding predicate. Therefore, after all the facts filed in the DB file for the predicate. The predicate concerned must be declared as `facts`; if it is not, then `data_facts:assertz_fact/1` is used.

**call/1:**

*Meta-predicate* with arguments: `call(fact)`.

*Usage:* `call(Fact)`

- *Description:* Same as `current_fact/1` if the predicate concerned is declared as `facts`. If it is not, an exception is raised.

**current_fact/1:**

*Meta-predicate* with arguments: `current_fact(fact)`.

*Usage:* `current_fact(Fact)`

- *Description:* Version of `data_facts:current_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is unified with an actual fact in the current definition of the corresponding predicate. Therefore, with a fact previously asserted or filed in the DB file for the predicate, if it has not been retracted. The predicate concerned must be declared as `facts`; if it is not, then `data_facts:current_fact/1` is used.

**retract_fact/1:**

*Meta-predicate* with arguments: `retract_fact(fact)`.

*Usage:* `retract_fact(Fact)`
Description: Version of `data_facts:retract_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is unified with an actual fact in the current definition of the corresponding predicate; such a fact is deleted from the predicate definition. This is true even for the facts filed in the DB file for the predicate; but these are NOT deleted from the file (unless the predicate is persistent). The predicate concerned must be declared as facts; if it is not, then `data_facts:retract_fact/1` is used.

The following properties should hold at call time:

- `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

### 140.3 Documentation on multifiles (factsdb_rt)

**$factsdb$cached_goal/3:**

$factsdb$cached_goal(Spec, Spec, Key)

Predicate `Spec` is filed within database `Key`. Programmers should not define this predicate directly in the program.

The predicate is `multifile`.

**persistent_dir/2:**

See `persdb`.

The predicate is `multifile`.

The predicate is of type `data`.

**file_alias/2:**

See `symfnames`. This predicate is used only if `persistent_dir/2` fails.

The predicate is `multifile`.

The predicate is of type `data`.

### 140.4 Documentation on internals (factsdb_rt)

**facts/2:**

Usage: `:- facts(PredDesc, Keyword).`

- **Description:** Declares the predicate `PredDesc` as filed. `Keyword` is the identifier of a location where the file DB for the predicate is kept. The location `Keyword` is described in the `file_alias` predicate, which must contain a fact in which the first argument unifies with `Keyword`.

- **The following properties should hold upon exit:**

`PredDesc` is a Name/Arity structure denoting a predicate name:

```prolog
predname(P/A) :-
  atm(P),
  nnegint(A).  \(\text{(basic_props:predname/1)}\)
```

`Keyword` is an atom corresponding to a directory identifier. (persdbcache:keyword/1)
keyword/1:
See persdbrt. The same conventions for location of DB files apply in both packages.

140.5 Known bugs and planned improvements (factsdb_rt)

- The DB files for persistent predicates have to be used as such from the beginning. Using a DB file for a filed predicate first, and then using it also when making the predicate persistent won’t work. Nor the other way around: using a DB file for a persistent predicate first, and then using it also when making the predicate filed.
Chapter 141: SQL persistent database interface

141 SQL persistent database interface


Version: 1.11#222 (2004/5/24, 13:8:7 CEST)


The purpose of this library is to implement an instance of the generic concept of persistent predicates, where external relational databases are used for storage (see the documentation of the persdb library and [CHGT98,Par97] for details). To this end, this library exports SQL persistent versions of the assertz_fact/1, retract_fact/1 and retractall_fact/1 builtins. Persistent predicates also allow concurrent updates from several programs, since each update is atomic.

The notion of persistence provides a very natural and transparent way to access database relations from a Prolog program. Stub definitions are provided for such predicates which access the database when the predicate is called (using the db_client library). A Prolog to SQL translator is used to generate the required SQL code dynamically (see library pl2sql).

This library also provides facilities for reflecting more complex views of the database relations as Prolog predicates. Such views can be constructed as conjunctions, disjunctions, projections, etc. of database relations. Also, SQL-like aggregation operations are supported.

141.1 Implementation of the Database Interface

The architecture of the low-level implementation of the database interface was defined with two goals in mind:

- to simplify the communication between the Prolog system and the relational database engines as much as possible, and
- to give as much flexibility as possible to the overall system. This includes simultaneous access to several databases, allowing both the databases and clients to reside on the same physical machine or different machines, and allowing the clients to reside in Win95/NT or Unix machines.

In order to allow the flexibility mentioned above, a client-server architecture was chosen. At the server side, a MySQL server connects to the databases using the MySQL. At the client side, a MySQL client interface connects to this server. The server daemon (mysqld) should be running in the server machine; check your MySQL documentation on how to do that.

After the connection is established a client can send commands to the mediator server which will pass them to the corresponding database server, and then the data will traverse in the opposite direction. These messages include logging on and off from the database, sending SQL queries, and receiving the responses.

The low level implementation of the current library is accomplished by providing abstraction levels over the MySQL interface library. These layers of abstraction implement the persistent predicate view, build the appropriate commands for the database using a translator of Prolog goals to SQL commands, issue such commands using the mediator send/receive procedures, parse the responses, and present such responses to the Prolog engine via backtracking.

141.2 Example(s)

:- module(_, _, [persdb_mysql, functions]).

% Some contributions from Guy-Noel Mathieu
:- use_module(library(write)).
:- use_module(library(format)).
:- use_module(user_and_password).

sql_persistent_location(people, db(people, User, Password, HP)):-
  mysql_host_and_port(HP),
  mysql_user(User),
  mysql_password(Password).

:- sql_persistent(
  people(string, string, int), %% Prolog predicate and types
  people(name, sex, age), %% Table name and attributes
  people). %% Database local id

% Low level MySQL interface.
:- use_module(library('persdb_mysql/mysql_client')).

main :-
  nl,
  display('Creating database'), nl,nl,
  create_people_db,
  nl,
  display('Inserting people'), nl,nl,
  insert_people,
  nl,
  display('Showing people'), nl,nl,
  show_people,
  display('Removing John'), nl,nl,
  remove_people(john,_Y,_Z),
  display('Showing people, after removing John'), nl,nl,
  show_people,
  remove_people(_X,female,_Z),
  display('Showing people, after removing female'), nl,nl,
  show_people.

% Create a database and a table of people. Still needs to be ironed out.

create_people_db :-
  mysql_user(User),
  mysql_password(Password),
  mysql_host_and_port(HP),
  mysql_connect(HP, '', User, Password, DbConnection),

  write("mysql_query(DbConnection,
             "drop database if exists people"), nl,
  write("mysql_query(DbConnection, "create database people")), nl,
  write("mysql_query(DbConnection, "use people")), nl,
  write("mysql_query(DbConnection,
"create table people(name char(16) not null, 
  sex text, age int, primary key(name))"), nl, 
mysql_disconnect(DbConnection).

% Inserts people into the 'people' table.

male(john, 15).
male(peter, 24).
male(ralph, 24).
male(bart, 50).
female(kirsten, 24).
female(mary, 17).
female(jona, 12).
female(maija, 34).

%% Tuples are inserted as in the local Prolog dynamic database
insert_people :-
  (male(N, A),
   display('Inserting '),
   display(male(N, A)),
   nl,
   dbassertz_fact(people(N, male, A)), fail
   ; true
  ),
  (female(N, A),
   display('Inserting '),
   display(female(N, A)),
   nl,
   dbassertz_fact(people(N, female, A)), fail
   ; true
  ).

%% Removes people from the 'people' table.

%% Still not working in MySQL due to differences in SQL: working on it.
remove_people(A, B, C) :-
  dbretractall_fact(people(A, B, C)).
remove_people_2(A, B, C) :-
  dbretract_fact(people(A, B, C)),
  display('Removed row '), display(people(A, B, C)), nl, fail.
remove_people_2(_, _, _) :-
display('No more rows'), nl.

show_people :-
    people(Name, Sex, Age),
    display(people(Name, Sex, Age)),
    nl,
    fail.
show_people :-
    display('No more rows'), nl.

141.3 Usage and interface (persdbrt_mysql)

- **Library usage:**
  Typically, this library is used including the `persdb_mysql` package into the package list of the module, or using the `use_package/1` declaration:

  In a module:
  ```prolog
  :- module(bar, [main/1], [persdb_mysql]).
  or
  :- module(bar, [main/1]).
  :- include(library(persdb_mysql)).
  ```

  In a user file:
  ```prolog
  :- use_package([persdb_mysql]).
  or
  :- include(library(persdb_mysql)).
  ```

  This loads the run-time and compile-time versions of the library (`persdbtr_mysql.pl` and `persdbrt_mysql.pl`) and includes some needed declarations.

- **Exports:**
  - **Predicates:**
    - `init_sql_persdb/0`, `dbassertz_fact/1`, `dbretract_fact/1`, `dbcurrent_fact/1`, `dbretractall_fact/1`, `make_sql_persistent/3`, `dbfindall/4`, `dbcall/2`, `sql_query/3`, `sql_get_tables/2`, `sql_table_types/3`.
  - **Multifiles:**
    - `sql_persistent_location/2`.

- **Other modules used:**
  - **System library modules:**
    - `det_hook/det_hook_rt`, `persdb_mysql/db_client_types`, `persdb_mysql/pl2sql`, `persdb_sql_common/sqltypes`, `dynamic`, `terms`, `terms_vars`, `messages`, `lists`, `aggregates`, `persdb_mysql/mysql_client`, `persdb_sql_common/pl2sqlinsert`, `persdb_mysql/delete_compiler/pl2sqldelete`. 
141.4 Documentation on exports (persdbrt_mysql)

init_sql_persdb/0:

Usage:

Description: Internal predicate, used to transform predicates statically declared as persistent (see sql_persistent/3) into real persistent predicates.

PREDICATE

dbassertz_fact/1:

Usage: dbassertz_fact(Fact)

Description: Persistent extension of assertz_fact/1: the current instance of Fact is interpreted as a fact (i.e., a relation tuple) and is added to the end of the definition of the corresponding predicate. If any integrity constraint violation is done (database stored predicates), an error will be displayed. The predicate concerned must be statically (sql_persistent/3) or dynamically (make_sql_persistent/3) declared. Any uninstantiated variables in the Fact will be replaced by new, private variables. Note: assertion of facts with uninstantiated variables not implemented at this time.

Call and exit should be compatible with:
Fact is a fact (a term whose main functor is not ':-'/2). (persdbrt_mysql:fact/1)

The following properties should hold at call time:
Fact is currently a term which is not a free variable. (term_typing:nonvar/1)

PREDICATE

dbretract_fact/1:

Usage: dbretract_fact(Fact)

Description: Persistent extension of retract_fact/1: deletes on backtracking all the facts which unify with Fact. The predicate concerned must be statically (sql_persistent/3) or dynamically (make_sql_persistent/3) declared.

Call and exit should be compatible with:
Fact is a fact (a term whose main functor is not ':-'/2). (persdbrt_mysql:fact/1)

The following properties should hold at call time:
Fact is currently a term which is not a free variable. (term_typing:nonvar/1)

PREDICATE

dbcurrent_fact/1:

Usage: dbcurrent_fact(Fact)

Description: Persistent extension of current_fact/1: the fact Fact exists in the current database. Provides on backtracking all the facts (tuples) which unify with Fact.

Call and exit should be compatible with:
Fact is a fact (a term whose main functor is not ':-'/2). (persdbrt_mysql:fact/1)

The following properties should hold at call time:
Fact is currently a term which is not a free variable. (term_typing:nonvar/1)
dbretractall_fact/1:

**Usage:** dbretractall_fact(Fact)

- **Description:** Persistent extension of retractall_fact/1: when called deletes all the facts which unify with Fact. The predicate concerned must be statically (sql_persistent/3) or dynamically (make_sql_persistent/3) declared.

- **Call and exit should be compatible with:**
  - Fact is a fact (a term whose main functor is not ':-'/2). (persdbrt_mysql:fact/1)
  - The following properties should hold at call time:
    - Fact is currently a term which is not a free variable. (term_typing:nonvar/1)

make_sql_persistent/3:

**Meta-predicate** with arguments: make_sql_persistent(addmodule,?,?,?).

**Usage:** make_sql_persistent(PrologPredTypes, TableAttributes, Keyword)

- **Description:** Dynamic version of the sql_persistent/3 declaration.

- **The following properties should hold upon exit:**
  - PrologPredTypes is a structure describing a Prolog predicate name with its types.
    - (persdbrt_mysql:prologPredTypes/1)
  - TableAttributes is a structure describing a table name and some attributes.
    - (persdbrt_mysql:tableAttributes/1)
  - Keyword is the name of a persistent storage location. (persdbrt_mysql:persLocId/1)

dbfindall/4:

**Meta-predicate** with arguments: dbfindall(?,?,goal,?).

**Usage:** dbfindall(DBId, Pattern, ComplexGoal, Results)

- **Description:** Similar to findall/3, but Goal is executed in database DBId. Certain restrictions and extensions apply to both Pattern and ComplexGoal stemming from the Prolog to SQL translation involved (see the corresponding type definitions for details).

- **Call and exit should be compatible with:**
  - DBId is a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - Pattern is a database projection term. (pl2sql:projterm/1)
  - ComplexGoal is a database query goal. (pl2sql:querybody/1)
  - Results is a list. (basic_props:list/1)

- **The following properties should hold at call time:**
  - DBId is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Pattern is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ComplexGoal is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Results is a free variable. (term_typing:var/1)
**PREDICATE**

**dbcall/2:**

**Usage:** `dbcall(DBId, ComplexGoal)`

- **Description:** Internal predicate, used by the transformed versions of the persistent predicates. Not meant to be called directly by users. It is exported by the library so that it can be used by the transformed versions of the persistent predicates in the modules in which they reside. Sends `ComplexGoal` to database `DBId` for evaluation. `ComplexGoal` must be a call to a persistent predicate which resides in database `DBId`.

- **Call and exit should be compatible with:**
  - `DBId` a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - `ComplexGoal` is a database query goal. (pl2sql:querybody/1)

- **The following properties should hold at call time:**
  - `DBId` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `ComplexGoal` is currently a term which is not a free variable. (term_typing:nonvar/1)

**sql_query/3:**

**Usage:** `sql_query(DBId, SQLString, AnswerTableTerm)`

- **Description:** `AnswerTableTerm` is the response from database `DBId` to the SQL query in `SQLString` to database `DBId`. `AnswerTableTerm` can express a set of tuples, an error answer or a ‘ok’ response (see `answertableterm/1` for details). At the moment, `sql_query/3` log in and out for each query. This should be changed to log in only the first time and log out on exit and/or via a timer in the standard way.

- **Call and exit should be compatible with:**
  - `DBId` a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - `SQLString` is a string containing SQL code. (pl2sql:sqlstring/1)
  - `AnswerTableTerm` is a response from the ODBC database interface. (persdbrt_mysql:answertableterm/1)

- **The following properties should hold at call time:**
  - `DBId` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `SQLString` is currently a term which is not a free variable. (term_typing:nonvar/1)

**sql_get_tables/2:**

**Usage 1:** `sql_get_tables(Location, Tables)`

- **Description:** `Tables` contains the tables available in `Location`.

- **Call and exit should be compatible with:**
  - `Location` is a structure describing a database. (persdbrt_mysql:database_desc/1)
  - `Tables` is a list of atms. (basic_props:list/2)

- **The following properties should hold at call time:**
  - `Location` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Tables` is a free variable. (term_typing:var/1)

**Usage 2:** `sql_get_tables(DbConnection, Tables)`

- **Description:** `Tables` contains the tables available in `DbConnection`. 

- **Call and exit should be compatible with:**
  - `DbConnection` is a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - `Tables` is a list of atms. (basic_props:list/2)

- **The following properties should hold at call time:**
  - `DbConnection` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Tables` is a free variable. (term_typing:var/1)
Call and exit should be compatible with:

- DbConnection a unique identifier of a database session connection. (mysql_client:dbconnection/1)
- Tables is a list of atms. (basic_props:list/2)

The following properties should hold at call time:

- DbConnection is currently a term which is not a free variable. (term_typing:nonvar/1)
- Tables is a free variable. (term_typing:var/1)

**sql_table_types/3:**

Usage 1: sql_table_types(Location, Table, AttrTypes)

- Description: AttrTypes are the attributes and types of Table in Location.
- Call and exit should be compatible with:
  - Location is a structure describing a database. (persdbrt_mysql:database_desc/1)
  - Table is an atom. (basic_props:atm/1)
  - AttrTypes is a list. (basic_props:list/1)
- The following properties should hold at call time:
  - Location is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Table is currently a term which is not a free variable. (term_typing:nonvar/1)
  - AttrTypes is a free variable. (term_typing:var/1)

Usage 2: sql_table_types(DbConnection, Table, AttrTypes)

- Description: AttrTypes are the attributes and types of Table in DbConnection.
- Call and exit should be compatible with:
  - DbConnection a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - Table is an atom. (basic_props:atm/1)
  - AttrTypes is a list. (basic_props:list/1)
- The following properties should hold at call time:
  - DbConnection is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Table is currently a term which is not a free variable. (term_typing:nonvar/1)
  - AttrTypes is a free variable. (term_typing:var/1)

**socketname/1:**

Usage: socketname(IPP)

- Description: IPP is a structure describing a complete TCP/IP port address.

**dbname/1:**

Usage: dbname(DBId)

- Description: DBId is the identifier of an database.
user/1:
   Usage: user(User)
   - Description: User is a user name in the database.

passwd/1:
   Usage: passwd(Passwd)
   - Description: Passwd is the password for the user name in the database.

projterm/1:
   Usage: projterm(DBProjTerm)
   - Description: DBProjTerm is a database projection term.

querybody/1:
   Usage: querybody(DBGoal)
   - Description: DBGoal is a database query goal.

sqltype/1: (UNDOC_REEXPORT)
   Imported from sqltypes (see the corresponding documentation for details).

141.5 Documentation on multifiles (persdbrt_mysql)

sql_persistent_location/2: 
   Relates names of locations (the Keywords) with descriptions of such locations (Locations).
   The predicate is multile.
   The predicate is of type data.
   Usage: sql_persistent_location(Keyword, DBLocation)
   - Description: In this usage, DBLocation is a relational database, in which case the predicate is stored as tuples in the database.
   - The following properties should hold upon exit:
     Keyword is the name of a persistent storage location. (persdbrt_mysql:persLocId/1)
     DBLocation is a structure describing a database. (persdbrt_mysql:database_desc/1)
141.6 Documentation on internals (persdbrt_mysql)

**tuple/1:**

```prolog
tuple(T) :-
    list(T, atm).
```

**Usage:** `tuple(T)`

- **Description:** `T` is a tuple of values from the ODBC database interface.

**dbconnection/1:**

```prolog
Usage: dbconnection(H)
```

- **Description:** `H` a unique identiﬁer of a database session connection.

**sql_persistent/3:**

```prolog
Usage: :- sql_persistent(PrologPredTypes, TableAttributes, Keyword).
```

- **Description:** Declares the predicate corresponding to the main functor of `PrologPredTypes` as SQL persistent. `Keyword` is the name of a location where the persistent storage for the predicate is kept, which in this case must be an external relational database. The description of this database is given through the `sql_persistent_location` predicate, which must contain a fact in which the ﬁrst argument uniﬁes with `Keyword`. `TableAttributes` provides the table name and attributes in the database corresponding respectively to the predicate name and arguments of the (virtual) Prolog predicate.

Although a predicate may be persistent, other usual clauses can be deﬁned in the source code. When querying a persistent predicate with non-persistent clauses, persistent and non-persistent clauses will be evaluated in turn; the order of evaluation is the usual Prolog order, considering that persistent clauses are deﬁned in the program point where the `sql_persistent/3` declaration is.

**Example:**

```prolog
:- sql_persistent(product( integer, integer, string, string ),
                   product( quantity, id, name, size ),
                   radiowebdb).
```

- **sql_persistent_location:**

  ```prolog
sql_persistent_location(radiowebdb,
    db('SQL Anywhere 5.0 Sample', user, pass,
      'r2d5.dia.fi.upm.es':2020)).
```

- **The following properties should hold upon exit:**

  - **PrologPredTypes** is a structure describing a Prolog predicate name with its types.
  - **TableAttributes** is a structure describing a table name and some attributes.
  - **Keyword** is the name of a persistent storage location.
db_query/4:

Usage: db_query(DBId, ProjTerm, Goal, ResultTerm)

- Description: ResultTerm contains all the tuples which are the response from database DBId to the Prolog query Goal, projected onto ProjTerm. Uses pl2sqlstring/3 for the Prolog to SQL translation and sql_query/3 for posing the actual query.

- Call and exit should be compatible with:
  - DBId a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - ProjTerm is a database projection term. (pl2sql:projterm/1)
  - Goal is a database query goal. (pl2sql:querybody/1)
  - ResultTerm is a tuple of values from the ODBC database interface. (persdbrt_mysql:tuple/1)

- The following properties should hold at call time:
  - DBId is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ProjTerm is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Goal is currently a term which is not a free variable. (term_typing:nonvar/1)

db_query_one_tuple/4:

Usage: db_query_one_tuple(DBId, ProjTerm, Goal, ResultTerm)

- Description: ResultTerm is one of the tuples which are the response from database DBId to the Prolog query Goal, projected onto ProjTerm. Uses pl2sqlstring/3 for the Prolog to SQL translation and sql_query_one_tuple/3 for posing the actual query. After last tuple has been reached, a null tuple is unified with ResultTerm, and the connection to the database finishes.

- Call and exit should be compatible with:
  - DBId a unique identifier of a database session connection. (mysql_client:dbconnection/1)
  - ProjTerm is a database projection term. (pl2sql:projterm/1)
  - Goal is a database query goal. (pl2sql:querybody/1)
  - ResultTerm is a predicate containing a tuple. (persdbrt_mysql:answertupleterm/1)

- The following properties should hold at call time:
  - DBId is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ProjTerm is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Goal is currently a term which is not a free variable. (term_typing:nonvar/1)

sql_query_one_tuple/3:

Usage: sql_query_one_tuple(DBId, SQLString, ResultTuple)

- Description: ResultTuple contains an element from the set of tuples which represents the response in DBId to the SQL query SQLString. If the connection is kept, successive calls return consecutive tuples, until the last tuple is reached. Then a null tuple is unified with ResultTuple and the connection is finished (calls to mysql_disconnect/1).
Call and exit should be compatible with:

- **DBId** a unique identifier of a database session connection. 
  
- **SQLString** is a string containing SQL code. 
  
- **ResultTuple** is a tuple of values from the ODBC database interface. 

The following properties should hold at call time:

- **DBId** is currently a term which is not a free variable. 
- **SQLString** is currently a term which is not a free variable. 

141.7 Known bugs and planned improvements (persdbrt_mysql)

- At least in the shell, reloading a file after changing the definition of a persistent predicate does not eliminate the old definition...
- Functionality missing: some questions need to be debugged.
- Warning: still using kludgy string2term and still using some non-uniquified temp files.
- Needs to be unified with the file-based library.
142 Prolog to SQL translator

Author(s): C. Draxler. Adapted by M. Hermenegildo and I. Caballero.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#60 (2003/11/27, 21:23:0 CET)

This library performs translation of Prolog queries into SQL. The code is an adaptation for Ciao of the Prolog to SQL compiler written by Christoph Draxler, CIS Centre for Information and Speech Processing, Ludwig-Maximilians-University Munich, draxler@cis.uni-muenchen.de, Version 1.1. Many thanks to Christoph for allowing us to include this adaptation of his code with Ciao.

The translator needs to know the correspondence between Prolog predicates and the SQL tables in the database. To this end this module exports two multifile predicates, sql__relation/3 and sql__attribute/4. See the description of these predicates for details on how such correspondence is specified.

The main entry points to the translator are pl2sqlstring/3 and pl2sqlterm/3. Details on the types of queries allowed can be found in the description of these predicates.

Example: the following program would print out a term representing the SQL query corresponding to the given Prolog query:

```
%jcf%:- use_module(library('persdb_sql/pl2sql')).
:- use_module(library('persdb_mysql/pl2sql')).
%jcf%
:- use_module(library(strings)).

:- multifile [relation/3,attribute/4].
:- data [relation/3,attribute/4].

relation(product,3,'PRODUCT').
attribute(1,'PRODUCT','ID',int).
attribute(2,'PRODUCT','QUANTITY',int).
attribute(3,'PRODUCT','NAME',string).

main :-
    pl2sqlstring( f(L,K),
    ((product(L,N,a); product(L,N,b)),
     + product(2,3,b),
     L + 2 > avg(Y, Z^product(Z,Y,a)),
     K is N + max(X, product(X,2,b))
    ), T),
    write_string(T).
```

Note: while the translator can be used directly in programs, it is more convenient to use a higher-level abstraction: persistent predicates (implemented in the persdb library). The notion of persistent predicates provides a completely transparent interface between Prolog and relational databases. When using this library, the Prolog to SQL translation is called automatically as needed.
142.1 Usage and interface (pl2sql)

- **Library usage:**
  ```prolog```
  ```:- use_module(library(pl2sql)).```  
- **Exports:**
  - **Predicates:**
    - pl2sqlstring/3, pl2sqlterm/3, sqlterm2string/2.
  - **Regular Types:**
    - querybody/1, projterm/1, sqlstring/1.
  - **Multifiles:**
    - sql__relation/3, sql__attribute/4.
- **Other modules used:**
  - **System library modules:**
    - persdb_sql_common/sqltypes, iso_misc, lists, aggregates, messages.

142.2 Documentation on exports (pl2sql)

**pl2sqlstring/3:**

**Predicate**

**Usage:**

```prolog```
```pl2sqlstring(ProjectionTerm, DatabaseGoal, SQLQueryString)```  
- **Description:** This is the top level predicate which translates complex Prolog goals into the corresponding SQL code.
  
  The query code is prepared in such a way that the result is projected onto the term `ProjectionTerm` (also in a similar way to the first argument of `setof/3`). See the predicate `translate_projection/3` for restrictions on this term.
  
  `SQLQueryString` contains the code of the SQL query, ready to be sent to an SQL server.

- **Call and exit should be compatible with:**
  
  - `ProjectionTerm` is a database projection term. (`pl2sql:projterm/1`)
  - `DatabaseGoal` is a database query goal. (`pl2sql:querybody/1`)
  - `SQLQueryString` is a string containing SQL code. (`pl2sql:sqlstring/1`)

- **The following properties should hold at call time:**
  
  - `ProjectionTerm` is currently a term which is not a free variable. (`term_typing:nonvar/1`)
  - `DatabaseGoal` is currently a term which is not a free variable. (`term_typing:nonvar/1`)
  - `SQLQueryString` is a free variable. (`term_typing:var/1`)

**querybody/1:**

**Regular Type**

`DBGoal` is a goal meant to be executed in the external database. It can be a complex term containing conjunctions, disjunctions, and negations, of:

- Atomic goals, which must have been defined via `sql__relation/3` and `sql__attribute/4` and reside in the (same) database. Their arguments must be either
ground or free variables. If they are ground, they must be bound to constants of the
type declared for that argument. If an argument is a free variable, it may share with
(i.e., be the same variable as) other free variables in other goal arguments.

- Database comparison goals, whose main functor must be a database comparison oper-
  ator (see pl2sql: comparison/2) and whose arguments must be database arithmetic
  expressions.

- Database calls to is/2. The left side of such a call may be either unbound, in which
case it is bound to the result of evaluating the right side, or bound in which case an
equality condition is tested. The right side must be a database arithmetic expression.

The binding of variables follows Prolog rules:

- variables are bound by positive base goals and on the left side of the is/2 predicate.
- Comparison operations, negated goals, and right sides of the is/2 predicate do not
  return variable bindings and may even require all arguments to be bound for a safe
  evaluation.

Database arithmetic expressions may contain:

- Numeric constants (i.e., integers, reals, etc.).
- Bound variables, i.e., variables which will be bound during execution through occurrence within a positive database goal, or by a preceding arithmetic function.
- Database arithmetic functions, which are a subset of those typically accepted within
  is/2 (see pl2sql: arithmetic_functor/2).
- Database aggregation functions, each of which has two arguments: a variable indicating the argument over which the function is to be computed, and a goal argument which must contain in at least one argument position the variable (e.g. avg(Seats, plane(Type, Seats))). The goal argument may only be a conjunction of (positive or negative) base goals. See pl2sql: aggregate_functor/2 for the admissible aggregate functions.

In addition, variables can be existentially quantified using ^/2 (in a similar way to how it
is done in setof/3).

Note that it is assumed that the arithmetic operators in Prolog and SQL are the same,
i.e., + is addition in Prolog and in SQL, etc.

Usage: querybody(DBGoal)
   – Description: DBGoal is a database query goal.

projterm/1:  

REGTYPE 

DBProjTerm is a term onto which the result of a database query code is (in a similar way to the first argument of setof/3)).

A ProjectionTerm must meet the following restrictions:

- The functor of ProjectionTerm may not be one of the built-in predicates, i.e. ', ', '; ',
  etc. are not allowed.

- Only variables and constants are allowed as arguments, i.e., no structured terms may
  appear.

Usage: projterm(DBProjTerm)
   – Description: DBProjTerm is a database projection term.
sqlstring/1:

```prolog
sqlstring(S) :-
    string(S).
```

Usage: sqlstring(S)

- Description: S is a string containing SQL code.

pl2sqlterm/3:

Usage: pl2sqlterm(ProjectionTerm, DatabaseGoal, SQLQueryTerm)

- Description: Similar to pl2sqlstring/3 except that SQLQueryTerm is a representation of the SQL query as a Prolog term.
- Call and exit should be compatible with:
  - ProjectionTerm is a database projection term. (pl2sql:projterm/1)
  - DatabaseGoal is a database query goal. (pl2sql:querybody/1)
  - SQLQueryTerm is a list of sqlterms. (basic_props:list/2)
- The following properties should hold at call time:
  - ProjectionTerm is currently a term which is not a free variable. (term_typing:nonvar/1)
  - DatabaseGoal is currently a term which is not a free variable. (term_typing:nonvar/1)
  - SQLQueryTerm is a free variable. (term_typing:var/1)

sqlterm2string/2:

Usage: sqlterm2string(Queries, QueryString)

- Description: QueryString is a string representation of the list of queries in Prolog-term format in Queries.
- Call and exit should be compatible with:
  - Queries is a list of sqlterms. (basic_props:list/2)
  - QueryString is a string containing SQL code. (pl2sql:sqlstring/1)
- The following properties should hold at call time:
  - Queries is currently a term which is not a free variable. (term_typing:nonvar/1)
  - QueryString is a free variable. (term_typing:var/1)

sqltype/1: (UNDOC_REEXPORT)

Imported from sqltypes (see the corresponding documentation for details).

142.3 Documentation on multifiles (p12sql)
Description: This predicate, together with \texttt{sql__attribute/4}, defines the correspondence between Prolog predicates and the SQL tables in the database. These two relations constitute an extensible meta-database which maps Prolog predicate names to SQL table names, and Prolog predicate argument positions to SQL attributes. \texttt{PredName} is the chosen Prolog name for an SQL table. \texttt{Arity} is the number of arguments of the predicate. \texttt{TableName} is the name of the SQL table in the Database Management System.

- \textit{Call and exit should be compatible with:}
  - \texttt{PredName} is an atom. 
  - \texttt{Arity} is an integer. 
  - \texttt{TableName} is an atom.

\texttt{sql__attribute/4}:

This predicate maps the argument positions of a Prolog predicate to the SQL attributes of its corresponding table. The types of the arguments need to be specified, and this information is used for consistency checking during the translation and for output formatting. A minimal type system is provided to this end. The allowable types are given by \texttt{sqltype/1}.

\texttt{ANumber} is the argument number in the Prolog relation. \texttt{TblName} is the name of the SQL table in the Database Management System. \texttt{AName} is the name of the corresponding attribute in the table. \texttt{AType} is the (translator) data type of the attribute.

- \textit{Call and exit should be compatible with:}
  - \texttt{ANumber} is an integer. 
  - \texttt{TblName} is an atom. 
  - \texttt{AName} is an atom. 
  - \texttt{AType} is an SQL data type supported by the translator.

142.4 Documentation on internals (pl2sql)

\texttt{query_generation/3}:

This predicate translates a list of Prolog conjunctions into SQL queries. It takes a list of Prolog conjunctions, a projection term, and a list of SQL queries as input. Each conjunction is translated into an SQL query, and the queries are connected through a UNION operator.

- \textit{Description:} For each Conjunction in \texttt{ListOfConjunctions}, translate the pair \texttt{(ProjectionTerm, Conjunction)} to an SQL query and connect each such query through a UNION-operator to result in the \texttt{ListOfQueries}.

A Conjunction consists of positive or negative subgoals. Each subgoal is translated as follows:
- the functor of a goal that is not a comparison operation is translated to a relation name with a range variable,
- negated goals are translated to NOT EXISTS-subqueries with * projection,
- comparison operations are translated to comparison operations in the WHERE-clause,
- aggregate function terms are translated to aggregate function (sub)queries.
The arguments of a goal are translated as follows:
- variables of a goal are translated to qualified attributes,
- variables occurring in several goals are translated to equality comparisons (equi join) in the WHERE-clause,
- constant arguments are translated to equality comparisons in the WHERE-clause.

Arithmetic functions are treated specially (translate_arithmetic_function/5). See also querybody/1 for details on the syntax accepted and restrictions.

The following properties should hold at call time:
- ListOfConjunctions is currently a term which is not a free variable. (term_typing:nonvar/1)
- ProjectionTerm is currently a term which is not a free variable. (term_typing:nonvar/1)
- ListOfQueries is a free variable. (term_typing:var/1)

\[
\text{translate_conjunction/5:} \\
\text{Usage: translate_conjunction(Conjunction, SQLFrom, SQLWhere, Dict, NewDict)} \\
\text{Description: Translates a conjunction of goals (represented as a list of goals preceded by existentially quantified variables) to FROM-clauses and WHERE-clauses of an SQL query. A dictionary containing the associated SQL table and attribute names is built up as an accumulator pair (arguments Dict and NewDict).}
\]

\[
\text{translate_goal/5:} \\
\text{Usage: translate_goal(Goal, SQLFrom, SQLWhere, Dict, NewDict)} \\
\text{Description: Translates:} \\
- a positive database goal to the associated FROM- and WHERE clause of an SQL query, 
- a negated database goal to a negated existential subquery, 
- an arithmetic goal to an arithmetic expression or an aggregate function query, 
- a comparison goal to a comparison expression, and 
- a negated comparison goal to a comparison expression with the opposite comparison operator.
\]

\[
\text{translate_arithmetic_function/5:} \\
\text{Usage: translate_arithmetic_function(Result, Expression, SQLWhere, Dict, NewDict)} \\
\text{Description: Arithmetic functions (left side of is/2 operator is bound to value of expression on right side) may be called with either:} \\
- Result unbound: then Result is bound to the value of the evaluation of Expression, 
- Result bound: then an equality condition is returned between the value of Result and the value of the evaluation of Expression. 
\text{Only the equality test shows up in the WHERE clause of an SQL query.}
\]
translate_comparison/5:
PREDICATE
Usage: translate_comparison(LeftArg, RightArg, CompOp, Dict, SQLComparison)

- Description: Translates the left and right arguments of a comparison term into the appropriate comparison operation in SQL. The result type of each argument expression is checked for type compatibility.

aggregate_function/3:
PREDICATE
Usage: aggregate_function(AggregateFunctionTerm, Dict, AggregateFunctionQuery)

- Description: Supports the Prolog aggregate function terms listed in aggregate_functor/2 within arithmetic expressions. Aggregate functions are translated to the corresponding SQL built-in aggregate functions.

comparison/2:
PREDICATE
Usage: comparison(PrologOperator, SQLOperator)

- Description: Defines the mapping between Prolog operators and SQL operators:
  
  comparison(=,=).
  comparison(<,<).
  comparison(>,>).
  comparison(@<,<).
  comparison(@>,>).

- Call and exit should be compatible with:
  PrologOperator is an atom. (basic_props:atm/1)
  SQLOperator is an atom. (basic_props:atm/1)

negated_comparison/2:
PREDICATE
Usage: negated_comparison(PrologOperator, SQLOperator)

- Description: Defines the mapping between Prolog operators and the complementary SQL operators:
  
  negated_comparison(=,<>).
  negated_comparison(\==,=).
  negated_comparison(>,=<).
  negated_comparison(=<,>).
  negated_comparison(<,>=).
  negated_comparison(>=,<).

- Call and exit should be compatible with:
  PrologOperator is an atom. (basic_props:atm/1)
  SQLOperator is an atom. (basic_props:atm/1)

arithmetic_functor/2:
PREDICATE
Usage: arithmetic_functor(PrologFunctor, SQLFunction)

- Description: Defines the admissible arithmetic functions on the Prolog side and their correspondence on the SQL side:
arithmetic_functor(+,+).
arithmetic_functor(−,−).
arithmetic_functor(*,∗).
arithmetic_functor(/,\)/).

– Call and exit should be compatible with:

PrologFunctor is an atom. (basic_props:atm/1)
SQLFunction is an atom. (basic_props:atm/1)

aggregate_functor/2:

Usage: aggregate_functor(PrologFunctor, SQLFunction)
– Description: Defines the admissible aggregate functions on the Prolog side and their correspondence on the SQL side:

aggregate_functor(avg,'AVG').
aggregate_functor(min,'MIN').
aggregate_functor(max,'MAX').
aggregate_functor(sum,'SUM').
aggregate_functor(count,'COUNT').

– Call and exit should be compatible with:

PrologFunctor is an atom. (basic_props:atm/1)
SQLFunction is an atom. (basic_props:atm/1)

142.5 Known bugs and planned improvements (pl2sql)

• Need to separate db predicate names by module.
143 Low-level socket interface to SQL/ODBC databases

Author(s): Jose Morales.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#155 (2004/1/7, 20:9:58 CET)

This library provides a low-level interface to MySQL using the MySQL C API and the Ciao foreign interface to C.

143.1 Usage and interface (mysql_client)

- Library usage:
  :- use_module(library(mysql_client)).
- Exports:
  - Predicates:
    mysql_connect/5, mysql_query/3, mysql_query_one_tuple/3, mysql_free_query_connection/1, mysql_fetch/2, mysql_get_tables/2, mysql_table_types/3, mysql_disconnect/1.
  - Regular Types:
    dbconnection/1, dbqueryconnection/1.
- Other modules used:
  - System library modules:
    foreign_interface/foreign_interface_properties, persdb_mysql/db_client_types.

143.2 Documentation on exports (mysql_client)

mysql_connect/5:

No further documentation available for this predicate.

dbconnection/1:

dbconnection(_1) :-
  address(_1).

Usage: dbconnection(H)
  - Description: H a unique identifier of a database session connection.

mysql_query/3:

No further documentation available for this predicate.

mysql_query_one_tuple/3:

No further documentation available for this predicate.
dbqueryconnection/1: REGTYPE
    dbqueryconnection(_1) :-
        address(_1).
Usage: dbqueryconnection(H)
    – Description: H is a unique identifier of a query answer in a database session connection.

mysql_free_query_connection/1: PREDICATE
    No further documentation available for this predicate.

mysql_fetch/2: PREDICATE
    No further documentation available for this predicate.

mysql_get_tables/2: PREDICATE
    No further documentation available for this predicate.

mysql_table_types/3: PREDICATE
    No further documentation available for this predicate.

mysql_disconnect/1: PREDICATE
    No further documentation available for this predicate.
144 Types for the Low-level interface to SQL databases

Author(s): D. Cabeza, M. Carro, I. Caballero, and M. Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#109 (2003/12/22, 18:8:9 CET)
This module implement the types for the low level interface to SQL databases

144.1 Usage and interface (db_client_types)

- Library usage:
  ```prolog
  :- use_module(library(db_client_types)).
  ```
- Exports:
  - Regular Types:
    socketname/1, dbname/1, user/1, passwd/1, answertableterm/1, tuple/1, answertupleterm/1, sqlstring/1.

144.2 Documentation on exports (db_client_types)

socketname/1:  REGTYPE
  ```prolog
  socketname(IPAddress:PortNumber) :-
  atm(IPAddress),
  int(PortNumber).
  ```
  Usage: socketname(IPP)
  - Description: IPP is a structure describing a complete TCP/IP port address.

dbname/1:  REGTYPE
  ```prolog
  dbname(DBId) :-
  atm(DBId).
  ```
  Usage: dbname(DBId)
  - Description: DBId is the identifier of an database.

user/1:  REGTYPE
  ```prolog
  user(User) :-
  atm(User).
  ```
  Usage: user(User)
  - Description: User is a user name in the database.

passwd/1:  REGTYPE
  ```prolog
  passwd(Passwd) :-
  atm(Passwd).
  ```
  Usage: passwd(Passwd)
  - Description: Passwd is the password for the user name in the database.
answertableterm/1:  
Represents the types of responses that will be returned from the database interface. These can be a set of answer tuples, or the atom `ok` in case of a successful addition or deletion.  
Usage: answertableterm(AT)  
- Description: AT is a response from the database interface.

tuple/1:  
tuple(T) :-  
    list(T, atm).  
Usage: tuple(T)  
- Description: T is a tuple of values from the database interface.

answertupleterm/1:  
answertupleterm([]).  
answertupleterm(tup(T)) :-  
    tuple(T).  
Usage: answertupleterm(X)  
- Description: X is a predicate containing a tuple.

sqlstring/1:  
sqlstring(S) :-  
    string(S).  
Usage: sqlstring(S)  
- Description: S is a string of SQL code.
145 sqltypes (library)

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#127 (2001/10/26, 14:52:5 CEST)

145.1 Usage and interface (sqltypes)

- Library usage:
  :- use_module(library(sqltypes)).
- Exports:
  - Predicates:
    accepted_type/2, get_type/2, type_compatible/2, type_union/3, sybase2sqltypes_list/2, sybase2sqltype/2, postgres2sqltypes_list/2, postgres2sqltype/2.
  - Regular Types:
    sqltype/1, sybasetype/1, postgrestype/1.

145.2 Documentation on exports (sqltypes)

sqltype/1:

REGTYPE
sqltype(int).
sqltype(flt).
sqltype(num).
sqltype(string).
sqltype(date).
sqltype(time).
sqltype(datetime).

These types have the same meaning as the corresponding standard types in the basictypes library.
Usage: sqltype(Type)
- Description: Type is an SQL data type supported by the translator.

accepted_type/2:

PREDICATE
Usage: accepted_type(SystemType, NativeType)
- Description: For the moment, tests whether the SystemType received is a sybase or a postgres type (in the future other systems should be supported) and obtains its equivalent NativeType sqltype.
- Call and exit should be compatible with:
  SystemType is an SQL data type supported by Sybase. (sqltypes:sybasetype/1)
  NativeType is an SQL data type supported by the translator. (sqltypes:sqltype/1)
get_type/2:

Usage: get_type(Constant, Type)

- Description: Prolog implementation-specific definition of type retrievals. CIAO Prolog version given here (ISO).
- Call and exit should be compatible with:
  - Constant is any term. (basic_props:term/1)
  - Type is an SQL data type supported by the translator. (sqltypes:sqltype/1)
- The following properties should hold at call time:
  - Constant is currently a term which is not a free variable. (term_typing:nonvar/1)

type_compatible/2:

Usage: type_compatible(TypeA, TypeB)

- Description: Checks if TypeA and TypeB are compatible types, i.e., they are the same or one is a subtype of the other.
- Call and exit should be compatible with:
  - TypeA is an SQL data type supported by the translator. (sqltypes:sqltype/1)
  - TypeB is an SQL data type supported by the translator. (sqltypes:sqltype/1)

type_union/3:

Usage: type_union(TypeA, TypeB, Union)

- Description: Union is the union type of TypeA and TypeB.
- Call and exit should be compatible with:
  - TypeA is an SQL data type supported by the translator. (sqltypes:sqltype/1)
  - TypeB is an SQL data type supported by the translator. (sqltypes:sqltype/1)
  - Union is an SQL data type supported by the translator. (sqltypes:sqltype/1)

sybasetype/1:

SQL datatypes supported by Sybase for which a translation is defined:

- sybasetype(integer).
- sybasetype(numeric).
- sybasetype(float).
- sybasetype(double).
- sybasetype(date).
- sybasetype(char).
- sybasetype(varchar).
- sybasetype('long varchar').
- sybasetype(binary).
- sybasetype('long binary').
- sybasetype(timestamp).
- sybasetype(time).
- sybasetype(tinyint).

Usage: sybasetype(Type)

- Description: Type is an SQL data type supported by Sybase.
sybase2sqltypes_list/2:
Usage: sybase2sqltypes_list(SybaseTypesList, SQLTypesList)
   - Description: SybaseTypesList is a list of Sybase SQL types. PrologTypesList contains their equivalent SQL-type names in CIAO.
   - The following properties should hold upon exit:
     SybaseTypesList is a list. (basic_props:list/1)
     SQLTypesList is a list. (basic_props:list/1)

sybase2sqltype/2:
Usage: sybase2sqltype(SybaseType, SQLType)
   - Description: SybaseType is a Sybase SQL type name, and SQLType is its equivalent SQL-type name in CIAO.
   - The following properties should hold upon exit:
     SybaseType is an SQL data type supported by Sybase. (sqltypes:sybasetype/1)
     SQLType is an SQL data type supported by the translator. (sqltypes:sqltype/1)

postgrestype/1:
SQL datatypes supported by postgreSQL for which a translation is defined:
   postgresqltype(int2).
   postgresqltype(int4).
   postgresqltype(int8).
   postgresqltype(float4).
   postgresqltype(float8).
   postgresqltype(date).
   postgresqltype(timestamp).
   postgresqltype(time).
   postgresqltype(char).
   postgresqltype(varchar).
   postgresqltype(text).
   postgresqltype(bool).
Usage: postgrestype(Type)
   - Description: Type is an SQL data type supported by postgres.

postgres2sqltypes_list/2:
Usage: postgres2sqltypes_list(PostgresTypesList, SQLTypesList)
   - Description: PostgresTypesList is a list of postgres SQL types. PrologTypesList contains their equivalent SQL-type names in CIAO.
   - The following properties should hold upon exit:
     PostgresTypesList is a list. (basic_props:list/1)
     SQLTypesList is a list. (basic_props:list/1)
postgres2sqltype/2:

Usage: postgres2sqltype(PostgresType, SQLType)

- Description: PostgresType is a postgres SQL type name, and SQLType is its equivalent SQL-type name in CIAO.

- The following properties should hold upon exit:
  PostgresType is an SQL data type supported by postgres. (sqltypes:postgrestype/1)
  SQLType is an SQL data type supported by the translator. (sqltypes:sqltype/1)
146 persdbtr_sql (library)

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#42 (2002/12/13, 17:55:42 CET)

146.1 Usage and interface (persdbtr_sql)

- Library usage:
  :- use_module(library(persdbtr_sql)).
- Exports:
  - Predicates:
    sql_persistent_tr/2, sql_goal_tr/2, dbId/2.
- Other modules used:
  - System library modules:
    dynamic.

146.2 Documentation on exports (persdbtr_sql)

sql_persistent_tr/2:  
No further documentation available for this predicate.

sql_goal_tr/2:  
No further documentation available for this predicate.

dbId/2:  
No further documentation available for this predicate.
The predicate is of type data.
The Ciao Prolog System
Chapter 147: pl2sqlinsert (library)

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#63 (2003/11/29, 2:26:35 CET)

147.1 Usage and interface (pl2sqlinsert)

- Library usage:
  :- use_module(library(pl2sqlinsert)).
- Exports:
  - Predicates:
    - pl2sqlInsert/2.
  - Multifiles:
    - sql__relation/3, sql__attribute/4.
- Other modules used:
  - System library modules:
    - operators, default_predicates, lists.

147.2 Documentation on exports (pl2sqlinsert)

pl2sqlInsert/2: PREDICATE
  No further documentation available for this predicate.

147.3 Documentation on multifiles (pl2sqlinsert)

sql__relation/3: PREDICATE
  No further documentation available for this predicate.
  The predicate is multiﬁle.
  The predicate is of type data.

sql__attribute/4: PREDICATE
  No further documentation available for this predicate.
  The predicate is multiﬁle.
  The predicate is of type data.
Chapter 148: Prolog to Java interface

Author(s): Jesús Correas.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#67 (2003/3/14, 12:48:36 CET)

This module defines the Ciao Prolog to Java interface. This interface allows a Prolog program to start a Java process, create Java objects, invoke methods, set/get attributes (fields), and handle Java events.

This interface only works with JDK version 1.2 or higher.

Although the Java side interface is explained in Javadoc format (it is available at library/javall/javadoc/ in your Ciao installation), the general interface structure is detailed here.

148.1 Prolog to Java Interface Structure

This interface is made up of two parts: a Prolog part and a Java part, running in separate processes. The Prolog part receives requests from a Prolog program and sends them to the Java part through a socket. The Java part receives requests from the socket and performs the actions included in the requests.

If an event is thrown in the Java side, an asynchronous message must be sent away to the Prolog side, in order to launch a Prolog goal to handle the event. This asynchronous communication is performed using a separate socket. The nature of this communication needs the use of threads both in Java and Prolog: to deal with the 'sequential program flow,' and other threads for event handling.

In both sides the threads are automatically created by the context of the objects we use. The user must be aware that different requests to the other side of the interface could run concurrently.

148.1.1 Prolog side of the Java interface

The Prolog side receives the actions to do in the Java side from the user program, and sends them to the Java process through the socket connection. When the action is done in the Java side, the result is returned to the user Prolog program, or the action fails if there is any problem in the Java side.

Prolog data representation of Java elements is very simple in this interface. Java primitive types such as integers and characters are translated into the Prolog corresponding terms, and even some Java objects are translated in the same way (e.g. Java strings). Java objects are represented in Prolog as compound terms with a reference id to identify the corresponding Java object. Data conversion is made automatically when the interface is used, so the Prolog user programs do not have to deal with the complexity of this tasks.

148.1.2 Java side

The Java side of this layer is more complex than the Prolog side. The tasks this part have to deal to are the following:

- Wait for requests from the Prolog side.
- Translate the Prolog terms received in the Prolog 'serialized' form to a more useful Java representation (see the Java interface documentation available at library/javall/javadoc/ in your Ciao installation for details regarding Java representation of Prolog terms).
- Interpret the requests received from the Prolog side, and execute them.
- Handle the set of objects created by or derived from the requests received from de prolog side.
- Handle the events raised in the Java side, and launch the listeners added in the prolog side.
- Handle the exceptions raised in the Java side, and send them to the Prolog side.

In the implementation of the Java side, two items must be carefully designed: the handling of Java objects, and the representation of prolog data structures. The last item is specially important because all the interactions between Prolog and Java are made using Prolog structures, an easy way to standardize the different data management in both sides. Even the requests themselves are encapsulated using Prolog structures. The overload of this encapsulation is not significant in terms of socket traffic, due to the optimal implementation of the prolog serialized term.

The java side must handle the objects created from the Prolog side dynamically, and these objects must be accessed as fast as possible from the set of objects. The Java API provides a powerful implementation of Hash tables that achieves all the requirements of our implementation.

On the other hand, the java representation of prolog terms is made using the inheritance of java classes. In the java side exists a representation of a generic prolog term, implemented as an abstract class in java. Variables, atoms, compound terms, lists, and numeric terms are classes in the java side which inherit from the term class. Java objects can be seen also under the prolog representation as compound terms, where the single argument corresponds to the Hash key of the actual java object in the Hash table referred to before. This behaviour makes the handling of mixed java and prolog elements easy. Prolog goals are represented in the java side as objects which contain a prolog compound term with the term representing the goal. This case will be seen more in depth next, when the java to prolog is explained.

148.2 Java event handling from Prolog

Java event handling is based on a delegation model since version 1.1.x. This approach to event handling is very powerful and elegant, but a user program cannot handle all the events that can arise on a given object: for each kind of event, a listener must be implemented and added specifically. However, the Java 2 API includes a special listener (AWTEventListener) that can manage the internal java event queue.

The prolog to java interface has been designed to emulate the java event handler, and is also based on event objects and listeners. The prolog to java interface implements its own event manager, to handle those events that have prolog listeners associated to the object that raises the event. From the prolog side can be added listeners to objects for specific events. The java side includes a list of goals to launch from the object and event type.

Due to the events nature, the event handler must work in a separate thread to manage the events asynchronously. The java side has its own mechanisms to work this way. The prolog side must be implemented specially for event handling using threads. The communication between java and prolog is also asynchronous, and an additional socket stream is used to avoid interferences with the main socket stream. The event stream will work in this implementation only in one way: from java to prolog. If an event handler needs to send back requests to java, it will use the main socket stream, just like the requests sent directly from a prolog program.

The internal process of register a Prolog event handler to a Java event is shown in the next figure:

When an event raises, the Prolog to Java interface has to send to the Prolog user program the goal to evaluate. Graphically, the complete process takes the tasks involved in the following figure:
148.3 Java exception handling from Prolog

Java exception handling is very similar to the peer prolog handling: it includes some specific statements to trap exceptions from user code. In the java side, the exceptions can be originated from an incorrect request, or can be originated in the code called from the request. Both exception types will be sent to prolog using the main socket stream, allowing the prolog program manage the exception. However, the first kind of exceptions are prefixed, so the user program can distinguish them from the second type of exceptions.

In order to handle exceptions properly using the prolog to java and java to prolog interfaces simultaneously, in both sides of the interface will be filtered those exceptions coming from their own side: this avoids an endless loop of exceptions bouncing from one side to another.

148.4 Usage and interface (javart)

- Library usage:
  ```prolog
  :- use_module(library(javart)).
  ```
- Exports:
  - Predicates:
    ```prolog
    java_start/0, java_start/1, java_start/2, java_stop/0, java_connect/2, java_disconnect/0, java_use_module/1, java_create_object/2, java_delete_object/1, java_invoke_method/2, java_get_value/2, java_set_value/2, java_add_listener/3, java_remove_listener/3.
    ```
  - Regular Types:
    ```prolog
    machine_name/1, java_constructor/1, java_object/1, java_event/1, prolog_goal/1, java_field/1, java_method/1.
    ```
- Other modules used:
  - System library modules:
    ```prolog
    concurrency/concurrency, iso_byte_char, format, lists, read, write, javall/javasock, system.
    ```

148.5 Documentation on exports (javart)

**java_start/0:**

**Usage:**

- Description: Starts the Java server on the local machine, connects to it, and starts the event handling thread.

**java_start/1:**

**Usage:** java_start(Classpath)

- Description: Starts the Java server on the local machine, connects to it, and starts the event handling thread. The Java server is started using the classpath received as argument.
- Call and exit should be compatible with:
  ```prolog
  Classpath is a string (a list of character codes). (basic_props:string/1)
  ```
- The following properties should hold at call time:
  Classpath is currently a term which is not a free variable. (term_typing:nonvar/1)

**java_start/2:**

*Usage:*

- *Description:* Starts the Java server in machine_name (using rsh!), connects to it, and starts the event handling thread. The Java server is started using the classpath received as argument.

- *Call and exit should be compatible with:*
  - Arg1 is currently instantiated to an atom. (term_typing:atom/1)
  - Arg2 is a string (a list of character codes). (basic_props:string/1)
  - *The following properties should hold at call time:*
    - Arg1 is the network name of a machine. (javart:machine_name/1)
    - undefined:classpath(Arg2) (undefined property)

**java_stop/0:**

*Usage:*

- *Description:* Stops the interface terminating the threads that handle the socket connection, and finishing the Java interface server if it was started using java_start/n.

**java_connect/2:**

*Usage:*

- *Description:* Connects to an existing Java interface server running in machine_name and listening at port port_number. To connect to a Java server located in the local machine, use 'localhost' as machine_name.

- *Call and exit should be compatible with:*
  - Arg1 is the network name of a machine. (javart:machine_name/1)
  - Arg2 is an integer. (basic_props:int/1)
  - *The following properties should hold at call time:*
    - Arg1 is the network name of a machine. (javart:machine_name/1)
    - undefined:port_number(Arg2) (undefined property)

**java_disconnect/0:**

*Usage:*

- *Description:* Closes the connection with the java process, terminating the threads that handle the connection to Java. This predicate does not terminate the Java process (this is the disconnection procedure for Java servers not started from Prolog). This predicate should be used when the communication is established with java_connect/2.

**machine_name/1:**

*Usage:*

- *Description:* X is the network name of a machine.
java_constructor/1:  
Usage: java_constructor(X)  
Description: X is a java constructor (structure with functor as constructor full name, and arguments as constructor arguments).

java_object/1:  
Usage: java_object(X)  
Description: X is a java object (a structure with functor '$java.object', and argument an integer given by the java side).

java_event/1:  
Usage: java_event(X)  
Description: X is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener').

prolog_goal/1:  
Usage: prolog_goal(X)  
Description: X is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called.

java_field/1:  
Usage: java_field(X)  
Description: X is a java field (structure on which the functor name is the field name, and the single argument is the field value).

java_use_module/1:  
Usage: java_use_module(Module)  
Description: Loads a module and makes it available from Java.  
Call and exit should be compatible with:  
Module is any term. (basic<Props:term/1)  
The following properties should hold at call time:  
Module is currently a term which is not a free variable. (term_typing:nonvar/1)

java_create_object/2:  
Usage:  
Description: New java object creation. The constructor must be a compound term as defined by its type, with the full class name as functor (e.g., 'java.lang.String'), and the parameters passed to the constructor as arguments of the structure.
- **Call and exit should be compatible with:**
  \( \text{Arg1} \) is a java constructor (structure with functor as constructor full name, and arguments as constructor arguments). (javart:java_constructor/1)
  \( \text{Arg2} \) is a java object (a structure with functor '$java\_object', and argument an integer given by the java side). (javart:java_object/1)

- **The following properties should hold at call time:**
  \( \text{Arg1} \) is a java constructor (structure with functor as constructor full name, and arguments as constructor arguments). (javart:java_constructor/1)
  \( \text{Arg2} \) is a free variable. (term_typing:var/1)

- **The following properties should hold upon exit:**
  \( \text{Arg2} \) is a java object (a structure with functor '$java\_object', and argument an integer given by the java side). (javart:java_object/1)

**java_delete_object/1:**

**Usage:**

- **Description:** Java object deletion. It removes the object given as argument from the Java object table.

- **Call and exit should be compatible with:**
  \( \text{Arg1} \) is a java object (a structure with functor '$java\_object', and argument an integer given by the java side). (javart:java_object/1)

- **The following properties should hold at call time:**
  \( \text{Arg1} \) is a java object (a structure with functor '$java\_object', and argument an integer given by the java side). (javart:java_object/1)

**java_invoke_method/2:**

**Usage:**

- **Description:** Invokes a java method on an object. Given a Java object reference, invokes the method represented with the second argument.

- **Call and exit should be compatible with:**
  \( \text{Arg1} \) is a java object (a structure with functor '$java\_object', and argument an integer given by the java side). (javart:java_object/1)
  \( \text{Arg2} \) is a java method (structure with functor as method name, and arguments as method ones, plus a result argument. This result argument is unified with the atom 'Yes' if the java method returns void). (javart:java_method/1)

- **The following properties should hold at call time:**
  \( \text{Arg1} \) is a java object (a structure with functor '$java\_object', and argument an integer given by the java side). (javart:java_object/1)
  \( \text{Arg2} \) is a java method (structure with functor as method name, and arguments as method ones, plus a result argument. This result argument is unified with the atom 'Yes' if the java method returns void). (javart:java_method/1)

**java_method/1:**

**Usage:** java_method(X)

- **Description:** X is a java method (structure with functor as method name, and arguments as method ones, plus a result argument. This result argument is unified with the atom 'Yes' if the java method returns void).
**java_get_value/2:**

Usage:

- **Description:** Gets the value of a field. Given a Java object as first argument, it instantiates the variable given as second argument. This field must be uninstantiated in the java_field functor, or this predicate will fail.

- **Call and exit should be compatible with:**
  - Arg1 is a java object (a structure with functor `$java_object`, and argument an integer given by the java side).
  - Arg2 is a java field (structure on which the functor name is the field name, and the single argument is the field value).

- **The following properties should hold at call time:**
  - Arg1 is a java object (a structure with functor `$java_object`, and argument an integer given by the java side).
  - Arg2 is a java field (structure on which the functor name is the field name, and the single argument is the field value).

**java_set_value/2:**

Usage:

- **Description:** Sets the value of a Java object field. Given a Java object reference, it assigns the value included in the java_field compound term. The field value in the java_field structure must be instantiated.

- **Call and exit should be compatible with:**
  - Arg1 is a java object (a structure with functor `$java_object`, and argument an integer given by the java side).
  - Arg2 is a java field (structure on which the functor name is the field name, and the single argument is the field value).

- **The following properties should hold at call time:**
  - Arg1 is a java object (a structure with functor `$java_object`, and argument an integer given by the java side).
  - Arg2 is a java field (structure on which the functor name is the field name, and the single argument is the field value).

**java_add_listener/3:**

Meta-predicate with arguments: `java_add_listener(?,,goal)`. 

Usage:

- **Description:** Adds a listener to an event on an object. Given a Java object reference, it registers the goal received as third argument to be launched when the Java event raises.

- **Call and exit should be compatible with:**
  - Arg1 is a java object (a structure with functor `$java_object`, and argument an integer given by the java side).
  - Arg2 is a java event represented as an atom with the full event constructor name (e.g., `java.awt.event.ActionListener`).
  - Arg3 is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or
even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)

- The following properties should hold at call time:
  - Arg1 is a java object (a structure with functor 'java_object', and argument an integer given by the java side).
  - Arg2 is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener').
  - Arg3 is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)

java_remove_listener/3:

Usage:
- Description: It removes a listener from an object event queue. Given a Java object reference, goal registered for the given event is removed.
- Call and exit should be compatible with:
  - Arg1 is a java object (a structure with functor 'java_object', and argument an integer given by the java side).
  - Arg2 is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener').
  - Arg3 is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)
- The following properties should hold at call time:
  - Arg1 is a java object (a structure with functor 'java_object', and argument an integer given by the java side).
  - Arg2 is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener').
  - Arg3 is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)
149 Java to Prolog interface

Author(s): Jesús Correas.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#65 (2003/3/14, 12:48:10 CET)

This module defines the Prolog side of the Java to Prolog interface. This side of the interface only has one public predicate: a server that listens at the socket connection with Java, and executes the commands received from the Java side.

In order to evaluate the goals received from the Java side, this module can work in two ways: executing them in the same engine, or starting a thread for each goal. The easiest way is to launch them in the same engine, but the goals must be evaluated sequentially: once a goal provides the first solution, all the subsequent goals must be finished before this goal can backtrack to provide another solution. The Prolog side of this interface works as a top-level, and the goals partially evaluated are not independent.

The solution of this goal dependence is to evaluate the goals in a different prolog engine. Although Ciao includes a mechanism to evaluate goals in different engines, the approach used in this interface is to launch each goal in a different thread.

The decision of what kind of goal evaluation is selected is done by the Java side. Each evaluation type has its own command terms, so the Java side can choose the type it needs.

A Prolog server starts by calling the prolog_server/0 predicate, or by calling prolog_server/1 predicate and providing the port number as argument. The user predicates and libraries to be called from Java must be included in the executable file, or be accessible using the built-in predicates dealing with code loading.

149.1 Usage and interface (jtopl)

Library usage:
:- use_module(library(jtopl)).

Exports:
- Predicates:
  prolog_server/0, prolog_server/1, prolog_server/2, shell_s/0,
  query_solutions/2, query_requests/2, running_queries/2.

Other modules used:
- System library modules:
  concurrency/concurrency, system, read, write, dynamic, lists, format,
  compiler/compiler, atom2term, javall/javasock, prolog_sys.

149.2 Documentation on exports (jtopl)

prolog_server/0:

Usage:
- Description: Prolog server entry point. Reads from the standard input the node name and port number where the java client resides, and starts the prolog server listening at the jp socket. This predicate acts as a server: it includes an endless read-process loop until the prolog_halt command is received.
However, from the low-level communication point of view, this Prolog server actually works as a client of the Java side. This means that Java side waits at the given port to a Prolog server trying to create a socket; Prolog side connects to that port, and then waits for Java requests (acting as a 'logical' server). To use this Prolog server as a real server waiting for connections at a given port, use \texttt{prolog_server/1}.

\textbf{prolog_server/1:} \\
\textbf{Usage:} \\
\textit{Description:} Waits for incoming Java connections to act as a Prolog goal server for Java requests. This is the only \texttt{prolog_server/ *} predicate that works as a true server: given a port number, waits for a connection from Java and then serves Java requests. When a termination request is received, finishes the connection to Java and waits next Java connection request. This behaviour is different with respect to previous versions of this library. To work as before, use \texttt{prolog_server/2}.

Although it currently does not support simultaneous Java connections, some work is being done in that direction.

\textit{Call and exit should be compatible with:} \\
Arg1 is an atom. \quad (\texttt{basic_props:atm/1})

\textbf{prolog_server/2:} \\
\textbf{Usage:} \\
\textit{Description:} Prolog server entry point. Given a network node and a port number, starts the prolog server trying to connect to Java side at that node:port address, and then waits for Java requests. This predicate acts as a server: it includes an endless read-process loop until the \texttt{prolog_halt} command is received.

However, from the low-level communication point of view, this Prolog server actually works as a client of the Java side. This means that Java side waits at the given port to a Prolog server trying to create a socket; Prolog side connects to that port, and then waits for Java requests (acting as a 'logical' server). To use this Prolog server as a real server waiting for connections at a given port, use \texttt{prolog_server/1}.

\textit{Call and exit should be compatible with:} \\
Arg1 is an atom. \quad (\texttt{basic_props:atm/1}) \\
Arg2 is an atom. \quad (\texttt{basic_props:atm/1})

\textbf{shell_s/0:} \\
\textbf{Usage:} \\
\textit{Description:} Command execution loop. This predicate is called when the connection to Java is established, and performs an endless loop processing the commands received. This predicate is only intended to be used by the Prolog to Java interface and it should not be used by a user program.

\textbf{query_solutions/2:} \\
No further documentation available for this predicate. \\
The predicate is of type \texttt{concurrent}. 

\texttt{PREDICATE}
query_requests/2:
   No further documentation available for this predicate.
   The predicate is of type concurrent.

running_queries/2:
   No further documentation available for this predicate.
   The predicate is of type concurrent.
150 Low-level Prolog to Java socket connection

Author(s): Jesús Correas.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#66 (2003/3/14, 12:48:24 CET)

This module defines a low-level socket interface, to be used by javart and jtopl. Includes all the code related directly to the handling of sockets. This library should not be used by any user program, because is a very low-level connection to Java. Use javart (Prolog to Java interface) or jtopl (Java to Prolog interface) libraries instead.

150.1 Usage and interface (javasock)

- Library usage:
  ```prolog
  :- use_module(library(javasock)).
  ```
- Exports:
  - Predicates:
    ```prolog
    bind_socket_interface/1, start_socket_interface/2, stop_socket_interface/0, join_socket_interface/0, java_query/2, java_response/2, prolog_query/2, prolog_response/2, is_connected_to_java/0, java_debug/1, java_debug_redo/1, start_threads/0.
    ```
- Other modules used:
  - System library modules:
    ```prolog
    fastrw, read, sockets/sockets, dynamic, format, concurrency/concurrency, javall/jtopl, sockets/sockets_io.
    ```

150.2 Documentation on exports (javasock)

**bind_socket_interface/1:**

Usage: `bind_socket_interface(Port)`

Description: Given an port number, waits for a connection request from the Java side, creates the sockets to connect to the java process, and starts the threads needed to handle the connection.

Call and exit should be compatible with:
- `Port` is an integer. (basic_props:int/1)
- The following properties should hold at call time:
  - `Port` is currently a term which is not a free variable. (term_typing:nonvar/1)

**start_socket_interface/2:**

Usage: `start_socket_interface(Address, Stream)`

Description: Given an address in format 'node:port', creates the sockets to connect to the java process, and starts the threads needed to handle the connection.

Call and exit should be compatible with:
- `Address` is any term. (basic_props:term/1)
- `Stream` is an open stream. (streams_basic:stream/1)
The following properties should hold at call time:

- Address is currently a term which is not a free variable.  \( \text{term_typing:nonvar/1} \)
- Stream is currently a term which is not a free variable.  \( \text{term_typing:nonvar/1} \)

**stop_socket_interface/0:**

**Usage:**
- **Description:** Closes the sockets to disconnect from the java process, and waits until the threads that handle the connection terminate.

**join_socket_interface/0:**

**Usage:**
- **Description:** Waits until the threads that handle the connection terminate.

**java_query/2:**

The predicate is of type **concurrent**.

**Usage:** `java_query(ThreadId, Query)`
- **Description:** Data predicate containing the queries to be sent to Java. First argument is the Prolog thread Id, and second argument is the query to send to Java.
- **Call and exit should be compatible with:**
  - ThreadId is an atom.  \( \text{basic_props:atm/1} \)
  - Query is any term.  \( \text{basic_props:term/1} \)

**java_response/2:**

The predicate is of type **concurrent**.

**Usage:** `java_response(Id, Response)`
- **Description:** Data predicate that stores the responses to requests received from Java. First argument corresponds to the Prolog thread Id; second argument corresponds to the response itself.
- **Call and exit should be compatible with:**
  - Id is an atom.  \( \text{basic_props:atm/1} \)
  - Response is any term.  \( \text{basic_props:term/1} \)

**prolog_query/2:**

The predicate is of type **concurrent**.

**Usage:** `prolog_query(Id, Query)`
- **Description:** Data predicate that keeps a queue of the queries requested to Prolog side from Java side.
- **Call and exit should be compatible with:**
  - Id is an integer.  \( \text{basic_props:int/1} \)
  - Query is any term.  \( \text{basic_props:term/1} \)
prolog_response/2:

- The predicate is of type concurrent.
- **Usage:** prolog_response(Id, Response)
  - **Description:** Data predicate that keeps a queue of the responses to queries requested to Prolog side from Java side.
  - **Call and exit should be compatible with:**
    - Id is an integer. (basic_props:int/1)
    - Response is any term. (basic_props:term/1)

is_connected_to_java/0:

- **Usage:**
  - **Description:** Checks if the connection to Java is established.

java_debug/1:

- No further documentation available for this predicate.

java_debug_redo/1:

- No further documentation available for this predicate.

start_threads/0:

- **Usage:**
  - **Description:** Starts the threads that will handle the connection to Java. This predicate is declared public for internal purposes, and it is not intended to be used by a user program.
151 Calling emacs from Prolog

Author(s): The CLIP Group.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#110 (2003/12/22, 18:14:0 CET)

This library provides a prolog-emacs interface. This interface is complementary to (and independent from) the emacs mode, which is used to develop programs from within the emacs editor/environment. Instead, this library allows calling emacs from a running Prolog program. This facilitates the use of emacs as a “user interface” for a Prolog program. Emacs can be made to:

- Visit a file, which can then be edited.
- Execute arbitrary emacs lisp code, sent from Prolog.

In order for this library to work correctly, the following is needed:

- You should be running the emacs editor on the same machine where the executable calling this library is executing.
- This emacs should be running the emacs server. This can be done by including the following line in your .emacs file:

  ```lisp
  ;; Start a server that emacsclient can connect to.
  (server-start)
  ```
  
  Or typing M-x server-start within emacs.

This suffices for using emacs to edit files. For running arbitrary code the following also needs to be added to the .emacs file:

```
(setq enable-local-eval t)
  Allows executing lisp code without asking.

(setq enable-local-eval nil)
  Does not allow executing lisp code without asking.

(setq enable-local-eval 'maybe)
  Allows executing lisp code only if user agrees after asking (asks interactively for every invocation).
```

Examples:

Assuming that a .pl file loads this library, then:

```
..., emacs_edit('foo'), ...
Opens file foo for editing in emacs.

..., emacs_eval_nowait("(run-ciao-toplevel)")
Starts execution of a Ciao top-level within emacs.
```
151.1 Usage and interface (emacs)

- **Library usage:**
  
  ```prolog
  :- use_module(library(emacs)).
  ```

- **Exports:**
  - **Predicates:**
    - `emacs_edit/1`, `emacs_edit_nowait/1`, `emacs_eval/1`, `emacs_eval_nowait/1`.
  - **Regular Types:**
    - `elisp_string/1`.

- **Other modules used:**
  - **System library modules:**
    - `terms_check`, `lists`, `terms`, `system`.

151.2 Documentation on exports (emacs)

**`emacs_edit/1`:**

**Usage:**

- **Description:** Opens the given file for editing in `emacs`. Waits for editing to finish before continuing.
- **The following properties should hold at call time:**
  
  Arg1 is an atom which is the name of a file.

**`emacs_edit_nowait/1`:**

**Usage:**

- **Description:** Opens the given file for editing in `emacs` and continues without waiting for editing to finish.
- **The following properties should hold at call time:**
  
  Arg1 is an atom which is the name of a file.

**`emacs_eval/1`:**

**Usage:**

- **Description:** Executes in `emacs` the lisp code given as argument. Waits for the command to finish before continuing.
- **The following properties should hold at call time:**
  
  Arg1 is a string containing `emacs` lisp code.

**`emacs_eval_nowait/1`:**

**Usage:**

- **Description:** Executes in `emacs` the lisp code given as argument and continues without waiting for it to finish.
The following properties should hold at call time:

- Arg1 is a string containing emacs lisp code.

elisp_string/1:

- Usage: elisp_string(L)
- Description: L is a string containing emacs lisp code.
Chapter 152: linda (library)

Version: 0.9#66 (1999/4/29, 12:28:0 MEST)

This is a SICStus-like linda package. Note that this is essentially quite obsolete, and provided mostly in case it is needed for compatibility, since Ciao now supports all Linda functionality (and more) through the concurrent fact database.

152.1 Usage and interface (linda)

- Library usage:
  ```prolog```
  ```
  :- use_module(library(linda)).
  ```

- Exports:
  ```prolog```
  ```
  Predicates:
  linda_client/1, close_client/0, in/1, in/2, in_noblock/1, out/1, rd/1, rd/2, rd_noblock/1, rd_findall/3, linda_timeout/2, halt_server/0, open_client/2, in_stream/2, out_stream/2.
  ```

- Other modules used:
  ```prolog```
  ```
  System library modules:
  read, fastrw, sockets/sockets.
  ```

152.2 Documentation on exports (linda)

linda_client/1:
No further documentation available for this predicate.

```
```

close_client/0:
No further documentation available for this predicate.

```
```

in/1:
No further documentation available for this predicate.

```
```

in/2:
No further documentation available for this predicate.

```
```

in_noblock/1:
No further documentation available for this predicate.

```
```

out/1:
No further documentation available for this predicate.
rd/1:  
No further documentation available for this predicate.

rd/2:  
No further documentation available for this predicate.

rd_noblock/1:  
No further documentation available for this predicate.

rd_findall/3:  
No further documentation available for this predicate.

linda_timeout/2:  
No further documentation available for this predicate.

halt_server/0:  
No further documentation available for this predicate.

open_client/2:  
No further documentation available for this predicate.

in_stream/2:  
No further documentation available for this predicate.

out_stream/2:  
No further documentation available for this predicate.
PART IX - Abstract data types

Author(s): The CLIP Group.

This part includes libraries which implement some generic data structures (abstract data types) that are used frequently in programs or in the Ciao system itself.
153 Extendable arrays with logarithmic access time

Author(s): Lena Flood.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#112 (2003/12/22, 18:18:22 CET)

This module implements extendable arrays with logarithmic access time. It has been adapted
from shared code written by David Warren and Fernando Pereira.

153.1 Usage and interface (arrays)

- Library usage:
  :- use_module(library(arrays)).
- Exports:
  - Predicates:
    new_array/1, is_array/1, aref/3, arefa/3, arefl/3, aset/4, array_to_list/2.

153.2 Documentation on exports (arrays)

new_array/1:  
Usage: new_array(Array)
Description: returns an empty new array Array.
The following properties should hold at call time:
  Array is a free variable.  

is_array/1:  
Usage: is_array(Array)
Description: Array actually is an array.
The following properties should hold at call time:
  Array is currently a term which is not a free variable.  

aref/3:  
Usage: aref(Index, Array, Element)
Description: unifies Element to Array[Index], or fails if Array[Index] has not been set.
The following properties should hold at call time:
  Index is currently a term which is not a free variable. 
  Array is currently a term which is not a free variable.
**arefa/3:**

**Usage:** arefa(Index, Array, Element)

- **Description:** is as aref/3, except that it unifies Element with a new array if Array[Index] is undefined. This is useful for multidimensional arrays implemented as arrays of arrays.

- The following properties should hold at call time:
  - Index is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Array is currently a term which is not a free variable. (term_typing:nonvar/1)

**arefl/3:**

**Usage:** arefl(Index, Array, Element)

- **Description:** is as aref/3, except that Element appears as [] for undefined cells. Thus, arefl(_,_,[]) always succeeds no matter what you give in the first or second args.

- The following properties should hold at call time:
  - Index is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Array is currently a term which is not a free variable. (term_typing:nonvar/1)

**aset/4:**

**Usage:** aset(Index, Array, Element, NewArray)

- **Description:** unifies NewArray with the result of setting Array[Index] to Element.

- The following properties should hold at call time:
  - Index is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Array is currently a term which is not a free variable. (term_typing:nonvar/1)
  - NewArray is a free variable. (term_typing:var/1)

**array_to_list/2:**

**Usage:** array_to_list(Array, List)

- **Description:** returns a List of pairs Index-Element of all the elements of Array that have been set.

- The following properties should hold at call time:
  - Array is currently a term which is not a free variable. (term_typing:nonvar/1)
  - List is a free variable. (term_typing:var/1)
154 counters (library)

Version: 0.4#5 (1998/2/24)

154.1 Usage and interface (counters)

- Library usage:
  :- use_module(library(counters)).
- Exports:
  - Predicates:
    setcounter/2, getcounter/2, inccounter/2.

154.2 Documentation on exports (counters)

setcounter/2: No further documentation available for this predicate.

getcounter/2: No further documentation available for this predicate.

inccounter/2: No further documentation available for this predicate.
Chapter 155: Identity lists

Author(s): Francisco Bueno.
Version: 1.11#145 (2004/1/1, 14:1:41 CET)

The operations in this module handle lists by performing equality checks via identity instead of unification.

155.1 Usage and interface (idlists)

- Library usage:
  ```prolog```
  ```
  :- use_module(library(idlists)).
  ```

- Exports:
  ```prolog```
  ```
  Predicates:
  member_0/2, memberchk/2, list_insert/2, add_after/4, add_before/4, delete/3, subtract/3, union_idlists/3.
  ```

155.2 Documentation on exports (idlists)

**member_0/2:**

```prolog```

```prolog```

**memberchk/2:**

```prolog```

**list_insert/2:**

```prolog```

**add_after/4:**

```prolog```
The following properties should hold at call time:

- \( L_0 \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( E_0 \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( E \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( L \) is a free variable. \((\text{term_typing:var/1})\)

**add_before/4:**

**Usage:** `add_before(L0, E0, E, L)`

**Description:** Adds element \( E \) before the first element identical to \( E_0 \) (or at start) of list \( L_0 \), returning in \( L \) the new list.

- \( L_0 \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( E_0 \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( E \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( L \) is a free variable. \((\text{term_typing:var/1})\)

**delete/3:**

**Usage:** `delete(List, Element, Rest)`

**Description:** \( \text{Rest} \) has the same elements of \( \text{List} \) except for all the occurrences of elements identical to \( \text{Element} \).

- \( \text{List} \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( \text{Element} \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( \text{Rest} \) is a free variable. \((\text{term_typing:var/1})\)

**subtract/3:**

**Usage:** `subtract(Set, Set0, Difference)`

**Description:** \( \text{Difference} \) has the same elements of \( \text{Set} \) except those which have an identical occurrence in \( \text{Set0} \).

- \( \text{Set} \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( \text{Set0} \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( \text{Difference} \) is a free variable. \((\text{term_typing:var/1})\)

**union_idlists/3:**

**Usage:** `union_idlists(List1, List2, List)`

**Description:** \( \text{List} \) has the elements which are in \( \text{List1} \) but are not identical to an element in \( \text{List2} \) followed by the elements in \( \text{List2} \).

- \( \text{List1} \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( \text{List2} \) is currently a term which is not a free variable. \((\text{term_typing:nonvar/1})\)
- \( \text{List} \) is a free variable. \((\text{term_typing:var/1})\)
156 Lists of numbers

Author(s): The CLIP Group.

Version: 1.11#113 (2003/12/22, 18:22:36 CET)

This module implements some kinds of lists of numbers.

156.1 Usage and interface (numlists)

- Library usage:
  :- use_module(library(numlists)).
- Exports:
  - Predicates:
    get_primes/2, sum_list/2, sum_list/3, sum_list_of_lists/2, sum_list_of_lists/3.
  - Regular Types:
    intlist/1, numlist/1.
- Other modules used:
  - System library modules:
    lists.

156.2 Documentation on exports (numlists)

get_primes/2:

Usage: get_primes(N, Primes)
Description: Computes the Nth first prime numbers in ascending order.
The following properties should hold at call time:
  N is an integer. (basic Props:int/1)
The following properties should hold upon exit:
  Primes is a list of integers. (numlists:intlist/1)

intlist/1:

Usage: intlist(X)
Description: X is a list of integers.

numlist/1:

Usage: numlist(X)
Description: X is a list of numbers.
sum_list/2: PREDICATE
Usage: sum_list(List, N)
   Description: N is the total sum of the elements of List.
   The following properties should hold at call time:
   List is a list of numbers. (numlists:numlist/1)
   The following properties should hold upon exit:
   N is a number. (basic_props:num/1)

sum_list/3: PREDICATE
Usage: sum_list(List, N0, N)
   Description: N is the total sum of the elements of List plus N0.
   The following properties should hold at call time:
   List is a list of numbers. (numlists:numlist/1)
   N0 is a number. (basic_props:num/1)
   The following properties should hold upon exit:
   N is a number. (basic_props:num/1)

sum_list_of_lists/2: PREDICATE
Usage: sum_list_of_lists(Lists, N)
   Description: N is the total sum of the elements of the lists of Lists.
   The following properties should hold at call time:
   List is a list of numlists. (basic_props:list/2)
   The following properties should hold upon exit:
   N is a number. (basic_props:num/1)

sum_list_of_lists/3: PREDICATE
Usage: sum_list_of_lists(Lists, N0, N)
   Description: N is the total sum of the elements of the lists of Lists plus N0.
   The following properties should hold at call time:
   List is a list of numlists. (basic_props:list/2)
   N0 is a number. (basic_props:num/1)
   The following properties should hold upon exit:
   N is a number. (basic_props:num/1)
157 Pattern (regular expression) matching

**Author(s):** The CLIP Group.

**Version:** 1.11#222 (2004/5/24, 13:8:7 CEST)

**Version of last change:** 1.11#133 (2003/12/30, 23:32:28 CET)

This library provides facilities for matching strings and terms against patterns (i.e., regular expressions).

157.1 Usage and interface (patterns)

- **Library usage:**
  
  ```prolog
  :- use_module(library(patterns)).
  ```

- **Exports:**
  - *Predicates:*
    - `match_pattern/2`, `match_pattern/3`, `case_insensitive_match/2`, `letter_match/2`, `match_pattern_pred/2`.
  - *Regular Types:*
    - `pattern/1`.

- **Other modules used:**
  - *System library modules:*
    - `lists`.

157.2 Documentation on exports (patterns)

**match_pattern/2:**

**Usage:** `match_pattern(Pattern, String)`

- **Description:** Matches `String` against `Pattern`. For example, `match_pattern("*.pl","foo.pl")` succeeds.

- **The following properties should hold at call time:**
  - `Pattern` is a pattern to match against. (patterns:pattern/1)
  - `String` is a string (a list of character codes). (basic_props:string/1)

**match_pattern/3:**

**Usage:** `match_pattern(Pattern, String, Tail)`

- **Description:** Matches `String` against `Pattern`. `Tail` is the remainder of the string after the match. For example, `match_pattern("??*","foo.pl",Tail)` succeeds, instantiating `Tail` to "o.pl".

- **The following properties should hold at call time:**
  - `Pattern` is a pattern to match against. (patterns:pattern/1)
  - `String` is a string (a list of character codes). (basic_props:string/1)
  - `Tail` is a string (a list of character codes). (basic_props:string/1)
case_insensitive_match/2:
Usage: case_insensitive_match(Pred1, Pred2)
Description: Tests if two predicates Pred1 and Pred2 match in a case-insensitive way.

letter_match/2:
Usage: letter_match(X, Y)
Description: True iff X and Y represents the same letter

pattern/1:
Special characters for Pattern are:
* Matches any string, including the null string.
? Matches any single character.
[...] Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a range; any character lexically between those two characters, inclusive, is matched. If the first character following the [ is a ^ then any character not enclosed is matched. No other character is special inside this construct. To include a ] in a character set, you must make it the first character. To include a ‘-’, you must use it in a context where it cannot possibly indicate a range: that is, as the first character, or immediately after a range.
| Specifies an alternative. Two patterns A and B with | in between form an expression that matches anything that either A or B will match.
{"} Groups alternatives inside larger patterns.
\ Quotes a special character (including itself).
Usage: pattern(P)
Description: P is a pattern to match against.

match_pattern_pred/2:
Usage: match_pattern_pred(Pred1, Pred2)
Description: Tests if two predicates Pred1 and Pred2 match using regular expressions.
158 Graphs

Author(s): Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#114 (2003/12/22, 18:25:48 CET)
This module implements utilities for work with graphs

158.1 Usage and interface (graphs)

- Library usage:
  
  ```prolog
  :- use_module(library(graphs)).
  ```

- Exports:
  - Predicates:
    - dgraph_to_ugraph/2, dlgraph_to_lgraph/2, edges_to_ugraph/2, edges_to_lgraph/2.
  - Regular Types:
    - dgraph/1, dlgraph/1.
  - Other modules used:
    - System library modules:
      - sort, graphs/ugraphs, graphs/lgraphs.

158.2 Documentation on exports (graphs)

dgraph/1: REGTYPE
  dgraph(Graph)
  A directed graph is a term graph(V,E) where V is a list of vertices and E is a list of edges (none necessarily sorted). Edges are pairs of vertices which are directed, i.e., (a,b) represents a->b. Two vertices a and b are equal only if a==b.
  Usage: dgraph(Graph)
  - Description: Graph is a directed graph.

dlgraph/1: REGTYPE
  dlgraph(Graph)
  A labeled directed graph is a directed graph where edges are triples of the form (a,l,b) where l is the label of the edge (a,b).
  Usage: dlgraph(Graph)
  - Description: Graph is a directed labeled graph.

dgraph_to_ugraph/2: PREDICATE
  Usage: dgraph_to_ugraph(Graph, UGraph)
  - Description: Converts Graph to UGraph.
The following properties should hold at call time:

- Graph is currently a term which is not a free variable. (term_typing:nonvar/1)
- UGraph is a free variable. (term_typing:var/1)
- Graph is a directed graph. (graphs:dgraph/1)
- UGraph is a free variable. (term_typing:var/1)

The following properties should hold upon exit:

- Graph is a directed graph. (graphs:dgraph/1)
- UGraph is an ugraph. (ugraphs:ugraph/1)

**dlgraph_to_lgraph/2:**

*Usage:* dlgraph_to_lgraph(Graph, LGraph)

*Description:* Converts Edges to LGraph.

- The following properties should hold at call time:
  - Graph is currently a term which is not a free variable. (term_typing:nonvar/1)
  - LGraph is a free variable. (term_typing:var/1)
  - Graph is a directed labeled graph. (graphs:dlgraph/1)
  - LGraph is a free variable. (term_typing:var/1)

- The following properties should hold upon exit:
  - Graph is a directed labeled graph. (graphs:dlgraph/1)
  - LGraph is a labeled graph of term terms. (lgraphs:lgraph/2)

**edges_to_ugraph/2:**

*Usage:* edges_to_ugraph(Edges, UGraph)

*Description:* Converts Graph to UGraph.

- The following properties should hold at call time:
  - Edges is currently a term which is not a free variable. (term_typing:nonvar/1)
  - UGraph is a free variable. (term_typing:var/1)
  - Edges is a list of pairs. (basic_props:list/2)
  - UGraph is a free variable. (term_typing:var/1)

- The following properties should hold upon exit:
  - Edges is a list of pairs. (basic_props:list/2)
  - UGraph is an ugraph. (ugraphs:ugraph/1)

**edges_to_lgraph/2:**

*Usage:* edges_to_lgraph(Edges, LGraph)

*Description:* Converts Edges to LGraph.

- The following properties should hold at call time:
  - Edges is currently a term which is not a free variable. (term_typing:nonvar/1)
  - LGraph is a free variable. (term_typing:var/1)
  - Edges is a list of triples. (basic_props:list/2)
  - LGraph is a free variable. (term_typing:var/1)

- The following properties should hold upon exit:
  - Edges is a list of triples. (basic_props:list/2)
  - LGraph is a labeled graph of term terms. (lgraphs:lgraph/2)
158.3 Documentation on internals (graphs)

**pair/1:**

**Usage:** pair(P)

- **Description:** P is a pair (_,_).

**triple/1:**

**Usage:** triple(P)

- **Description:** P is a triple (_,_,_).
159 Unweighted graph-processing utilities

Author(s): M. Carlsson, adapted from shared code written by Richard A O'Keefe. Mods by F.Bueno and M.Carro.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#64 (2003/12/9, 7:11:15 CET)

An unweighted directed graph (ugraph) is represented as a list of (vertex-neighbors) pairs, where the pairs are in standard order (as produced by keysort with unique keys) and the neighbors of each vertex are also in standard order (as produced by sort), and every neighbor appears as a vertex even if it has no neighbors itself.

An undirected graph is represented as a directed graph where for each edge \((U,V)\) there is a symmetric edge \((V,U)\).

An edge \((U,V)\) is represented as the term \(U-V\).

A vertex can be any term. Two vertices are distinct iff they are not identical (\(==/2\)).

A path is represented as a list of vertices. No vertex can appear twice in a path.

159.1 Usage and interface (ugraphs)

- Library usage:
  
  ```prolog
  :- use_module(library(ugraphs)).
  ```

- Exports:
  
  - Predicates:
    
    ```prolog
    vertices_edges_to_ugraph/3, neighbors/3, edges/2, del_vertices/3, vertices/2, add_vertices/3, add_edges/3, transpose/2, rooted_subgraph/3, point_to/3.
    ```
  
  - Regular Types:
    
    ```prolog
    ugraph/1.
    ```
  
- Other modules used:
  
  ```prolog
  dict, sets, sort.
  ```

159.2 Documentation on exports (ugraphs)

vertices_edges_to_ugraph/3: PREDICATE

No further documentation available for this predicate.

neighbors/3: PREDICATE

Usage: neighbors(Vertex, Graph, Neighbors)

- Description: Is true if Vertex is a vertex in Graph and Neighbors are its neighbors.
- The following properties should hold at call time:

  ```prolog
  Vertex is currently a term which is not a free variable. (term_typing:nonvar/1)
  Graph is currently a term which is not a free variable. (term_typing:nonvar/1)
  Neighbors is a free variable. (term_typing:var/1)
  ```
edges/2:
Usage: edges(Graph, Edges)
Description: Unifies Edges with the edges in Graph.
The following properties should hold at call time:
Graph is currently a term which is not a free variable. (term_typing:nonvar/1)
Edges is a free variable. (term_typing:var/1)

del_vertices/3:
Usage: del_vertices(Graph1, Vertices, Graph2)
Description: Is true if Graph2 is Graph1 with Vertices and all edges to and from Vertices removed from it.
The following properties should hold at call time:
Graph1 is currently a term which is not a free variable. (term_typing:nonvar/1)
Vertices is currently a term which is not a free variable. (term_typing:nonvar/1)
Graph2 is a free variable. (term_typing:var/1)

vertices/2:
Usage: vertices(Graph, Vertices)
Description: Unifies Vertices with the vertices in Graph.
The following properties should hold at call time:
Graph is currently a term which is not a free variable. (term_typing:nonvar/1)
Vertices is a free variable. (term_typing:var/1)

add_vertices/3:
Usage: add_vertices(Graph1, Vertices, Graph2)
Description: Is true if Graph2 is Graph1 with Vertices added to it.
The following properties should hold at call time:
Graph1 is currently a term which is not a free variable. (term_typing:nonvar/1)
Vertices is currently a term which is not a free variable. (term_typing:nonvar/1)
Graph2 is a free variable. (term_typing:var/1)

add_edges/3:
Usage: add_edges(Graph1, Edges, Graph2)
Description: Is true if Graph2 is Graph1 with Edges and their 'to' and 'from' vertices added to it.
The following properties should hold at call time:
Graph1 is currently a term which is not a free variable. (term_typing:nonvar/1)
Edges is currently a term which is not a free variable. (term_typing:nonvar/1)
Graph2 is a free variable. (term_typing:var/1)
**transpose/2:**  
**Usage:** `transpose(Graph, Transpose)`  
**Description:** Is true if `Transpose` is the graph computed by replacing each edge \((u,v)\) in `Graph` by its symmetric edge \((v,u)\). It can only be used one way around. The cost is \(O(N^2)\).  
**The following properties should hold at call time:**  
- `Graph` is currently a term which is not a free variable.  
- `Transpose` is a free variable.

**rooted_subgraph/3:**  
**Usage:** `rooted_subgraph(Graph, Sources, SubGraph)`  
**Description:** `SubGraph` is the subgraph of `Graph` which is reachable from `Sources`.  
**The following properties should hold at call time:**  
- `Graph` is an ugraph.  
- `Sources` is a list.  
- `SubGraph` is a free variable.  
**The following properties hold upon exit:**  
- `SubGraph` is an ugraph.

**point_to/3:**  
**Usage:** `point_to(Vertex, Graph, Point_to)`  
**Description:** Is true if `Point_to` is the list of nodes which go directly to `Vertex` in `Graph`.  
**The following properties should hold at call time:**  
- `Vertex` is currently a term which is not a free variable.  
- `Graph` is currently a term which is not a free variable.  
- `Point_to` is a free variable.

**ugraph/1:**  
**Usage:** `ugraph(Graph)`  
**Description:** `Graph` is an ugraph.
160 wgraphs (library)

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 0.4#5 (1998/2/24)

160.1 Usage and interface (wgraphs)

- Library usage:
  ```prolog```
  ```
  :- use_module(library(wgraphs)).
  ```
- Exports:
  - Predicates:
    ```prolog```
    ```
    vertices_edges_to_wgraph/3.
    ```
- Other modules used:
  - System library modules:
    ```prolog```
    ```
    sets, sort.
    ```

160.2 Documentation on exports (wgraphs)

vertices_edges_to_wgraph/3:

No further documentation available for this predicate.
161 Labeled graph-processing utilities

Author(s): Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#134 (2003/12/30, 23:45:39 CET)
See the comments for the ugraphs library.

161.1 Usage and interface (lgraphs)

- Library usage:
  :- use_module(library(lgraphs)).
- Exports:
  – Predicates:
    vertices_edges_to_lgraph/3.
  – Regular Types:
    lgraph/2.
- Other modules used:
  – System library modules:
    sort, sets.

161.2 Documentation on exports (lgraphs)

lgraph/2: REGTYPE
Usage: lgraph(Graph, Type)
  – Description: Graph is a labeled graph of Type terms.

vertices_edges_to_lgraph/3: PREDICATE
vertices_edges_to_lgraph(Vertices0, Edges, Graph)
This one is a copy of the same procedure in library(wgraphs) except for the definition of min/3 (ah! - the polymorphism!).
It would only be needed if there are multi-edges, i.e., several edges between the same two vertices.
162 queues (library)

Version: 0.4#5 (1998/2/24)

162.1 Usage and interface (queues)

- Library usage:
  \[ 
  :- \text{use} \text{\_module(library(queues))}. 
  \]

- Exports:
  - Predicates:
    \[ q\_empty/1, q\_insert/3, q\_member/2, q\_delete/3. \]

162.2 Documentation on exports (queues)

\texttt{q\_empty/1:}

No further documentation available for this predicate.

\texttt{q\_insert/3:}

No further documentation available for this predicate.

\texttt{q\_member/2:}

No further documentation available for this predicate.

\texttt{q\_delete/3:}

No further documentation available for this predicate.
163 Random numbers

Author(s): Daniel Cabeza.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#275 (2004/1/9, 16:25:9 CET)

This module provides predicates for generating pseudo-random numbers

163.1 Usage and interface (random)

- Library usage:
  ```prolog
  :- use_module(library(random)).
  ```

- Exports:
  - Predicates:
    `random/1`, `random/3`, `srandom/1`.

163.2 Documentation on exports (random)

random/1: PREDICATE

```prolog
random(Number)
```

**Number** is a (pseudo-) random number in the range [0.0,1.0]

**Usage:*
- *The following properties should hold at call time:*
  - **Number** is a free variable. *(term_typing:var/1)*
- *The following properties hold upon exit:*
  - **Number** is currently instantiated to a float. *(term_typing:float/1)*

random/3: PREDICATE

```prolog
random(Low, Up, Number)
```

**Number** is a (pseudo-) random number in the range [Low, Up]

**Usage 1:**
- *Description: If Low and Up are integers, Number is an integer.*
- *The following properties should hold at call time:*
  - **Low** is an integer. *(basic_props:int/1)*
  - **Up** is an integer. *(basic_props:int/1)*
  - **Number** is a free variable. *(term_typing:var/1)*
- *The following properties hold upon exit:*
  - **Number** is an integer. *(basic_props:int/1)*

**Usage 2:**
- *The following properties should hold at call time:*
  - **Low** is a float. *(basic_props:flt/1)*
  - **Up** is a number. *(basic_props:num/1)*
  - **Number** is a free variable. *(term_typing:var/1)*
The Ciao Prolog System

The following properties hold upon exit:
Number is a float. (basic_props:flt/1)

Usage 3:
- The following properties should hold at call time:
  Low is an integer. (basic_props:int/1)
  Up is a float. (basic_props:flt/1)
  Number is a free variable. (term_typing:var/1)
- The following properties hold upon exit:
  Number is a float. (basic_props:flt/1)

srandom/1:
PREDICATE
srandom(Seed)
Changes the sequence of pseudo-random numbers according to Seed. The stating sequence of numbers generated can be duplicated by calling the predicate with Seed unbound (the sequence depends on the OS).
Usage:
- Calls should, and exit will be compatible with:
  Seed is an integer. (basic_props:int/1)
- The following properties hold upon exit:
  Seed is an integer. (basic_props:int/1)
164 Set Operations

Author(s): Lena Flood.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#115 (2003/12/22, 18:32:34 CET)

This module implements set operations. Sets are just ordered lists.

164.1 Usage and interface (sets)

- Library usage:
  :- use_module(library(sets)).

- Exports:
  Predicates:
  insert/3, ord_delete/3, ord_member/2, ord_test_member/3, ord_subtract/3,
  ord_intersection/3, ord_intersection_diff/4, ord_intersect/2, ord_subset/2,
  ord_subset_diff/3, ord_union/3, ord_union_diff/4, ord_union_symdiff/4, ord_ union_change/3, merge/3, ord_disjoint/2, setproduct/3.

- Other modules used:
  System library modules:
  sort.

164.2 Documentation on exports (sets)

insert/3:
Usage: insert(Set1, Element, Set2)
  Description: It is true when Set2 is Set1 with Element inserted in it, preserving the order.
  The following properties should hold at call time:
  Set1 is currently a term which is not a free variable. (term_typing:nonvar/1)
  Element is currently a term which is not a free variable. (term_typing:nonvar/1)
  Set2 is a free variable. (term_typing:var/1)

ord_delete/3:
Usage: ord_delete(Set0, X, Set)
  Description: It succeeds if Set is Set0 without element X.
  The following properties should hold at call time:
  Set0 is currently a term which is not a free variable. (term_typing:nonvar/1)
  X is currently a term which is not a free variable. (term_typing:nonvar/1)
  Set is a free variable. (term_typing:var/1)
ord_member/2:
Usage: ord_member(X, Set)
- Description: It succeeds if \( X \) is member of \( \text{Set} \).
- The following properties should hold at call time:
  \( X \) is currently a term which is not a free variable.
  \( \text{Set} \) is currently a term which is not a free variable.

ord_test_member/3:
Usage: ord_test_member(Set, X, Result)
- Description: If \( X \) is member of \( \text{Set} \) then Result=\text{yes}. Otherwise Result=\text{no}.
- The following properties should hold at call time:
  \( \text{Set} \) is currently a term which is not a free variable.
  \( X \) is currently a term which is not a free variable.
  Result is a free variable.

ord_subtract/3:
Usage: ord_subtract(Set1, Set2, Difference)
- Description: It is true when Difference contains all and only the elements of \( \text{Set1} \) which are not also in \( \text{Set2} \).
- The following properties should hold at call time:
  \( \text{Set1} \) is currently a term which is not a free variable.
  \( \text{Set2} \) is currently a term which is not a free variable.

ord_intersection/3:
Usage: ord_intersection(Set1, Set2, Intersection)
- Description: It is true when Intersection is the ordered representation of \( \text{Set1} \) and \( \text{Set2} \), provided that \( \text{Set1} \) and \( \text{Set2} \) are ordered lists.
- The following properties should hold at call time:
  \( \text{Set1} \) is currently a term which is not a free variable.
  \( \text{Set2} \) is currently a term which is not a free variable.

ord_intersection_diff/4:
Usage: ord_intersection_diff(Set1, Set2, Intersect, NotIntersect)
- Description: Intersect contains those elements which are both in \( \text{Set1} \) and \( \text{Set2} \), and NotIntersect those which are in \( \text{Set1} \) but not in \( \text{Set2} \).
- The following properties should hold at call time:
  \( \text{Set1} \) is currently a term which is not a free variable.
  \( \text{Set2} \) is currently a term which is not a free variable.
  Intersect is a free variable.
  NotIntersect is a free variable.
ord_intersect/2:  
Usage: ord_intersect(Xs, Ys)  
- Description: Succeeds when the two ordered lists have at least one element in common.  
- The following properties should hold at call time:  
  Xs is currently a term which is not a free variable.  
  Ys is currently a term which is not a free variable.

ord_subset/2:  
Usage: ord_subset(Xs, Ys)  
- Description: Succeeds when every element of Xs appears in Ys.  
- The following properties should hold at call time:  
  Xs is currently a term which is not a free variable.  
  Ys is currently a term which is not a free variable.

ord_subset_diff/3:  
Usage: ord_subset_diff(Set1, Set2, Difference)  
- Description: It succeeds when every element of Set1 appears in Set2 and Difference has the elements of Set2 which are not in Set1.  
- The following properties should hold at call time:  
  Set1 is currently a term which is not a free variable.  
  Set2 is currently a term which is not a free variable.  
  Difference is a free variable.

ord_union/3:  
Usage: ord_union(Set1, Set2, Union)  
- Description: It is true when Union is the union of Set1 and Set2. When some element occurs in both sets, Union retains only one copy.  
- The following properties should hold at call time:  
  Set1 is currently a term which is not a free variable.  
  Set2 is currently a term which is not a free variable.

ord_union_diff/4:  
Usage: ord_union_diff(Set1, Set2, Union, Difference)  
- Description: It succeeds when Union is the union of Set1 and Set2, and Difference is Set2 set-minus Set1.  
- The following properties should hold at call time:  
  Set1 is currently a term which is not a free variable.  
  Set2 is currently a term which is not a free variable.  
  Union is a free variable.  
  Difference is a free variable.
ord_union_symdiff/4:  
  Usage: ord_union_symdiff(Set1, Set2, Union, Diff)  
  – Description: It is true when Diff is the symmetric difference of Set1 and Set2, and Union is the union of Set1 and Set2.  
  – The following properties should hold at call time:  
    Set1 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Set2 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Union is a free variable. (term_typing:var/1)  
    Diff is a free variable. (term_typing:var/1)

ord_union_change/3:  
  Usage: ord_union_change(Set1, Set2, Union)  
  – Description: Union is the union of Set1 and Set2 and Union is different from Set2.  
  – The following properties should hold at call time:  
    Set1 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Set2 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Union is a free variable. (term_typing:var/1)

merge/3:  
  Usage: merge(Set1, Set2, Union)  
  – Description: See ord_union/3.  
  – The following properties should hold at call time:  
    Set1 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Set2 is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_disjoint/2:  
  Usage: ord_disjoint(Set1, Set2)  
  – Description: Set1 and Set2 have no element in common.  
  – The following properties should hold at call time:  
    Set1 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Set2 is currently a term which is not a free variable. (term_typing:nonvar/1)

setproduct/3:  
  Usage: setproduct(Set1, Set2, Product)  
  – Description: Product has all two element sets such that one element is in Set1 and the other in Set2, except that if the same element belongs to both, then the corresponding one element set is in Product.  
  – The following properties should hold at call time:  
    Set1 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Set2 is currently a term which is not a free variable. (term_typing:nonvar/1)  
    Product is a free variable. (term_typing:var/1)
165 Variable name dictionaries

Author(s): Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#35 (2003/8/28, 16:15:20 CEST)

165.1 Usage and interface (vndict)

- Library usage:
  :- use_module(library(vndict)).
- Exports:
  - Predicates:
    - create_dict/2, complete_dict/3, complete_vars_dict/3, prune_dict/3, sort_dict/2, dict2varnamesl/2, varnames12dict/2, find_name/4, rename/2, vars_names_dict/3.
  - Regular Types:
    - null_dict/1, varname/1, varnamesl/1, varnamedict/1.
- Other modules used:
  - System library modules:
    - idlists, terms_vars, sets, sort.

165.2 Documentation on exports (vndict)

null_dict/1:
  Usage: null_dict(D)
  Description: D is an empty dictionary.

create_dict/2:
  Usage: create_dict(Term, Dict)
  Description: Dict has names for all variables in Term.
  The following properties should hold at call time:
  Term is any term. (basic_props:term/1)
  The following properties should hold upon exit:
  Dict is a dictionary of variable names. (vndict:varnamedict/1)

complete_dict/3:
  Usage: complete_dict(Dict, Term, NewDict)
  Description: NewDict is Dict augmented with the variables of Term not yet in Dict.
  The following properties should hold at call time:
  Dict is currently a term which is not a free variable. (term_typing:nonvar/1)
  Term is currently a term which is not a free variable. (term_typing:nonvar/1)
  NewDict is a free variable. (term_typing:var/1)
**complete_vars_dict/3:**

Usage: `complete_vars_dict(Dict, Vars, NewDict)`

- Description: `NewDict` is `Dict` augmented with the variables of the list `Vars` not yet in `Dict`.
- The following properties should hold at call time:
  - `Dict` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Vars` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `NewDict` is a free variable. (term_typing:var/1)

**prune_dict/3:**

Usage: `prune_dict(Term, Dict, NewDict)`

- Description: `NewDict` is `Dict` reduced to just the variables of `Term`.
- The following properties should hold at call time:
  - `Term` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Dict` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `NewDict` is a free variable. (term_typing:var/1)

**sort_dict/2:**

Usage: `sort_dict(D, Dict)`

- Description: `D` is sorted into `Dict`.
- The following properties should hold at call time:
  - `D` is a dictionary of variable names. (vndict:varnamedict/1)
- The following properties should hold upon exit:
  - `Dict` is a dictionary of variable names. (vndict:varnamedict/1)

**dict2varnamesl/2:**

Usage: `dict2varnamesl(Dict, VNs)`

- Description: Translates `Dict` to `VN`.s
- The following properties should hold at call time:
  - `Dict` is a dictionary of variable names. (vndict:varnamedict/1)
- The following properties should hold upon exit:
  - `VN` is a list of Name=Var, for a variable `Var` and its name `Name`. (vndict:varnamesl/1)

**varnamesl2dict/2:**

Usage: `varnamesl2dict(VNs, Dict)`

- Description: Translates `VN` to `Dict`.
- The following properties should hold at call time:
  - `VN` is a list of Name=Var, for a variable `Var` and its name `Name`. (vndict:varnamesl/1)
- The following properties should hold upon exit:
  - `Dict` is a dictionary of variable names. (vndict:varnamedict/1)
find_name/4:

find_name(Vars, Names, V, Name)

Given that \texttt{vars_names_dict(Dict, Vars, Names)} holds, it acts as \texttt{rename(X, Dict)}, but the name of X is given as \texttt{Name} instead of unified with it.

rename/2:

Usage: \texttt{rename(Term, Dict)}

\begin{itemize}
  \item Description: Unifies each variable in \texttt{Term} with its name in \texttt{Dict}. If no name is found, a new name is created.
  \item The following properties should hold at call time:
  \hspace{1cm} \texttt{Dict} is a dictionary of variable names.
\end{itemize}

\hspace{1cm} (\texttt{vndict:varnamedict/1})

varname/1:

Usage: \texttt{varname(N)}

\begin{itemize}
  \item Description: \texttt{N} is term representing a variable name.
\end{itemize}

varnamesl/1:

Usage: \texttt{varnamesl(D)}

\begin{itemize}
  \item Description: \texttt{D} is a list of Name=Var, for a variable Var and its name Name.
\end{itemize}

varnamedict/1:

Usage: \texttt{varnamedict(D)}

\begin{itemize}
  \item Description: \texttt{D} is a dictionary of variable names.
\end{itemize}

vars_names_dict/3:

Usage: \texttt{vars_names_dict(varnamedict(Dict), list(Vars), list(Names))}

\begin{itemize}
  \item Description: \texttt{Vars} is a sorted list of variables, and \texttt{Names} is a list of their names, which correspond in the same order.
\end{itemize}
PART X - Miscellaneous standalone utilities

Author(s): clip@clip.dia.fi.upm.es, http://www.clip.dia.fi.upm.es/, The CLIP Group, School of Computer Science, Technical University of Madrid.

This is the documentation for a set of miscellaneous standalone utilities contained in the etc directory of the Ciao distribution.
166 A Program to Help Cleaning your Directories

Author(s): Manuel Carro.
Version: 0.1#3 (2001/10/25, 14:31:59 CEST)

A simple program for traversing a directory tree and deciding which files may be deleted in order to save space and not to lose information.

166.1 Usage (cleandirs)

cleandirs [--silent] <initial_dir> <delete_options> <backup_options>
cleandirs explores <initial_dir> (which should be an absolute path) and looks for backup files and files which can be generated from other files, using a plausible heuristic aimed at retaining the same amount of information while recovering some disk space. The heuristic is based on the extension of the filename.

Delete options is one of:
--list: just list the files/directories which are amenable to be deleted, but do not delete them. SAFE.
--ask: list the files/directories and ask for deletion. UNSAFE if you make a mistake.
--delete: just delete the files/directories without asking. I envy your brave soul if you choose this option.

Backup options is one of:
--includebackups: include backup files in the list of files to check.
--excludebackups: do not include backup files in the list of files to check.
--onlybackups: include only backup files in the list of files to check.

Symbolic links are not traversed. Special files are not checked.

Invoking the program with no arguments will return an up-to-date information on the options.

166.2 Known bugs and planned improvements (cleandirs)

- Recursive removal of subdirectories relies on the existence of a recursive /bin/rm command in your system.
Chapter 167: Printing the declarations and code in a file

Author(s): Manuel Hermenegildo.

Version: 0.5#7 (2003/7/29, 18:7:56 CEST)

A simple program for printing assertion information (predicate declarations, property declarations, type declarations, etc.) and printing code-related information (imports, exports, libraries used, etc.) on a file. The file should be a single Ciao or Prolog source file. It uses the Ciao compiler’s pass one to do it. This program is specially useful for example for checking what assertions the assertion normalizer is producing from the original assertions in the file or to check what the compiler is actually seeing after some of the syntactic expansions (but before goal translations).

167.1 Usage (fileinfo)

fileinfo -asr <filename.asr>
: pretty prints the contents of <filename.asr>

fileinfo [-v] [-m] [-a|-f|-c|-e] <filename> [libdir1] ... [libdirN]
-v : verbose output (e.g., lists all files read)
-m : restrict info to current module
-a : print assertions
-f : print code and interface (imports/exports, etc.)
-c : print code only
-e : print only errors - useful to check syntax of assertions in file

fileinfo -h
: print this information

Note that system lib paths *must* be given explicitly, e.g.:

fileinfo -m -c foo.pl /
/home/clip/System/ciao/lib /
/home/clip/System/ciao/library /

167.2 More detailed explanation of options (fileinfo)

- If the -a option is selected, fileinfo prints the assertions (only code-oriented assertions – not comment-oriented assertions) in the file after normalization. If the -f option is selected fileinfo prints the file interface, the declarations contained in the file, and the actual code. The -c option prints only the code. If the -e option is selected fileinfo prints only any syntactic and import-export errors found in the file, including the assertions.
- filename must be the name of a Prolog or Ciao source file.
- This filename can be followed by other arguments which will be taken to be library directory paths in which to look for files used by the file being analyzed.
- If the -m option is selected, only the information related to the current module is printed.
- The -v option produces verbose output. This is very useful for debugging, since all the files accessed during assertion normalization are listed.
- In the -asr usage, fileinfo can be used to print the contents of a .asr file in human-readable form.
168 Printing the contents of a bytecode file

Author(s): Daniel Cabeza.
Version: 0.5#2 (1999/11/11, 19:20:50 MET)

This simple program takes as an argument a bytecode (.po) file and prints out in symbolic form the information contained in the file. It uses compiler and engine builtins to do so, so that it keeps track with changes in bytecode format.

168.1 Usage (viewpo)

viewpo <file1>.po
   : print .po contents in symbolic form

viewpo -h
   : print this information
169 Crossed-references of a program

Author(s): Francisco Bueno.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#135 (2003/12/30, 23:50:8 CET)

The *xrefs* crossed-references Ciao library includes several modules which allow displaying crossed-references of the code in a program. Crossed-references identify modules which import code from other modules, or files (be them modules or not) which use code in other files. Crossed-references can be obtained as a term representing a graph, displayed graphically (using daVinci, a graph displayer developed by U. of Bremen, Germany), or printed as a list.

The libraries involved are as follows:
- *etc(xmrefs)* displays a graph of crossed-references between modules using daVinci,
- *etc(xfrefs)* displays a graph of crossed-references between files using daVinci,
- *library(xrefs)* obtains a graph of crossed-references between files,
- *library('xrefs/mrefs')* obtains a graph of crossed-references between modules,
- *library('xrefs/pxrefs')* prints a list of crossed-references between files.

The first two are intended to be used by loading in *ciaosh*. The other three are intended to be used as modules within an application.

The following is an example graph of the library modules involved in the crossed-references application. It has been obtained with:

```
[ciao/etc]> ciaosh
Ciao-Prolog 1.5 #24: Tue Dec 28 14:12:11 CET 1999
?- use_module(xmrefs).
yes
?- set_flag(X).
X = 3 ?
yes
?- set_files([xfrefs, xmrefs, library(xrefs), library('xrefs/mrefs'), library('xrefs/pxrefs'), library('xrefs/xrefs2graph'), library('xrefs/xrefsbuild'), library('xrefs/xrefsread')]).
yes
?- xmrefs.
```
The following is an example graph of the same module files, where crossed-references have been obtained with `xfrefs:xfrefs(whodefs)` instead of `xmrefs:xmrefs`:

For more information refer to the xrefs documentation (`xrefs_doc.dvi`) in the source library of the Ciao distribution.
Chapter 170: callgraph (library)  

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)  
Version of last change: 1.11#65 (2003/12/10, 1:19:38 CET)

170 callgraph (library)

170.1 Usage and interface (callgraph)

- Library usage:
  `:- use_module(library(callgraph)).`
- Exports:
  - Predicates:
    `call_graph/2, reachability/4`.
- Other modules used:
  - System library modules:
    `assertions/c_itf_props, sets, terms, graphs/ugraphs, xrefs/xrefsread`.

170.2 Documentation on exports (callgraph)

**call_graph/2:**

- Usage: `call_graph(File, Graph)`
- *Description:* `Graph` is the call-graph of the code in `File`.
- *The following properties should hold at call time:*
  - `File` is an atom describing the name of a file. `
c_itf_props:filename/1`
  - `Graph` is a free variable. `
term_typing:var/1`
- *The following properties hold upon exit:*
  - `Graph` is an ugraph. `
ugraphs:ugraph/1`

**reachability/4:**

- Usage: `reachability(Graph, Sources, Reached, UnReached)`
- *Description:* `Reached` are the vertices in `Graph` reachable from `Sources`, `UnReached` are the rest.
- *The following properties should hold at call time:*
  - `Graph` is an ugraph. `
ugraphs:ugraph/1`
  - `Sources` is a list. `
basic_props:list/1`
  - `Reached` is a free variable. `
term_typing:var/1`
  - `UnReached` is a free variable. `
term_typing:var/1`
171 Gathering the dependent files for a file

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This simple program takes a single Ciao or Prolog source filename (which is typically the main file of an application). It prints out the list of all the dependent files, i.e., all files needed in order to build the application, including those which reside in libraries. This is particularly useful in Makefiles, for building standalone distributions (e.g., .tar files) automatically.

The filename should be followed by other arguments which will be taken to be library directory paths in which to look for files used by the file being analyzed.

171.1 Usage (get_deps)

```bash
get_deps [-u <filename>] <filename> [lib_dir1] ... [lib_dirN]
    : return dependent files for <filename>
    : found in [lib_dir1] ... [lib_dirN]

get_deps -h
    : print this information
```
172 Finding differences between two Prolog files

Author(s): Francisco Bueno.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#136 (2003/12/30, 23:51:20 CET)

This simple program works like the good old diff but for files that contain Prolog code. It prints out the clauses that it finds are different in the files. Its use avoids textual differences such as different variable names and different formatting of the code in the files.

172.1 Usage (pldiff)

pldiff <file1> <file2>
   : find differences

pldiff -h
   : print this information

but you can also use the program as a library and invoke the predicate:

pldiff( <filename> , <filename> )

172.2 Known bugs and planned improvements (pldiff)

- Currently uses variant/2 to compare clauses. This is useful, but there should be an option to select the way clauses are compared, e.g., some form of equivalence defined by the user.
Chapter 173: The Ciao lpmake scripting facility

Author(s): Manuel Hermenegildo, clip@dia.fi.upm.es, http://www.clip.dia.fi.upm.es/, The CLIP Group, Facultad de Informática, Universidad Politécnica de Madrid.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#27 (2002/11/20, 13:4:12 CET)

Note: lpmake and the make library are still under development, and they may change in future releases.

lpmake is a Ciao application which uses the Ciao make library to implement a dependency-driven scripts in a similar way to the Un*x make facility.

The original purpose of the Un*x make utility is to determine automatically which pieces of a large program needed to be recompiled, and issue the commands to recompile them. In practice, make is often used for many other purposes: it can be used to describe any task where some files must be updated automatically from others whenever these change. lpmake can be used for the same types of applications as make, and also for some new ones, and, while being simpler, it offers a number of advantages over make. The first one is portability. When compiled to a bytecode executable lpmake runs on any platform where a Ciao engine is available. Also, the fact that typically many of the operations are programmed in Prolog within the makefile, not needing external applications, improves portability further. The second advantage of lpmake is improved programming capabilities. While lpmake is simpler than make, lpmake allows using the Ciao Prolog language within the scripts. This allows establishing more complex dependencies and programming powerful operations within the make file, and without resorting to external packages (e.g., operating system commands), which also helps portability. A final advantage of lpmake is that it supports a form of autodocumentation: comments associated to targets can be included in the configuration files. Calling lpmake in a directory which has such a configuration file explains what commands the configuration file support and what these commands will do.

173.1 General operation

To prepare to use lpmake, and in a similar way to make, you must write a configuration file: a module (typically called Makefile.pl) that describes the relationships among files in your program or application, and states the commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files. Another example is running latex and dvips on a set of source .tex files to generate a document in dvi and postscript formats. Once a suitable makefile exists, each time you change some source files, simply typing lpmake suffices to perform all necessary operations (recompilations, processing text files, etc.). The lpmake program uses the dependency rules in the makefile and the last modification times of the files to decide which of the files need to be updated. For each of those files, it issues the commands recorded in the makefile. For example, in the latex/dvips case one rule states that the .dvi file should be updated from the .tex files whenever one of them changes and another rule states that the .ps file needs to be updated from a .dvi file every time it changes. The rules also describe the commands to be issued to update the files.

So, the general process is as follows: lpmake executes commands in the configuration file to update one or more target names, where name is often a program, but can also be a file to be generated or even a “virtual” target. lpmake updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist. You can provide command line arguments to lpmake to control which files should be regenerated, or how.
173.2 Format of the Configuration File

lpmake uses as default configuration file the file Makefile.pl, if it is present in the current directory. This can be overridden and another file used by means of the -m option. The configuration file must a module that uses the make package. This package provides syntax for defining the dependency rules and functionality for correctly interpreting these rules. The configuration files can contain such rules and also arbitrary Ciao Prolog predicates. The syntax of the rules is described in Chapter 83 [The Ciao Make Package], page 369, together with some examples.

173.3 lpmake usage

Supported command line options:

lpmake [-v] <command1> ... <commandn>

Process commands <command1> ... <commandn>, using file 'Makefile.pl' in the current directory as configuration file. The configuration file must be a module. This is useful to implement inheritance across different configuration files, i.e., the values declared in a configuration file can be easily made to override those defined in another.

The optional argument '-v' produces verbose output, reporting on the processing of the dependency rules. Very useful for debugging Makefiles.

lpmake [-v] [-m <.../Configfile.pl>] <command1> ... <commandn>

Same as above, but using file <.../Configfile.pl> as configuration file.

lpmake -h  [ -m <.../Configfile.pl> ]
lpmake -help  [ -m <.../Configfile.pl> ]

Print this help message. If a configuration file is given, and the commands in it are commented, then information on these commands is also printed.

173.4 Acknowledgments (lpmake)

Some parts of the documentation are taken from the documentation of GNU's gmake.
174 Find out which architecture we are running on

Author(s): Manuel Carro, Robert Manchek.

Version: 0.0#6 (2001/3/26, 13:56:52 CEST)

The architecture and operating system the engine is compiled for determines whether we can use or not certain libraries. This script, taken from a PVM distribution, uses a heuristic (which may need to be tuned from time to time) to find out the platform. It returns a string which is used throughout the engine (in #ifdefs) to enable/disable certain characteristics.

174.1 Usage (ciao_get_arch)

Usage: ciao_get_arch

174.2 More details (ciao_get_arch)

Look at the script itself...
175 Print out WAM code

Author(s): Manuel Carro.

Version: 0.5 (2003/1/20, 17:12:6 CET)

This program prints to standard output a symbolic form of the (modified) Wam code the
compiler generates for a given source file. Directives are ignored.

175.1 Usage (compiler_output)

Print (modified, partial) WAM code for a .pl file

Usage: compiler_output <file.pl>
176 Compile Ciao in an arbitrary subdirectory

**Author(s):** M. Carro.

**Version:** 0.1 (2003/7/24, 15:53:34 CEST)

A simple bash script for compile the whole Ciao distribution, in order to test automatically whether something is broken.

### 176.1 Usage (auto_compile_ciao)

Usage: `auto_compile_ciao ciaodir_name destdir_name`

### 176.2 More details (auto_compile_ciao)

The script copies the ciao distribution passed on as first argument to a subdirectory of the directory name passed as second argument. The subdirectory is always the same, in order to speed up tests made repeateadly in a distribution. It also deletes from the destination files which do not appear any longer in the sources, and tries not to copy files which will eventually be generated or deleted prior to compilation.

After that, the `SETTINGS` file is updated to reflect the new location, the sources are cleaned up, and a regular compilation is started.
177 Gathering all modules in a directory

Author(s): D. Cabeza, M. Carro.
Version: 0.0#1 (2002/1/3, 17:40:19 CET)
A simple bash script for collecting files which are modules, in order to compile automatically the CIAO libraries.

177.1 Usage (collect_modules)

Usage: collect_modules module_name

177.2 More details (collect_modules)

Look at the Makefiles in lib and library
PART XI - Contributed libraries

Author(s): The CLIP Group.

This part includes a number of libraries which have contributed by users of the Ciao system. Over time, some of these libraries are moved to the main library directories of the system.
Chapter 178: Programming MYCIN rules

Author(s): Angel Fernandez Pineda.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#153 (2004/1/7, 13:31:25 CET)

MYCIN databases are declared as Prolog modules containing mycin rules. Those rules are given a certainty factor (CF) which denotates an expert’s credibility on that rule:
- A value of -1 stands for surely not.
- A value of 1 stands for certainly.
- A value of 0 stands for I don’t know.

Intermediate values are allowed.

Mycin rules work on a different way as Prolog clauses: a rule will never fail (in the Prolog sense), it will return a certainty value instead. As a consequence all mycin rules will be explored during inference, so the order in which rules are written is not significant. For this reason, the usage of the Prolog cut (!) is discouraged.

178.1 Usage and interface (mycin)

- Library usage:
  In order to declare a mycin database you must include the following declaration as the first one in your file:
  
  ```prolog
  :- mycin(MycinDataBaseName).
  ```

- New declarations defined:
  export/1.

178.2 Documentation on new declarations (mycin)

export/1:

This directive allows a given mycin predicate to be called from Prolog programs. The way in which mycin rules are called departs from Prolog ones. For instance, the followin mycin predicate:

```prolog
:- export p/1.
```

must be called from Prolog Programs as: mycin(p(X),CF), where CF will be binded to the resulting certainty factor. Obviously, the variables on P/1 may be instantiated as you wish. Since the Prolog predicate mycin/2 may be imported from several mycin databases, it is recommended to fully qualify those predicate goals. For example:

```prolog
mydatabase:mycin(p(X),CF).
```

Usage: :- export(Spec).

- Description: Spec will be a callable mycin predicate.

178.3 Known bugs and planned improvements (mycin)

- Not fully implemented.
- Dynamic mycin predicates not implemented: open question.
- Importation of user-defined mycin predicates requires further design. This includes importation of mycin databases from another mycin database.
Chapter 179: A Chart Library

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#154 (2003/12/4, 17:39:3 CET)

This library is intended to ease the task of displaying some graphical results. This library allows the programmer to visualize different graphs and tables without knowing anything about specific graphical packages.

You need to install the BLT package in your computer. BLT is an extension to the Tk toolkit and it does not require any patching of the Tcl or Tk source files. You can find it in http://www.tcltk.com/blt/index.html

Basiclly, when the user invokes a predicate, the library (internally) creates a bltwish interpreter and passes the information through a socket to display the required widget. The interpreter parses the received commands and executes them.

The predicates exported by this library can be classified in four main groups, according to the types of representation they provide.

- bar charts
- line graphs
- scatter graphs
- tables

To represent graphs, the Cartesian coordinate system is used. I have tried to show simple samples for every library exported predicate in order to indicate how to call them.

179.1 Bar charts

In this section we shall introduce the general issues about the set of barchart predicates. By calling the predicates that pertain to this group a bar chart for plotting two-dimensional data (X-Y coordinates) can be created. A bar chart is a graphic means of comparing numbers by displaying bars of lengths proportional to the y-coordinates they represented. The barchart widget has many configurable options such as title, header text, legend and so on. You can configure the appearance of the bars as well. The bar chart widget has the following components:

Header text
The text displayed at the top of the window. If it is '' no text will be displayed.

Save button
The button placed below the header text. Pops up a dialog box for the user to select a file to save the graphic in PostScript format.¹

Bar chart title
The title of the graph. It is displayed at the top of the bar chart graph. If text is '' no title will be displayed.

X axis title
X axis title. If text is '' no x axis title will be displayed.

Y axis title
Y axis title. If text is '' no y axis title will be displayed.

¹ Limitation: Some printers can have problems if the PostScript file is too complex (i.e. too many points/lines appear in the picture).
X axis  X coordinate axis. The x axis is drawn at the bottom margin of the bar chart graph. The x axis consists of the axis line, ticks and tick labels. Tick labels can be numbers or plain text. If the labels are numbers, they could be displayed at uniform intervals (the numbers are treated as normal text) or depending on its x-coordinate value. You can also set limits (maximum and minimum) for the x axis, but only if the tick labels are numeric.

Y axis  Y coordinate axis. You can set limits (maximum and minimum) for the y axis. The y axis is drawn at the right margin of the bar chart graph. The y axis consists of the axis line, ticks and tick labels. The tick labels are numeric values determined from the data and are drawn at uniform intervals.

Bar chart graph
This is the plotting area, placed in the center of the window and surrounded by the axes, the axis titles and the legend (if any). The range of the axes controls what region of the data is plotted. By default, the minimum and maximum limits are determined from the data, but you can set them (as mentioned before). Data points outside the minimum and maximum value of the axes are not plotted.

Legend
The legend displays the name and symbol of each bar. The legend is placed in the right margin of the Bar chart graph.

Footer text
Text displayed at the lower part of the window. If text is '', no header text will be displayed.

Quit button
Button placed below the footer text. Click it to close the window.

All of them are arranged in a window. However you can, for example, show a bar chart window without legend or header text. Other configuration options will be explained later.

In addition to the window appearance there is another important issue about the bar chart window, namely its behaviour in response to user actions. The association user actions to response is called bindings. The main bindings currently specified are the following:

Default bindings
Those are well known by most users. They are related to the frame displayed around the window. As you know, you can interactively move, resize, close, iconify, deiconify, send to another desktop etc. a window.

Bindings related to bar chart graph and its legend
Clicking the left mouse key over a legend element, the corresponding bar turns out into red. After clicking again, the bar toggles to its original look. In addition, you can do zoom-in by pressing the left mouse key over the bar chart graph and dragging to select an area. To zoom out simply press the right mouse button.

When the pointer passes over the plotting area the cross hairs are drawn. The cross hairs consists of two intersecting lines (one vertical and one horizontal). Besides, if the pointer is over a legend element, its background changes.

Bindings related to buttons
There are two buttons in the main widget. Clicking the mouse on the Save button a "Save as" dialog box is popped up. The user can select a file to save the graph. If the user choose a file that already exists, the dialog box prompts the user for confirmation on whether the existing file should be overwritten or not. Furthermore, you can close the widget by clicking on the Quit button.

When the pointer passes over a button the button color changes.

The predicates that belong to this group are those whose names begin with barchart and genmultibar. If you take a look at the predicate names that pertain to this group, you will notice
that they are not self-explanatory. It would have been better to name the predicates in a way that allows the user to identify the predicate features by its name, but it would bring about very long names (i.e. barchart_WithoutLegend_BarsAtUniformIntervals_RandomBarsColors). For this reason I decided to simply add a number after barchart to name them.

### 179.2 Line graphs

It is frequently the case that several datasets need to be displayed on the same plot. If so, you may wish to distinguish the points in different datasets by joining them by lines of different color, or by plotting with symbols of different types. This set of predicates allows the programmer to represent two-dimensional data (X-Y coordinates). Each dataset contains x and y vectors containing the coordinates of the data. You can configure the appearance of the points and the lines which the points are connected with. The configurable line graph components are:

- **line graph** This is the plotting area, placed in the center of the window and surrounded by the axes, the axes titles and the legend (if any). The range of the axes controls what region of the data is plotted. By default, the minimum and maximum limits are determined from the data, but you can set them. Data points outside the minimum and maximum value of the axes are not plotted. You can specify how connecting line segments joining successive datapoints are drawn by setting the `Smooth` argument. `Smooth` can be either linear, step, natural and quadratic. Furthermore, you can select the appearance of the points and lines.

- **Legend** The legend displays the name and symbol of each line. The legend is placed in the right margin of the graph.

The elements header, footer, quit and save buttons, the titles and the axes are quite similar to those in barchart graphs, except in that the tick labels will be numbers. All of them are arranged in a window by the geometry manager. However you can, as we mentioned in the above paragraphs, show a line graph window without any titles or footer text. Other configuration options will be explained later in this section or in the corresponding modules.

Related to the behaviour of the widgets in response to user actions (bindings) we will remark the following features:

- **Bindings related to line graph and its legend**
  - Clicking the left mouse key over a legend element, the corresponding line turns out into blue. Repeating the action reverts the line to its original color. Moreover, you can do zoom-in by clicking the left mouse key over the bar chart graph and dragging a rectangle defining the area you want to zoom in. To zoom out simply press the right mouse button.
  
  - When the pointer passes over the plotting area the cross hairs are drawn. The cross hairs consists of two intersecting lines (one vertical and one horizontal). Besides, if the pointer is over a legend element, its background changes.

- **Other bindings**
  - The default bindings and the bindings related to the save and quit buttons are similar to those in the bar chart graphs.

The predicates that belong to this group are those whose names begin with `graph_`.

### 179.3 Scatter graphs

The challenge of this section is to introduce some general aspects about the scatter graph predicates group. By invoking the scatter graph predicates the user can represent two-dimensional point datasets. Often you need to display one or several point datasets on the same plot. If so, you may wish to distinguish the points that pertain to different datasets by
using plotting symbols of different types, or by displaying them in different colors. This set of predicates allows you to represent two-dimensional data (X-Y coordinates). Each dataset contains x and y vectors containing the coordinates of the data. You can configure the appearance of the points. The configurable scatter graph components are:

**scatter graph**
This is the plotting area, placed in the center of the window and surrounded by the axes, the axes titles and the legend (if any). The range of the axes controls what region of the data is plotted. By default, the minimum and maximum limits are determined from the data, but you can set them (as we mentioned before). Data points outside the minimum and maximum value of the axes are not plotted. The user can select the appearance of the points.

**Legend**
The legend displays the name and symbol of each point dataset. The legend is drawn in the right margin of the graph.

The elements header, footer, quit and save buttons, the titles and the axes are similar to those in barchart graphs except for that, as in line graphs, the tick labels will be numbers. All of them are arranged in a window by the geometry manager. However you can, for example, show a scatter graph window without titles or footer text, as we mentioned before. Other configuration options will be explained later, in the corresponding modules.

Related to the behaviour of the widgets in response to user actions (bindings) the following features are:

**Bindings related to scatter graph and its legend**
Clicking the left mouse key over a legend element, the points which belong to the corresponding dataset turn out into blue. Repeating the action toggles the point dataset to its original color. Moreover, you can do zoom-in by clicking the left mouse key over the bar chart graph and dragging a rectangle defining the area you want to zoom-in on. To do zoom-out simply press the right mouse button.
When the pointer passes over the plotting area the cross hairs are drawn. The cross hairs consists of two intersecting lines (one vertical and one horizontal). Besides, if the pointer is over a legend element, its background changes.

**Other bindings**
The default bindings and the bindings related to the save and quit buttons are similar to those in the bar chart graphs.

The predicates that belong to this group are those whose names began with `scattergraph_`.

### 179.4 Tables

The purpose of this section is to allow the user to display results in a table. A table is a regular structure in which:
- Every row has the same number of columns, or
- Every column has the same number of rows.

The widget configurable components are as follows:

**Title**
Title of the widget, it is displayed centered at the top of the canvas. If text is "" no title will be displayed.

**Header text**
Left centered text displayed bellow the title. If text is "" no header text will be displayed.
Table

Is placed in the center of the window. The table is composed by cells ordered in rows and columns. The cell values can be either any kind of text or numbers and they can be empty as well (see the type definition in the corresponding chapter module). A table is a list of lists. Each sublist is a row, so every sublist in the table must contain the same number of elements.

Footer text

Left centered text displayed at the lower part of the window. If text is '', no header text will be displayed.

Quit button

Button placed below the footer text. You can click it to close the window.

If the arguments are not in a correct format an exception will be thrown. Moreover, these widgets have the default bindings and the binding related to the quit button:

The set of predicates that belongs to this group are those which names begin with table.

### 179.5 Overview of widgets

Although you don’t have to worry about how to arrange the widgets, here is an overview of how Tcl-tk, the underlying graphical system currently used by chartlib, performs this task. Quoting from the book *Tcl and Tk toolkit*, John K. Ousterhout.

The X Window System provides many facilities for manipulating windows in displays. The root window may have any number of child windows, each of which is called a top-level window. Top-level windows may have children of their own, which may have also children, and so on. The descendants of top-level windows are called internal windows. Internal windows are used for individual controls such as buttons, text entries, and for grouping controls together. An X-application typically manages several top-level windows. Tk uses X to implement a set of controls with the Motif look and feel. These controls are called widgets. Each widget is implemented using one X window, and the terms "window" and "widget" will be used interchangeably in this document. As with windows, widgets are nested in hierarchical structures. In this library top-level widgets (nonleaf or main) are just containers for organizing and arranging the leaf widgets (components). Thereby, the barchart widget is a top-level window which contains some widget components.

Probably the most painstaking aspect of building a graphical application is getting the placement and size of the widgets just right. It usually takes many iterations to align widgets and adjust their spacing. That’s because managing the geometry of widgets is simply not a packing problem, but also graphical design problem. Attributes such as alignment, symmetry, and balance are more important than minimizing the amount of space used for packing. Tk is similar to other X toolkits in that it does not allow widgets to determine their own geometries. A widget will not even appear unless it is managed by a geometry manager. This separation of geometry management from internal widget behaviour allows multiple geometry managers to exist simultaneously and permits any widget to be used with any geometry manager. A geometry manager’s job is to arrange one or more slave widgets relative to a master widgets. There are some geometry managers in Tk such as pack, place and canvas widget. We will use another one call table.

The table geometry manager arranges widgets in a table. It’s easy to align widgets (horizontally and vertically) or to create empty space to balance the arrangement of the widgets. Widgets (called slaves in the Tk parlance) are arranged inside a containing widget (called the master). Widgets are positioned at row, column locations and may span any number of rows or columns. More than one widget can occupy a single location. The placement of widget windows determines both the size and arrangement of the table. The table queries the requested size of
each widget. The requested size of a widget is the natural size of the widget (before the widget
is shrunk or expanded). The height of each row and the width of each column is the largest
widget spanning that row or column. The size of the table is in turn the sum of the row and
column sizes. This is the table’s normal size. The total number of rows and columns in a table
is determined from the indices specified. The table grows dynamically as windows are added at
larger indices.

179.6 Usage and interface (chartlib)

- Library usage:
  :- use_module(library(chartlib)).

- Other modules used:
  - System library modules:
    chartlib/genbar1, chartlib/genbar2, chartlib/genbar3, chartlib/genbar4,
    chartlib/genmultibar, chartlib/table_widget1,
    chartlib/table_widget2, chartlib/table_widget3, chartlib/table_widget4,
    chartlib/gengraph1, chartlib/gengraph2, chartlib/chartlib_errhandle.

179.7 Documentation on exports (chartlib)

barchart1/7: (UNDOC.REEXPORT)
  Imported from genbar1 (see the corresponding documentation for details).

barchart1/9: (UNDOC.REEXPORT)
  Imported from genbar1 (see the corresponding documentation for details).

percentbarchart1/7: (UNDOC.REEXPORT)
  Imported from genbar1 (see the corresponding documentation for details).

barchart2/7: (UNDOC.REEXPORT)
  Imported from genbar2 (see the corresponding documentation for details).

barchart2/11: (UNDOC.REEXPORT)
  Imported from genbar2 (see the corresponding documentation for details).

percentbarchart2/7: (UNDOC.REEXPORT)
  Imported from genbar2 (see the corresponding documentation for details).
barchart3/7: (UNDONE_REEXPORT)
Imported from genbar3 (see the corresponding documentation for details).

barchart3/9: (UNDONE_REEXPORT)
Imported from genbar3 (see the corresponding documentation for details).

percentbarchart3/7: (UNDONE_REEXPORT)
Imported from genbar3 (see the corresponding documentation for details).

barchart4/7: (UNDONE_REEXPORT)
Imported from genbar4 (see the corresponding documentation for details).

barchart4/11: (UNDONE_REEXPORT)
Imported from genbar4 (see the corresponding documentation for details).

percentbarchart4/7: (UNDONE_REEXPORT)
Imported from genbar4 (see the corresponding documentation for details).

multibarchart/8: (UNDONE_REEXPORT)
Imported from genmultibar (see the corresponding documentation for details).

multibarchart/10: (UNDONE_REEXPORT)
Imported from genmultibar (see the corresponding documentation for details).

tablewidget1/4: (UNDONE_REEXPORT)
Imported from table_widget1 (see the corresponding documentation for details).

tablewidget1/5: (UNDONE_REEXPORT)
Imported from table_widget1 (see the corresponding documentation for details).

tablewidget2/4: (UNDONE_REEXPORT)
Imported from table_widget2 (see the corresponding documentation for details).

tablewidget2/5: (UNDONE_REEXPORT)
Imported from table_widget2 (see the corresponding documentation for details).
tablewidget3/4: (UNDOC_REEXPORT) Imported from table_widget3 (see the corresponding documentation for details).

tablewidget3/5: (UNDOC_REEXPORT) Imported from table_widget3 (see the corresponding documentation for details).

tablewidget4/4: (UNDOC_REEXPORT) Imported from table_widget4 (see the corresponding documentation for details).

imported from table_widget4 (see the corresponding documentation for details).

tablewidget4/5: (UNDOC_REEXPORT) Imported from table_widget4 (see the corresponding documentation for details).

graph_b1/9: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

graph_b1/13: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

graph_w1/9: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

graph_w1/13: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

scattergraph_b1/8: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

scattergraph_b1/12: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

scattergraph_w1/8: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).

scattergraph_w1/12: (UNDOC_REEXPORT) Imported from gengraph1 (see the corresponding documentation for details).
graph_b2/9: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

graph_b2/13: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

graph_w2/9: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

graph_w2/13: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

scattergraph_b2/8: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

scattergraph_b2/12: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

scattergraph_w2/8: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

scattergraph_w2/12: (UNDOC_REEXPORT)
    Imported from gengraph2 (see the corresponding documentation for details).

chartlib_text_error_protect/1: (UNDOC_REEXPORT)
    Imported from chartlib_errhandle (see the corresponding documentation for details).

chartlib_visual_error_protect/1: (UNDOC_REEXPORT)
    Imported from chartlib_errhandle (see the corresponding documentation for details).
Chapter 180: Constraint programming over finite domains

Author(s): J.M. Gomez, M. Carro.

This package allows to write and evaluate constraint programming expressions over finite domains in a Ciao program.

(Other examples can be found in the source and library directories.)

- SEND + MORE = MONEY:

```prolog
:- use_package(fd).
:- use_module(library(prolog_sys), [statistics/2]).
:- use_module(library(format)).

smm(SMM) :-
    statistics(runtime, _),
    do_smm(SMM),
    statistics(runtime, [_, Time]),
    format("Used ~d milliseconds\n", Time).

do_smm(X) :-
    X = [S,E,N,D,M,O,R,Y],
    X in 0 .. 9,
    all_different(X),
    M > 0,
    S > 0,
    1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E =. 10000*M + 1000*O + 100*N + 10*E + Y,
    labeling(X).
```

- Queens:

```prolog
:- use_package(fd).
:- use_module(library(prolog_sys), [statistics/2]).
:- use_module(library(format)).
:- use_module(library(aggregates)).
:- use_module(library(lists), [length/2]).

queens(N, Qs) :-
    statistics(runtime, _),
    do_queens(N, Qs),
    statistics(runtime, [_, Time]),
    format("Used ~d milliseconds\n", Time).

do_queens(N, Qs) :-
    constrain_values(N, N, Qs),
    all_different(Qs), !,
    labeling(Qs).

constrain_values(0, _N, []).
constrain_values(N, Range, [X|Xs]) :-
    N > 0,
    X in 1 .. Range,
```
N1 is N - 1,
constrain_values(N1, Range, Xs),
no_attack(Xs, X, 1).

no_attack([], _Queen, _Nb).
no_attack([Y|Ys], Queen, Nb):-
    Nb1 is Nb + 1,
    no_attack(Ys, Queen, Nb1),
    Queen .<> Y + Nb,
    Queen .<> Y - Nb.

180.1 Usage and interface (fd)

- Library usage:
  :- use_package(fd).
  or
  :- module(...,...,[fd]).

- Exports:
  - Predicates:
    labeling/1, pitm/2, choose_var/3, choose_free_var/2, choose_var_nd/2,
    choose_value/2, retrieve_range/2, retrieve_store/2, glb/2, lub/2, bounds/3,
    retrieve_list_of_values/2.
  - Regular Types:
    fd_item/1, fd_range/1, fd_subrange/1, fd_store/1, fd_store_entity/1.

- New operators defined:
  =/.2 [700,xfx], .<>.2 [700,xfx], .<.2 [700,xfx], .=/.2 [700,xfx], .>.2 [700,xfx], .>=/.2 [700,xfx], ../.2 [500,xfx], .&/.2 [600,xfy], in/2 [700,xfy].

180.2 Documentation on exports (fd)

fd_item/1: REGTYPE
Usage: fd_item(FD_item)
   Description: FD_item is a finite domain entity, i.e. either a finite domains variable or an integer.

fd_range/1: REGTYPE
Usage: fd_range(FD_range)
   Description: FD_range is the range of a finite domain entity.

fd_subrange/1: REGTYPE
Usage:
   Description: A subrange is a pair representing a single interval.
fd_store/1:
  Usage: fd_store(FD_store)
  - Description: FD_store is a representation of the constraint store of a finite domain entity.

fd_store_entity/1:
  Usage:
  - Description: Representation of primitive constraints.

labeling/1:
  Usage: labeling(Vars)
  - Description: Implements the labeling process. Assigns values to the input variables Vars. On exit all variables are instantiated to a consistent value. On backtracking, the predicate returns all possible assignments. No labeling heuristics implemented so far, i.e. variables are instantiated in their order of appearance.
  - The following properties should hold at call time:
    Vars is a list of fd_items.

pitm/2:
  Usage: pitm(V, MiddlePoint)
  - Description: Returns in MiddlePoint the intermediate value of the range of V. In case V is a ground integer value the returned value is V itself.
  - The following properties should hold at call time:
    V is currently a term which is not a free variable. (term_typing:nonvar/1)
    MiddlePoint is a free variable. (term_typing:var/1)
    V is a finite domain entity, i.e. either a finite domains variable or an integer. (user(... /fd_doc):fd_item/1)
    MiddlePoint is an integer. (basic_props:int/1)

choose_var/3:
  Usage: choose_var(ListOfVars, Var, RestOfVars)
  - Description: Returns a finite domain item Var from a list of fd items ListOfVars and the rest of the list RestOfVars in a deterministic way. Currently it always returns the first item of the list.
  - The following properties should hold at call time:
    ListOfVars is currently a term which is not a free variable. (term_typing:nonvar/1)
    Var is a free variable. (term_typing:var/1)
    RestOfVars is a free variable. (term_typing:var/1)
    ListOfVars is a list of fd_items. (basic_props:list/2)
    Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(... /fd_doc):fd_item/1)
    RestOfVars is a list of fd_items. (basic_props:list/2)
choose_free_var/2:
Usage: choose_free_var(ListOfVars, Var)
- Description: Returns a free variable Var from a list of fd items ListOfVars. Currently it always returns the first free variable of the list.
- The following properties should hold at call time:
  ListOfVars is currently a term which is not a free variable. (term_typing:nonvar/1)
  Var is a free variable. (term_typing:var/1)
  ListOfVars is a list of fd_items. (basic_props:list/2)
  Var is a free variable. (term_typing:var/1)

choose_var_nd/2:
Usage: choose_var_nd(ListOfVars, Var)
- Description: Returns non deterministically an fd item Var from a list of fd items ListOfVars.
- The following properties should hold at call time:
  ListOfVars is currently a term which is not a free variable. (term_typing:nonvar/1)
  Var is a free variable. (term_typing:var/1)
  ListOfVars is a list of fd_items. (basic_props:list/2)
  Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(... /fd_doc):fd_item/1)

choose_value/2:
Usage: choose_value(Var, Value)
- Description: Produces an integer value Value from the domain of Var. On backtracking returns all possible values for Var.
- The following properties should hold at call time:
  Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  Value is a free variable. (term_typing:var/1)
  Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(... /fd_doc):fd_item/1)
  Value is an integer. (basic_props:int/1)

retrieve_range/2:
Usage: retrieve_range(Var, Range)
- Description: Returns in Range the range of an fd item Var.
- The following properties should hold at call time:
  Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  Range is a free variable. (term_typing:var/1)
  Var is a free variable. (term_typing:var/1)
  Range is the range of a finite domain entity. (user(... /fd_doc):fd_range/1)
Chapter 180: Constraint programming over finite domains

**retrieve_store/2:**

**Usage:** retrieve_store(Var, Store)

- **Description:** Returns in Store a representation of the constraint store of an fd item Var.

- **The following properties should hold at call time:**
  - Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Store is a free variable. (term_typing:var/1)
  - Var is a free variable.
  - Store is a representation of the constraint store of a finite domain entity. (user(.../fd_doc):fd_store/1)

**glb/2:**

**Usage:** glb(Var, LowerBound)

- **Description:** Returns in LowerBound the lower bound of the range of Var.

- **The following properties should hold at call time:**
  - Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  - LowerBound is a free variable. (term_typing:var/1)
  - Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(.../fd_doc):fd_item/1)
  - LowerBound is an integer. (basic_props:int/1)

**lub/2:**

**Usage:** lub(Var, UpperBound)

- **Description:** Returns in UpperBound the upper bound of the range of Var.

- **The following properties should hold at call time:**
  - Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  - UpperBound is a free variable. (term_typing:var/1)
  - Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(.../fd_doc):fd_item/1)
  - UpperBound is an integer. (basic_props:int/1)

**bounds/3:**

**Usage:** bounds(Var, LowerBound, UpperBound)

- **Description:** Returns in LowerBound and UpperBound the lower and upper bounds of the range of Var.

- **The following properties should hold at call time:**
  - Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  - LowerBound is a free variable. (term_typing:var/1)
  - UpperBound is a free variable. (term_typing:var/1)
  - Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(.../fd_doc):fd_item/1)
  - LowerBound is an integer. (basic_props:int/1)
  - UpperBound is an integer. (basic_props:int/1)
retrieve_list_of_values/2:

Usage: retrieve_list_of_values(Var, ListOfValues)

- Description: Returns in ListOfValues an enumeration of all the values in the range of Var.

- The following properties should hold at call time:
  - Var is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ListOfValues is a free variable. (term_typing:var/1)
  - Var is a finite domain entity, i.e. either a finite domains variable or an integer. (user(.../fd_doc):fd_item/1)
  - ListOfValues is a list of ints. (basic_props:list/2)
Chapter 181: XDR handle library

Author(s): Jos Manuel Gmez Prez.

This library offers facilities to enable users to setup preferences on the values an eventual XML document may take. XML documents are specified by XDR documents (eXternal Data Representation standard), in a way conceptually similar to that of objects and classes in object oriented programming. These facilities allow to take as input an XDR Schema defining the class of documents of interest, and establish a dialogue with the user via an HTML form that allows the user to setup preferences to select sub-classes of documents (those which satisfy the preferences). The preferences are the output of the process and may be in the form of XPath expressions, for example, as can be seen in the example attached in the "examples" directory.

181.1 Usage and interface (xdr_handle)

- Library usage:
  :- use_module(library(xdr_handle)).
- Exports:
  - Predicates:
    xdr_tree/3, xdr_tree/1, xdr2html/4, xdr2html/2, unfold_tree/2, unfold_tree_dic/3, xdr_xpath/2.
  - Regular Types:
    xdr_node/1.
- Other modules used:
  - System library modules:
    pillow/http, pillow/html, pillow/pillow_types, xdr_handle/xdr_types, aggregates, lists, terms.

181.2 Documentation on exports (xdr_handle)

xdr_tree/3: PREDICATE

Usage: xdr_tree(XDR_url, XDR_tree, XDR_id)

- Description: Parses an XDR (External Data Representation Standard) located at an url XDR_url into a tree structured Prolog term XDR_tree. It also returns an identifier of the XDR_tree XDR_id corresponding to the sequence of nodes in the tree (this is intended to be a hook to use in CGI applications).
- The following properties should hold at call time:
  XDR_url is currently a term which is not a free variable. (term_typing:nonvar/1)
  XDR_tree is a free variable. (term_typing:var/1)
  XDR_id is a free variable. (term_typing:var/1)
  XDR_url specifies a URL. (pillow_types:url_term/1)
  XDR_tree specifies an XDR document. (xdr_types:xdr/1)
  XDR_id is an integer. (basic_props:int/1)
xdr_tree/1:
Usage: xdr_tree(XDR_tree)
- Description: Checks the correctness of an XDR tree XDR_tree.
- The following properties should hold at call time:
  XDR_tree specifies an XDR document. (xdr_types:xdr/1)

xdr_node/1:
Usage: xdr_node(XDR_node)
- Description: XDR_node is a XDR tree node.

xdr2html/4:
Usage: xdr2html(XDRTree, HTMLOutput, UnfoldedTree, Dic)
- Description: Receives an XDR tree XDRTree and produces the corresponding HTML code HTMLOutput, an equivalent unfolded plain tree UnfoldedTree and a control dictionary Dic to hold a reference to eventual form objects.
- The following properties should hold at call time:
  XDRTree is currently a term which is not a free variable. (term_typing:nonvar/1)
  HTMLOutput is a free variable. (term_typing:var/1)
  UnfoldedTree is a free variable. (term_typing:var/1)
  Dic is a free variable. (term_typing:var/1)
  XDRTree specifies an XDR document. (xdr_types:xdr/1)
  HTMLOutput is a term representing HTML code. (pillow_types:html_term/1)
  UnfoldedTree specifies an XDR document. (xdr_types:xdr/1)
  Dic is a dictionary of values of the attributes of a form. It is a list of form_assignment (pillow_types:form_dict/1)

xdr2html/2:
Usage: xdr2html(XDRTree, HTMLOutput)
- Description: Receives an XDR tree XDRTree and produces the corresponding HTML code HTMLOutput. This HTML code is intended to be part of a form used as a means by which an eventual user can give value to an instance of the XDR, i.e. an XML element.
- The following properties should hold at call time:
  XDRTree is currently a term which is not a free variable. (term_typing:nonvar/1)
  HTMLOutput is a free variable. (term_typing:var/1)
  XDRTree specifies an XDR document. (xdr_types:xdr/1)
  HTMLOutput is a term representing HTML code. (pillow_types:html_term/1)

unfold_tree/2:
Usage: unfold_tree(XDRTree, UFT)
Description: Obtains an unfolded XDR tree UFT from a standard XDR tree XDRTree, i.e. an XDR tree where all references to XDR elements have been substituted with the elements themselves. Especially useful for eventual generation of equivalent XPATH expressions, (see example).

The following properties should hold at call time:
- XDRTree is currently a term which is not a free variable. \(\text{term\_typing:nonvar/1}\)
- UFT is a free variable. \(\text{term\_typing:var/1}\)
- XDRTree specifies an XDR document. \(\text{xdr\_types:xdr/1}\)
- UFT specifies an XDR document. \(\text{xdr\_types:xdr/1}\)

\[\text{unfold\_tree\_dic/3:}\]

**Usage:** unfold_tree_dic(XDRTree, UFT, Dic)

Description: Obtains an unfolded XDR tree UFT and a form dictionary Dic from a standard XDR tree XDRTree. Especially useful for HTML form data exchange (see example).

The following properties should hold at call time:
- XDRTree is currently a term which is not a free variable. \(\text{term\_typing:nonvar/1}\)
- UFT is a free variable. \(\text{term\_typing:var/1}\)
- Dic is a free variable. \(\text{term\_typing:var/1}\)
- XDRTree specifies an XDR document. \(\text{xdr\_types:xdr/1}\)
- UFT specifies an XDR document. \(\text{xdr\_types:xdr/1}\)
- Dic is a dictionary of values of the attributes of a form. It is a list of form_assignment \(\text{pillow\_types:form\_dict/1}\)

\[\text{xdr\_xpath/2:}\]

**Usage:** xdr_xpath(XDRTree, XPath)

Description: Produces an XPATH expression XPath from an XDR tree XDRTree. If the given XDR tree has no definite value the xpath expression produced will be empty.

The following properties should hold at call time:
- XDRTree is currently a term which is not a free variable. \(\text{term\_typing:nonvar/1}\)
- XPath is a free variable. \(\text{term\_typing:var/1}\)
- XDRTree specifies an XDR document. \(\text{xdr\_types:xdr/1}\)
- XPath is an atom. \(\text{basic\_props:atm/1}\)
182 XML query library

Author(s): Jos Manuel Gmez Prez.
Version: 0.1 (2003/12/1, 13:24:9 CET)

This package provides a language suitable for querying XML documents from a Prolog program. Constraint programming expressions can be included in order to prune search as soon as possible, i.e. upon constraint unsatisfiability, improving efficiency. Also, facilities are offered to improve search speed by transforming XML documents into Prolog programs, hence reducing search to just running the program and taking advantage of Prolog's indexing capabilities.

Queries in an XML document have a recursive tree structure that permits to detail the search on the XML element sought, its attributes, and its children. As a suffix, a constraint programming expression can be added. Queries return value for the free variables included (in case of success), and checks whether the XML document structure matches that depicted by the query itself.

The operators introduced are described below:

- @ Delimits a subquery on an element's attribute, such as `product@val(product_name, "car")`, the first argument being the attribute name and the second its value. Any of them can be free variables, being possible to write queries like `product@val(Name, "car")`, intended to find the 'Name' of attributes of element product whose value is the string "car".
- :: The right-hand side of the subexpression delimited by this operator is a query on the children elements of the element described on its left-hand side.
- with Declares the constraints the items sought must satisfy.

Some examples of this query language (more can be found in the examples directory):

- Example A:
  ```prolog
  product@val(product_name,"car")::(quantity(X),
  "time-left"(Y),
  negotiation::preference::price(Z))
  with X * Z >. Y
  ```

- Example B:
  ```prolog
  nitf::head::docdata::'doc-id'@val('id-string','020918050')::(Y),
  body::'body.head'::abstract::p(X)
  ```
182.1 Usage and interface (xml_path)

- **Library usage:**
  `- use_package(xml_path).
  or
  `- module(...,...,[xml_path]).

- **Exports:**
  - **Predicates:**
    - `xml_search/3, xml_parse/3, xml_parse_match/3, xml_search_match/3, xml_index_query/3, xml_index_to_file/2, xml_index/1, xml_query/3.
  - **New operators defined:**
    - `/2 [200,yfx], ::/2 [300,xfy], with/2 [800,yfx].
  - **Other modules used:**
    - **System library modules:**
      - `xml_path/xml_path_types`.

182.2 Documentation on exports (xml_path)

**xml_search/3:**

**Usage:** `xml_search(Query, Source, Doc)`

- **Description:** Checks a high level query `Query` against an XML document `Source`. If the query is successful it returns in `Doc` the whole xml element(s) of the document that matched it.

- **The following properties should hold at call time:**
  - `Query` is currently a term which is not a free variable.  
    Input: `term_typing:nonvar/1`
  - `Source` is currently a term which is not a free variable.  
    Input: `term_typing:nonvar/1`
  - `Doc` is a free variable.  
    Input: `term_typing:var/1`
  - `Query` is a primitive XML query.  
    Input: `xml_path_types:canonic_xml_query/1`
  - `Source` is either a XML attribute, a XML element or a line break.  
    Input: `xml_path_types:canonic_xml_item/1`
  - `Doc` is either a XML attribute, a XML element or a line break.  
    Input: `xml_path_types:canonic_xml_item/1`

**xml_parse/3:**

**Usage:** `xml_parse(Query, Source, Doc)`

- **Description:** Checks a high level query `Query` against an XML document `Source`. If the query is successful it returns in `Doc` the whole xml element(s) of the document that matched it. On the contrary as `xml_search/3`, the query can start at any level of the XML document, not necessarily at the root node.

- **The following properties should hold at call time:**
  - `Query` is currently a term which is not a free variable.  
    Input: `term_typing:nonvar/1`
  - `Source` is currently a term which is not a free variable.  
    Input: `term_typing:nonvar/1`
xml_parse_match/3:

**Usage:** xml_parse_match(Query, Source, Match)

- **Description:** Checks a high level query Query against an XML document Source. If the query is successful it returns in Doc the exact subtree of the XML document that matched it. On the contrary as `$xml_search_match/3`, the query can start at any level of the XML document, not necessarily at the root node.

- **The following properties should hold at call time:**
  - Query is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Source is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Match is a free variable. (term_typing:var/1)
  - Query is a primitive XML query. (xml_path_types:canonic_xml_query/1)
  - Source is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)
  - Match is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_search_match/3:

**Usage:** xml_search_match(BasicQuery, SourceDoc, Match)

- **Description:** Checks query BasicQuery against an XML document SourceDoc. If the query is successful it returns in Doc the exact subtree of the XML document that matched it.

- **The following properties should hold at call time:**
  - BasicQuery is currently a term which is not a free variable. (term_typing:nonvar/1)
  - SourceDoc is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Match is a free variable. (term_typing:var/1)
  - BasicQuery is a primitive XML query. (xml_path_types:canonic_xml_query/1)
  - SourceDoc is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)
  - Match is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_index_query/3:

**Usage:** xml_index_query(Query, Id, Match)

- **Description:** Matches a high level query Query against an XML document previously transformed into a Prolog program. Id identifies the resulting document Match, which is the exact match of the query against the XML document.
The following properties should hold at call time:

Query is currently a term which is not a free variable. (term_typing:nonvar/1)
Id is a free variable. (term_typing:var/1)
Match is a free variable. (term_typing:var/1)
Query is a primitive XML query. (xml_path_types:canonic_xml_query/1)
Id is an atom. (basic_props:atm/1)
Match is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_index_to_file/2:
Usage: xml_index_to_file(SourceDoc, File)
Description: Transforms the XML document SourceDoc in a Prolog program which is output to file File.
The following properties should hold at call time:
SourceDoc is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)
File is an atom. (basic_props:atm/1)

xml_index/1:
Usage: xml_index(SourceDoc)
Description: Transforms the XML document SourceDoc in a Prolog program, generating the associated clauses, which are stored dynamically into the current process memory space.
The following properties should hold at call time:
SourceDoc is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_query/3:
Usage: xml_query(Query, Doc, Match)
Description: Checks that XML document Doc is compliant with respect to the query Query expressed in the low level query language. The exact mapping of the query over the document is returned in Match.
The following properties should hold at call time:
Query is currently a term which is not a free variable. (term_typing:nonvar/1)
Doc is currently a term which is not a free variable. (term_typing:nonvar/1)
Match is a free variable. (term_typing:var/1)
Query is a primitive XML query. (xml_path_types:canonic_xml_query/1)
Doc is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)
Match is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)
182.3 Documentation on internals (xml_path)

**canonic_xml_term/1:**
Usage: canonic_xml_term(XMLTerm)
- Description: XMLTerm is a term representing XML code in canonical form.

**canonic_xml_item/1:**
Usage: canonic_xml_item(XMLItem)
- Description: XMLItem is either a XML attribute, a XML element or a line break.

**tag_attrib/1:**
Usage: tag_attrib(Att)
- Description: Att is a XML attribute.

**canonic_xml_query/1:**
Usage: canonic_xml_query(Query)
- Description: Query is a primitive XML query.

**canonic_xml_subquery/1:**
Usage: canonic_xml_subquery(SQuery)
- Description: SQuery defines a XML subquery.
Chapter 183: Low level Interface between Prolog and blt

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#153 (2003/12/4, 17:38:53 CET)

This module exports some predicates to interact with Tcl-tk, particularly with the bltwish program. Bltwish is a windowing shell consisting of the Tcl command language, the Tk toolkit plus the additional commands that comes with the BLT library and a main program that reads commands. It creates a main window and then processes Tcl commands.

183.1 Usage and interface (bltclass)

- Library usage:
  :- use_module(library(bltclass)).

- Exports:
  - Predicates:
    new_interp/1, tcltk_raw_code/2, interp_file/2.
  - Regular Types:
    bltwish_interp/1.

- Other modules used:
  - System library modules:
    sockets/sockets, system, write, read, strings, format, terms.

183.2 Documentation on exports (bltclass)

new_interp/1: PREDICATE
new_interp(Interp)

Creates a bltwish interpreter and returns a socket. The socket allows the communication between Prolog and Tcl-tk. Thus, bltwish receives the commands through the socket.

Usage:
- The following properties should hold at call time:
  Interp is a free variable. (term_typing:var/1)

- The following properties should hold upon exit:
  bltclass:bltwish_interp(Interp) (bltclass:bltwish_interp/1)

tcltk_raw_code/2: PREDICATE
tcltk_raw_code(Command_Line, Interp)

Sends a command line to the interpreter. Tcl-tk parses and executes it.

Usage:
- The following properties should hold at call time:
  Command_Line is a string (a list of character codes). (basic_props:string/1)
  bltclass:bltwish_interp(Interp) (bltclass:bltwish_interp/1)
bltwish_interp/1:  

bltwish_interp(Interp)

This type defines a bltwish interpreter. In fact, the bltwish interpreter receives the commands through the socket created.

    bltwish_interp(Interp) :-
        stream(Interp).

interp_file/2:  

interp_file(File, Interp)

Sends the script file (File) to the interpreter through the socket. A script file is a file that contains commands that Tcl-tk can execute.

Usage:

-- The following properties should hold at call time:

File is a source name.                      (streams_basic:sourcename/1)
bltclass:bltwish_interp(Interp)             (bltclass:bltwish_interp/1)
Chapter 184: Error Handler for Chartlib

Author(s): Isabel Martín García.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#155 (2003/12/4, 17:39:9 CET)

This module is an error handler. If the format of the arguments is not correct in a call to a chartlib predicate an exception will be thrown. You can wrap the chartlib predicates with the predicates exported by this module to handle automatically the errors if any.

184.1 Usage and interface (chartlib_errhandle)

- Library usage:
  :- use_module(library(chartlib_errhandle)).
- Exports:
  - Predicates:
    chartlib_text_error_protect/1, chartlib_visual_error_protect/1.
- Other modules used:
  - System library modules:
    chartlib/bltclass, chartlib/install_utils.

184.2 Documentation on exports (chartlib_errhandle)

chartlib_text_error_protect/1: PREDICATE
  chartlib_text_error_protect(G)
  This predicate catches the thrown exception and sends it to the appropriate handler. The handler will show the error message in the standard output.
  Meta-predicate with arguments: chartlib_text_error_protect(goal).
  Usage:
  - The following properties should hold at call time:
    G is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

chartlib_visual_error_protect/1: PREDICATE
  chartlib_visual_error_protect(G)
  This predicate catches the thrown exception and sends it to the appropriate handler. The handler will pop up a message box.
  Meta-predicate with arguments: chartlib_visual_error_protect(goal).
  Usage:
  - The following properties should hold at call time:
    G is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
184.3 Documentation on internals (chartlib_errhandle)

**handler_type/1:**

```prolog
handler_type(X)
```

The library chartlib includes two error handlers already programmed.

```prolog
handler_type(text).
handler_type(visual).
```

**error_message/2:**

```prolog
error_message(ErrorCode, ErrorMessage)
```

Binds the error code with its corresponding text message.

**Usage:**

* The following properties should hold at call time:

  - ErrorCode is an atom. (basic_props:atm/1)
  - ErrorMessage is an atom. (basic_props:atm/1)

**error_file/2:**

```prolog
error_file(ErrorCode, ErrorFile)
```

Binds the error code with its corresponding script error file.

**Usage:**

* The following properties should hold at call time:

  - ErrorCode is an atom. (basic_props:atm/1)
  - ErrorFile is an atom. (basic_props:atm/1)
185 Color and Pattern Library

Author(s): Isabel Martín García.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#156 (2003/12/4, 17:39:13 CET)

This module contains predicates to access and check conformance to the available colors and patterns.

185.1 Usage and interface (color_pattern)

- Library usage:
  :- use_module(library(color_pattern)).
- Exports:
  - Predicates:
    color/2, pattern/2, random_color/1, random_lightcolor/1, random_darkcolor/1, random_pattern/1.
  - Regular Types:
    color/1, pattern/1.
- Other modules used:
  - System library modules:
    lists, random/random.

185.2 Documentation on exports (color_pattern)

color/1: REGTYPE
  color(Color)
    color('GreenYellow').
    color('Yellow').
    color('White').
    color('Wheat').
    color('BlueViolet').
    color('Violet').
    color('MediumTurquoise').
    color('DarkTurquoise').
    color('Turquoise').
    color('Thistle').
    color('Tan').
    color('Sienna').
    color('Salmon').
    color('VioletRed').
    color('OrangeRed').
    color('MediumVioletRed').
    color('IndianRed').
    color('Red').
    color('Plum').
color('Pink').
color('MediumOrchid').
color('DarkOrchid').
color('Orchid').
color('Maroon').
color('Magenta').
color('Orange').
color('Khaki').
color('LightGray').
color('DimGray').
color('DarkSlateGray').
color('YellowGreen').
color('SpringGreen').
color('SeaGreen').
color('PaleGreen').
color('MediumSpringGreen').
color('MediumSeaGreen').
color('LimeGreen').
color('ForestGreen').
color('DarkOliveGreen').
color('Gold').
color('Brown').
color('Firebrick').
color('Cyan').
color('Coral').
color('SteelBlue').
color('SlateBlue').
color('Goldenrod').
color('SkyBlue').
color('MidnightBlue').
color('MidnightBlue').
color('MediumSlateBlue').
color('MediumBlue').
color('LightSteelBlue').
color('LightBlue').
color('DarkSlateBlue').
color('CornflowerBlue').
color('CadetBlue').
color('Blue').
color('Black').
color('MediumAquamarine').
color('Aquamarine').

Defines available colors for elements such as points, lines or bars.

**color/2:**

**Usage:** color(C1, C2)

**Predicate:**
- **Description:** Test whether the color $C_1$ is a valid color or not. If $C_1$ is a variable the predicate will choose a valid color randomly. If $C_1$ is a ground term that is not a valid color an exception (error9) will be thrown.

- **The following properties should hold at call time:**
  \[
  \text{color\_pattern:color}(C_1) \quad (\text{color\_pattern:color/1})
  \]

- **The following properties should hold upon exit:**
  \[
  \text{color\_pattern:color}(C_2) \quad (\text{color\_pattern:color/1})
  \]

**pattern/1:**

**REGTYPE**

\[
\text{pattern}(\text{Pattern})
\]

- pattern(pattern1).
- pattern(pattern2).
- pattern(pattern3).
- pattern(pattern4).
- pattern(pattern5).
- pattern(pattern6).
- pattern(pattern7).
- pattern(pattern8).
- pattern(pattern9).

Defines valid patterns used in the stipple style bar attribute.

**pattern/2:**

**PREDICATE**

**Usage:** \text{pattern}(P_1, P_2)

- **Description:** Test whether the pattern $P_1$ is a valid pattern or not. If $P_1$ is a variable the predicate will choose a valid pattern randomly. If $P_1$ is a ground term that is not a valid pattern an exception (error10) will be thrown.

- **The following properties should hold at call time:**
  \[
  \text{color\_pattern:pattern}(P_1) \quad (\text{color\_pattern:pattern/1})
  \]

- **The following properties should hold upon exit:**
  \[
  \text{color\_pattern:pattern}(P_2) \quad (\text{color\_pattern:pattern/1})
  \]

**random\_color/1:**

**PREDICATE**

**random\_color(\text{Color})**

This predicate choose a valid color among the availables randomly.

**Usage:**

- **The following properties should hold at call time:**
  \[
  \text{color\_pattern:color}(\text{Color}) \quad (\text{color\_pattern:color/1})
  \]

**random\_lightcolor/1:**

**PREDICATE**

**random\_lightcolor(\text{Color})**

This predicate choose a valid light color among the availables randomly.

**Usage:**
– The following properties should hold at call time:
  Color is a free variable. (term_typing:var/1)

– The following properties should hold upon exit:
  color_pattern:color(Color) (color_pattern:color/1)

random_darkcolor/1:
random_darkcolor(Color)
This predicate choose a valid dark color among the availables randomly.
Usage:
– The following properties should hold at call time:
  Color is a free variable. (term_typing:var/1)

– The following properties should hold upon exit:
  color_pattern:color(Color) (color_pattern:color/1)

random_pattern/1:
random_pattern(Pattern)
This predicate choose a valid pattern among the availables randomly.
Usage:
– The following properties should hold at call time:
  Pattern is a free variable. (term_typing:var/1)

– The following properties should hold upon exit:
  color_pattern:pattern(Pattern) (color_pattern:pattern/1)
186 Barchart widgets - 1

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#137 (2003/12/31, 0:14:2 CET)

This module defines predicates to show barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window. They all share the following features:

- No numeric values for the x axis are needed because they will be interpreted as labels. See xbarelement1/1 definition type.
- The bars will be displayed at uniform intervals.
- The user can either select the appearance of the bars (background color, foreground color and stipple style) or not. See the xbarelement1 type definition. Thus, the user can call each predicate in two ways.
- The bar chart has a legend. One entry (symbol and label) per bar.
- If you don’t want to display text in the elements header, barchart title, x axis title, y axis title or footer, simply type ’’ as the value of the argument.
- The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception error2 will be thrown. If the vectors contains elements but are not correct, the exception error1 or error3 will be thrown, depending on the error type. error1 means that XVector and YVector do not contain the same number of elements and error3 indicates that not all the XVector elements contain a correct number of attributes.

Particular features will be pointed out in the corresponding predicate.

186.1 Usage and interface (genbar1)

- Library usage:
  :- use_module(library(genbar1)).
- Exports:
  - Predicates:
    barchart1/7, barchart1/9, percentbarchart1/7.
  - Regular Types:
    yelement/1, axis_limit/1, header/1, title/1, footer/1.
- Other modules used:
  - System library modules:
    chartlib/bltclass, chartlib/test_format, chartlib/color_pattern,
    chartlib/install_utils, lists, random/random.

186.2 Documentation on exports (genbar1)

barchart1/7:  

barchart1(Header, BarchartTitle, XTitle, XVector, YTitle, YVector, Footer)
The y axis range is determined from the limits of the data. Two examples are given to demonstrate clearly how to call the predicates. In the first example the user sets the bar appearance, in the second one the appearance features will be chosen by the system and the colors that have been assigned to the variables Color1, Color2 and Pattern will be shown also.

Example 1:

```prolog
barchart1('This is the header text',
'Barchart title',
'xaxistitle',
[['bar1','legend_element1','Blue','Yellow','pattern1'],
 ['bar2','legend_element2','Plum','SeaGreen','pattern2'],
 ['bar3','legend_element3','Turquoise','Yellow','pattern5']],
'yaxixtitle',
[20,10,59],
'footer').
```

Example 2:

```prolog
barchart1('This is the header text',
'Barchart title',
'xaxistitle',
[['element1','legend_element1',Color1,Color2,Pattern],
 ['element2','legend_element2'],
 ['element3','legend_element3']],
'yaxixtitle',
[20,10,59],
'footer').
```

Usage:

- The following properties should hold at call time:
  - **Header** is a text (an atom) describing the header of the graph. ([genbar1:header/1]
  - **BarchartTitle** is a text (an atom) to be used as label, usually not very long. ([genbar1:title/1]
  - **XTitle** is a text (an atom) to be used as label, usually not very long. ([genbar1:title/1]
  - **XVector** is a list of **xbarelement1s**. ([basic_props:list/2]
  - **YTitle** is a text (an atom) to be used as label, usually not very long. ([genbar1:title/1]
  - **YVector** is a list of **yelement**s. ([basic_props:list/2]
  - **Footer** is a text (an atom) describing the footer of the graph. ([genbar1:footer/1]

**barchart1/9:**

**PREDICATE**

You can set the minimum and maximum limits of the y axis. Data outside the limits will not be plotted. Each limit, as you can check by looking at the **axis_limit/1** definition, is a number. If the argument is a variable the limit will be calculated from the data (i.e., if **YMax** value is **YValueMax** the maximum y axis limit will calculated using the largest data value).

Example:
barchart1('This is the header text',
'Barchart title',
'xaxistitle',
[['element1','e1','Blue','Yellow','pattern1'],
 ['element2','e2','Turquoise','Plum','pattern5'],
 ['element3','e3','Turquoise','Green','pattern5'] ],
'yaxixtitle',
[20,10,59],
70,
-',
'footer').

Usage:
- The following properties should hold at call time:
  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  BTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XVector is a list of xbarelement1s. (basic_props:list/2)
  YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  YVector is a list of yelements. (basic_props:list/2)
  genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
  genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

percentbarchart1/7:
PREDICATE
percentbarchart1(Header, BTitle, XTitle, XVector, YTitle, YVector, Footer)
The y axis maximum coordinate value is 100. The x axis limits are automatically worked out.
Example:
percentbarchart1('This is a special barchart to represent percentages',
'Barchart with legend',
'My xaxistitle',
[1,'bar1','Blue','Yellow','pattern1'],
[8,'bar2','MediumTurquoise','Plum','pattern5'] ],
'My yaxixtitle',
[80,10],
'This is the footer text').

Usage:
- The following properties should hold at call time:
  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  BTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XVector is a list of xbarelement1s. (basic_props:list/2)
YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
YVector is a list of yelements. (basic.props: list/2)
Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

**yelement/1:**
```prolog
yelement(Y) :-
    number(Y).
```

Y is the bar length, so it must be a numeric value.
Both Prolog and Tcl-Tk support integers and floats. Integers are usually specified in
decimal, but if the first character is 0 the number is read in octal (base 8), and if the first
two characters are 0x, the number is read in hexadecimal (base 16). Float numbers may
be specified using most of the forms defined for ANSI C, including the following examples:
- 9.56
- 5.88e-2
- 5.1E2

Note: Be careful when using floats. While 8. or 7.e4 is interpreted by Tcl-tk as 8.0 and
7.0e4, Prolog will not read them as float numbers. Example:
```prolog
?- number(8.e+5).
{SYNTAX ERROR: (lns 130-130), or ) expected in arguments
number ( 8
** here **
. e + 5 ) .
}
```
```prolog
no
?- number(8.).
{SYNTAX ERROR: (lns 138-138), or ) expected in arguments
number ( 8
** here **
. ) .
}
```
```prolog
no
?- number(8.0e+5).
```
```prolog
yes
?- number(8.0).
```
```prolog
yes
```

Precision: Tcl-tk internally represents integers with the C type `int`, which provides at
least 32 bits of precision on most machines. Since Prolog integers can (in some implementa-
tions) exceed 32 bits but the precision in Tcl-tk depends on the machine, it is up to the
programmer to ensure that the values fit into the maximum precision of the machine for
integers. Real numbers are represented with the C type `double`, which is usually repre-
sented with 64-bit values (about 15 decimal digits of precision) using the IEEE Floating
Point Standard.
Conversion: If the list is composed by integers and floats, Tcl-tk will convert integers to floats.

axis_limit/1:  

\[
\text{axis}_{\text{limit}}(X) :-  
\text{number}(X).  
\text{axis}_{\text{limit}}(_1).  
\]

This type is defined in order to set the minimum and maximum limits of the axes. Data outside the limits will not be plotted. Each limit is a number or a variable. If the argument is not a number the limit will be calculated from the data (i.e., if YMax value is \text{Var} the maximum y axis limit will be calculated using the largest data value).

header/1:  

\[
\text{Usage: header}(X)  
\text{Description: } X \text{ is a text (an atom) describing the header of the graph.}  
\]

title/1:  

\[
\text{Usage: title}(X)  
\text{Description: } X \text{ is a text (an atom) to be used as label, usually not very long.}  
\]

footer/1:  

\[
\text{Usage: footer}(X)  
\text{Description: } X \text{ is a text (an atom) describing the footer of the graph.}  
\]

186.3 Documentation on internals (genbar1)

xbarelement1/1:  

\[
\text{xbarelement1}([XValue,LegendElement]) :-  
\text{atomic}(XValue),  
\text{atomic}(LegendElement).  
\text{xbarelement1}([XValue,LegendElement,ForegColor,BackgColor,SPattern]) :-  
\text{atomic}(XValue),  
\text{atomic}(LegendElement),  
\text{color}(ForegColor),  
\text{color}(BackgColor),  
\text{pattern}(SPattern).  
\]

 Defines the attributes of the bar.

\text{XValue}  
bar label. Although \text{XValue} values may be numbers, the will be treated as labels. Different elements with the same label will produce different bars.

\text{LegendElement}  
Legend element name. It may be a number or an atom and equal or different to the \text{XValue}. Every \text{LegendElement} value of the list must be unique.
ForeColor
It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

BackColor
It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

SPattern It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.
Chapter 187: Barchart widgets - 2

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#158 (2003/12/4, 17:39:21 CET)

This module defines predicates which show barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window. They all share the following features:

- Numeric values for the x axis are needed, otherwise it does not work properly. See xbarelement2/1 definition type.
- The bar position is proportional to the x-coordinate value.
- The user can either select the appearance of the bars (background color, foreground color and stipple style) or not. See the xbarelement2/1 type definition. Thus, the user can call each predicate in two ways.
- The bar chart has a legend and one entry (symbol and label) per bar.
- If you do not want to display text in the elements header, barchart title, x axis title, y axis title or footer, simply type '' as the value of the argument.
- The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception error2 will be thrown. If the vectors contain elements but are not correct, the exception error1 or error3 will be thrown, depending on the error type. error1 means that XVector and YVector does not contain the same number of elements and error3 indicates that not all the XVector elements contain a correct number of attributes.

Particular features will be pointed out in the corresponding predicate.

187.1 Usage and interface (genbar2)

- Library usage:
  :- use_module(library(genbar2)).
- Exports:
  - Predicates:
    barchart2/7, barchart2/11, percentbarchart2/7.
  - Regular Types:
    xbarelement2/1.
- Other modules used:
  - System library modules:
    chartlib/genbar1, chartlib/bltclass, chartlib/color_pattern, chartlib/test_format, chartlib/install_utils, lists, random/random.

187.2 Documentation on exports (genbar2)

barchart2/7:  PREDICATE
barchart2(Header, BarchartTitle, XTitle, XVector, YTitle, YVector, Footer)

The maximum and minimum limits for axes are determined from the data.

Example:
barchart2('This is the header text',
'Bar Chart with legend',
'My xaxistitle',
[ [1,'bar1','Blue','Yellow','pattern1'],
  [2,'bar2','MediumTurquoise','Plum','pattern5'] ],
'My yaxixtitle',
[20,10],
'This is the footer text').

Usage:
- The following properties should hold at call time:
  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  BarchartTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XVector is a list of xbarelement2s. (basic_props:list/2)
  YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  YVector is a list of yelement2s. (basic_props:list/2)
  Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

barchart2/11:                                        PREDICATE
barchart2(Header, BT, XT, XVector, XMax, XMin, YT, YVector, YMax, YMin, Footer)

You can set the minimum and maximum limits of the axes. Data outside the limits will
not be plotted. Each limit, as you can check looking at the axis_limit/1 definition, is a
number. If the argument is a variable the limit will be calculated from the data (i.e., if
YMax value is YValueMax the maximum y axis limit will calculated using the largest data
value).
Example:

barchart2('This is the header text',
'Bar Chart with legend',
'My xaxistitle',
[ [1,'bar1',Color1,Color2,Pattern1],
  [2,'bar2',Color3,Color4,Pattern2] ],
10,
-10,
'My yaxixtitle',
[20,10],
100,
-10,
'The limits for the axes are set by the user').

Usage:
- The following properties should hold at call time:
  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  BT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XVector is a list of xbarelement2s. (basic_props:list/2)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
YVector is a list of yelements. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

percentbarchart2/7:

percentbarchart2(Header, BTitle, XTitle, XVector, YTitle, YVector, Footer)
The y axis maximum coordinate value is 100. The x axis limits are autoarrange.
Example:
percentbarchart2('This is a special barchart to represent percentages', 'Barchart with legend', 'My xaxistitle', [ [1,'bar1','Blue','Yellow','pattern1'], [2,'bar2','MediumTurquoise','Plum','pattern5'] ], 'My yaxixtitle', [80,10], 'This is the footer text').

Usage:
The following properties should hold at call time:

Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
BTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XVector is a list of xbarelement2s. (basic_props:list/2)
YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
YVector is a list of yelements. (basic_props:list/2)
Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

xbarelement2/1:

xbarelement2([XValue,LegendElement]) :-
  number(XValue),
  atomic(LegendElement).
xbarelement2([XValue,LegendElement,ForegColor,BackgColor,SPattern]) :-
  number(XValue),
  atomic(LegendElement),
  color(ForegColor),
  color(BackgColor),
  pattern(SPattern).

Defines the attributes of the bar.
XValue x-coordinate position of the bar. Different elements with the same abscissas will produce overlapped bars.

LegendElement Element legend name. It may be a number or an atom and equal or different to the XValue. Every LegendElement value of the list must be unique.

ForegColor Is the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

BackgColor Is the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

SPattern Is the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.
Chapter 188: Depict barchart widgets - 3

**Author(s):** Isabel Martín García.

**Version:** 1.11#222 (2004/5/24, 13:8:7 CEST)

**Version of last change:** 1.11#146 (2004/1/1, 14:9:17 CET)

This module defines predicates which depict barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window and are similar to those exported in the genbar1 module except in that these defined in this module do not display a legend. Thus, not all the argument types are equal.

The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contain elements but are not correct, the exception `error1` or `error3` will be thrown, depending on the error type. `error1` means that `XVector` and `YVector` do not contain the same number of elements and `error3` indicates that not all the `XVector` elements contain a correct number of attributes.

### 188.1 Usage and interface (genbar3)

- **Library usage:**
  ```prolog```
  ```
  :- use_module(library(genbar3)).
  ```
- **Exports:**
  - **Predicates:**
    ```prolog```
    ```
    barchart3/7, barchart3/9, percentbarchart3/7.
    ```
- **Other modules used:**
  - **System library modules:**
    ```prolog```
    ```
    chartlib/genbar1, chartlib/bltclass, chartlib/color_pattern, chartlib/test_format, chartlib/install_utils, lists, random/random.
    ```

### 188.2 Documentation on exports (genbar3)

**barchart3/7:**

```prolog```
```
  barchart3(Header, BarchartTitle, XTitle, XVector, YTitle, YVector, Footer)
```

As we mentioned in the above paragraph, this predicate is comparable to `barchart1/8` except in the `XVector` argument type.

Example:

```prolog```
```
barchart3('This is the header text',
  'Barchart without legend',
  'My xaxistitle',
  [[['bar1'], ['bar2']]],
  'My yaxixtitle',
  [20, 10],
  'This is the footer text').
```

Usage:

- **The following properties should hold at call time:**
  - `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
BarchartTitle is a text (an atom) to be used as label, usually not very long.
(genbar1:title/1)
XTitle is a text (an atom) to be used as label, usually not very long.
(genbar1:title/1)
XVector is a list of xbarelement3s.  
(basic_props:list/2)
YTitle is a text (an atom) to be used as label, usually not very long.
(genbar1:title/1)
YVector is a list of yelements.
(basic_props:list/2)
Footer is a text (an atom) describing the footer of the graph. 
(genbar1:footer/1)

barchart3/9:  
PREDICATE
barchart3(Header, BTitle, XTitle, XVector, YTitle, YVector, YMax, YMin, Footer)

As we mentioned, this predicate is quite similar to the barchart1/10 except in the
XVector argument type, because the yielded bar chart lacks of legend.

Example:

barchart3('This is the header text',
    'Barchart without legend',
    'My xaxistitle',
    [['bar1'], ['bar2']],
    'My yaxixtitle',
    30,
    5,
    [20,10],
    'This is the footer text').

Usage:

– The following properties should hold at call time:

Header is a text (an atom) describing the header of the graph. 
(genbar1:header/1)
BTitle is a text (an atom) to be used as label, usually not very long. 
(genbar1:title/1)
XTitle is a text (an atom) to be used as label, usually not very long. 
(genbar1:title/1)
XVector is a list of xbarelement3s.  
(basic_props:list/2)
YTitle is a text (an atom) to be used as label, usually not very long. 
(genbar1:title/1)
YVector is a list of yelements.
(basic_props:list/2)

Footer is a text (an atom) describing the footer of the graph. 
(genbar1:footer/1)

percentbarchart3/7:  
PREDICATE
percentbarchart3(Header, BTitle, XTitle, XVector, YTitle, YVector, Footer)

The y axis maximum coordinate value is 100. The x axis limits are autoarrange.

Example:
percentbarchart3('This is a special barchart to represent percentages',
'Barchart without legend',
'My xaxistitle',
[['pr1','Blue','Yellow','pattern1'],
  ['pr2','MediumTurquoise','Plum','pattern5'] ],
'My yaxixtitle',
[80,10],
'This is the footer text').

Usage:
- The following properties should hold at call time:
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **BTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XVector** is a list of xbarelement3s. (basic_props:list/2)
  - **YTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **YVector** is a list of yelements. (basic_props:list/2)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

### 188.3 Documentation on internals (genbar3)

**xbarelement3/1:**

REGTYPE

```prolog
xbarelement3([XValue]) :- atomic(XValue).
xbarelement3([XValue,ForegColor,BackgColor,StipplePattern]) :-
  atomic(XValue),
  color(ForegColor),
  color(BackgColor),
  pattern(StipplePattern).
```

Defines the attributes of the bar.

**XValue** bar label. Although XValue values may be numbers, the will be treated as labels. Different elements with the same label will produce different bars.

**ForegColor**

It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

**BackgColor**

It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

**SPattern**

It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.
189 Depict barchart widgets - 4

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#147 (2004/1/1, 14:16:0 CET)

This module defines predicates which depict barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window and are similar to those exported in genbar2 module except that those defined in this module don’t display a legend. Thus, the user does not have to define legend element names.

The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception error2 will be thrown. If the vectors contains elements but are not correct, the exception error1 or error3 will be thrown, depending on the error type. error1 means that XVector and YVector do not contain the same number of elements and error3 indicates that not all the XVector elements contain a correct number of attributes.

189.1 Usage and interface (genbar4)

- Library usage:
  ```prolog```
  ```:- use_module(library(genbar4)). ```
- Exports:
  - Predicates:
    ```prolog```
    ```barchart4/7, barchart4/11, percentbarchart4/7. ```
- Other modules used:
  - System library modules:
    ```prolog```
    ```chartlib/genbar1, chartlib/bltclass, chartlib/color_pattern, chartlib/test_format, chartlib/install_utils, lists, random/random. ```

189.2 Documentation on exports (genbar4)

barchart4/7: 

```prolog```
```barchart4(Header, BarchartTitle, XTitle, XVector, YTitle, YVector, Footer) ```

As we mentioned in the above paragraph, this predicate is comparable to barchart2/8 except in the XVector argument type.

Example:

```prolog```
```barchart4('This is the header text', 'Barchart without legend', 'My xaxistitle', [[2],[5],[6]], 'My yaxistitle', [20,10,59], 'Numeric values in the xaxis'). ```

Usage:

- The following properties should hold at call time:
  ```prolog```
  ```Header is a text (an atom) describing the header of the graph. (genbar1:header/1) ```
BarchartTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XVector is a list of xbarelement4s. (basic_props:list/2)
YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
YVector is a list of yelements. (basic_props:list/2)
Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

barchart4/11:
PREDICATE
barchart4(Hder, BT, XT, XVector, XMax, XMin, YT, YVector, YMax, YMin, Fter)
As we stated before, this predicate is quite similar to barchart2/10 except in the following aspects:

• The XVector argument type, because the yielded bar chart lacks the legend.
• The user can set limits for both x axis and y axis.

Example:

barchart4('This is the header text, you can write a graph description',
'Barchart without legend',
'My xaxistitle',
[[2,'Blue','Yellow','pattern1'],
 [20,'MediumTurquoise','Plum','pattern5'],
 [30,'MediumTurquoise','Green','pattern5']],
50,
-10,
'My yaxixtitle',
[20,10,59],
100,
-10,
'Numeric values in the xaxis').

Usage:

– The following properties should hold at call time:
  Hder is a text (an atom) describing the header of the graph. (genbar1:header/1)
  BT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XVector is a list of xbarelement4s. (basic_props:list/2)
  genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
  genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
  YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  YVector is a list of yelements. (basic_props:list/2)
  genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
  genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  Fter is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
percentbarchart4/7:

percentbarchart4(Header, BTitle, XTitle, XVector, YTitle, YVector, Footer)

The y axis maximum coordinate value is 100. The x axis limits are automatically worked out. This predicate is useful when the bar height represents percentages.

Example:

percentbarchart4('This is the header text',
               'Barchart without legend',
               'My xaxistitle',
               [[2,'Blue','Yellow','pattern1'],[5,'Yellow','Plum','pattern5'],
                [6,'MediumTurquoise','Green','pattern5']],
               'My yaxixtitle',
               [20,10,59],
               'Numeric values in the xaxis').

Usage:

– The following properties should hold at call time:

Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
BTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
XVector is a list of xbarelement4s. (basic_props:list/2)
YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
YVector is a list of yelements. (basic_props:list/2)
Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

189.3 Documentation on internals (genbar4)

xbarelement4/1:

Defines the attributes of the bar.

XValue x-coordinate position of the bar. Different elements with the same abscissas will produce overlapped bars.

ForegColor
It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

BackgColor
It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

SPattern It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.
190 Depic line graph

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#148 (2004/1/1, 14:17:31 CET)

This module defines predicates which depict line graph and scatter graph widgets. All eight predicates exported by this module plot two-variable data. Each point is defined by its X-Y coordinate values. A dataset is defined by two lists xvector and yvector, which contain the points coordinates. As you might guess, the values placed in the the same position in both lists are the coordinates of a point. They both share the following features:

- Numeric values for vector elements are needed. We'll use two vectors to represent the X-Y coordinates of each set of plotted data, but in this case every dataset shares the X-vector, i.e., x-coordinate of points with the same index¹ in different datasets is the same. Thus, the numbers of points in each yvector must be equal to the number of points in the xvector.
- The active element color is navyblue, which means that when you select a legend element, the corresponding line or point dataset turns into navyblue.
- The user can either select the appearance of the lines and/or points of each dataset or not. If not, the system will choose the colors for the lines and the points among the available ones in accordance with the plot background color and it will also set the points size and symbol to the default. If the plot background color is black, the system will choose a lighter color, and the system will select a darker color when the plot background color is white. Thus, the user can define the appearance attributes of each dataset in four different ways. Take a look at the attributes/1 type definition and see the examples to understand it clearly.
- The graph has a legend and one entry (symbol and label) per dataset.
- If you do not want to display text in the element header, barchart title, xaxis title, yaxis title or footer, simply give '' as the value of the argument.
- The predicates check whether the format of the arguments is correct as well. The testing process involves some verifications. If one or both vectors are empty, the exception error2 will be thrown. If the vectors contains elements but are not correct, the exception error4 will be thrown.

The names of the line graph predicates begin with graph_ and those corresponding to the scatter graph group begin with scattergraph_.

¹ It should be pointed out that I am refering to an index as the position of an element in a list.
190.1 Usage and interface (gengraph1)

- Library usage:
  
  ```prolog
  :- use_module(library(gengraph1)).
  ```

- Exports:
  - Predicates:
    
    ```prolog
    graph_b1/9, graph_b1/13, graph_w1/9, graph_w1/13, scattergraph_b1/8, scattergraph_b1/12, scattergraph_w1/8, scattergraph_w1/12.
    ```

  - Regular Types:
    
    ```prolog
    vector/1, smooth/1, attributes/1, symbol/1, size/1.
    ```

- Other modules used:
  - System library modules:
    
    ```prolog
    chartlib/bltclass, chartlib/genbar1, chartlib/color_pattern, chartlib/test_format, chartlib/install_utils, lists, random/random.
    ```

190.2 Documentation on exports (gengraph1)

graph_b1/9:

**PREDICATE**

```prolog
graph_b1(Header, GTitle, XTitle, XVector, YTitle, YVectors, LAtts, Footer, Smooth)
```

Besides the features mentioned at the begining of the chapter, the displayed graph generated when calling this predicate has the following distinguishing characteristics:

- The plotting area background color is black.
- The cross hairs color is white.
- The axes limits are determined from the data.

Example:

```prolog
graph_b1('This is the header text', 'Graph_title', 'xaxistitle', [20,10,59], 'yaxixtitle', [[10,35,40],[25,50,60]], [['element1','Blue','Yellow','plus',6],['element2',Outline,Color]], 'footer', 'linear').
```

Usage:

- **The following properties should hold at call time:**
  
  - **Header** is a text (an atom) describing the header of the graph. *(genbar1:header/1)*
  
  - **GTitle** is a text (an atom) to be used as label, usually not very long. *(genbar1:title/1)*
  
  - **XTitle** is a text (an atom) to be used as label, usually not very long. *(genbar1:title/1)*

  - **gengraph1:vector(XVector)** *(gengraph1:vector/1)*
YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
YVectors is a list of vectors. (basic_props:list/2)
LAtts is a list of attributes. (basic_props:list/2)
Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

**graph_b1/13:**

graph_b1(Header, GT, XT, XV, XMax, XMin, YT, YVs, YMax, YMin, LAtts, Footer, Smooth)

The particular features related to this predicate are described below:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted.

Example:

```
graph_b1('This is the header text',
        'Graph_title',
        'xaxistitle',
        [20,10,59], 50,
        '-',
        'yaxistitle',
        [[10,35,40],[25,50,60]], 50,
        '-',
        [['line1','circle',4],['line2','OutlineColor',Color]],
        'footer',
        'step').
```

**Usage:**

- *The following properties should hold at call time:*
  - Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - GT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - gengraph1:vector(XV) (gengraph1:vector/1)
  - genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
  - genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
  - YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - YVs is a list of vectors. (basic_props:list/2)
  - genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
  - genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  - LAtts is a list of attributes. (basic_props:list/2)
  - Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
  - gengraph1:smooth(Smooth) (gengraph1:smooth/1)
**graph_w1/9:**

```prolog
graph_w1/9(Header, GTitle, XTitle, XVector, YTitle, YVectors, LAtts, Footer, Smooth)
```

This predicate is quite similar to `graph_b1/9`. The differences lies in the plot background color and in the cross hairs color, which are white and black respectively.

Example:

```prolog
graph_w1('This is the header text', 'Graph_title', 'xaxistitle', [20,10,40,50], 'yaxixtitle', [[10,35,40,50],[25,20,60,40]], [['line1','Blue','DarkOrchid'],['line2','circle',3]], 'footer', 'quadratic').
```

**Usage:**

- *The following properties should hold at call time:*
  - **Header** is a text (an atom) describing the header of the graph. (*genbar1:header/1*)
  - **GTitle** is a text (an atom) to be used as label, usually not very long. (*genbar1:title/1*)
  - **XTitle** is a text (an atom) to be used as label, usually not very long. (*genbar1:title/1*)
  - **YTitle** is a text (an atom) to be used as label, usually not very long. (*genbar1:title/1*)
  - **YVectors** is a list of vectors. (*basic Props:list/2*)
  - **LAtts** is a list of attributes. (*basic Props:list/2*)
  - **Footer** is a text (an atom) describing the footer of the graph. (*genbar1:footer/1*)
  - **Smooth** is a text (an atom) describing the smooth of the graph. (*gengraph1:smooth/1*)

**graph_w1/13:**

```prolog
graph_w1/13(Header, GT, Xt, Xv, XMax, XMin, YT, YVs, YMax, YMin, LAtts, Footer, Smooth)
```

This predicate is quite similar to `graph_b1/13`, the differences between them are listed below:

- The plotting area background color is white.
- The cross hairs color is black.

Example:

```prolog
graph_w1('This is the header text', 'Graph_title', 'xaxistitle', [20,10,59], 100, 10, 'yaxixtitle', [[10,35,40],[25,20,60]], 'footer', 'quadratic').
```
Chapter 190: Depic line graph

Usage:

- The following properties should hold at call time:
  
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **GTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **gengraph1:vector(XV)** (gengraph1:vector/1)
  - **genbar1:axis_limit(XMax)** (genbar1:axis_limit/1)
  - **genbar1:axis_limit(XMin)** (genbar1:axis_limit/1)
  - **YT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **YVs** is a list of vectors. (basic_props:list/2)
  - **genbar1:axis_limit(YMax)** (genbar1:axis_limit/1)
  - **genbar1:axis_limit(YMin)** (genbar1:axis_limit/1)
  - **LAtts** is a list of attributes. (basic_props:list/2)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
  - **gengraph1:smooth(Smooth)** (gengraph1:smooth/1)

**scattergraph_b1/8:**

```
scattergraph_b1(Header, GTitle, XTitle, XVector, YTitle, YVectors, PAtts, Footer)
```

Apart from the features brought up at the beginning of the chapter, the scatter graph displayed invoking this predicate has the following characteristics:

- The plotting area background color is black.
- The cross hairs color is white.
- The axes limits are determined from the data.

Example:

```
scattergraph_b1('This is the header text',
    'Graph_title',
    'xaxistitle',
    [10,15,20],
    'yaxixtitle',
    [[10,35,20],[15,11,21]],
    [['element1','Blue','Yellow'],['element2','Turquoise','Plum']],
    'footer').
```

Usage:

- The following properties should hold at call time:
  
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **GTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **gengraph1:vector(XVector)** (gengraph1:vector/1)
YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

YVectors is a list of vectors. (basic_props:list/2)

PAtts is a list of attributes. (basic_props:list/2)

Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_b1/12:

scattergraph_b1(Header, GT, XT, XV, XMax, XMin, YT, YVs, YMax, YMin, PAtts, Footer)

The particular features related to this predicate are described below:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted.

Example:

scattergraph_b1('This is the header text',
                 'Graph_title',
                 'xaxistitle',
                 [20,10,59],
                 50,
                 '-',
                 'yaxixtitle',
                 [[10,35,40],[25,50,60]],
                 50,
                 '-',
                 [['point dataset1','Blue','Yellow'],['point dataset2']],
                 'footer').

Usage:

- The following properties should hold at call time:
  - Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - GT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - gengraph1:vector(XV) (gengraph1:vector/1)
  - genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
  - genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
  - YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - YVs is a list of vectors. (basic_props:list/2)
  - genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
  - genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  - PAtts is a list of attributes. (basic_props:list/2)
  - Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_w1/8:

scattergraph_w1(Header, GT, XT, XVector, YT, YVectors, PAtts, Footer)

This predicate is quite similar to scattergraph_b1/8 except in the following:
• The plotting area background color is black.
• The cross hairs color is white.
• If the user does not fix the points colors, they will be chosen among the lighter ones.

Example:

```
scattergraph_w1('This is the header text',
   'Graph_title',
   'xaxistitle',
   [20,10,59],
   'yaxixtitle',
   [[10,35,40],[25,20,60]],
   [['e1','Blue','Green'],[e2,'MediumVioletRed','Plum']],
   'footer').
```

Usage:

- The following properties should hold at call time:

  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **GT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **gengraph1:vector(XVector)**
  - **YT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **YVectors** is a list of vectors. (basic_props:list/2)
  - **PAtts** is a list of attributes. (basic_props:list/2)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

---

**scattergraph_w1/12:**

**PREDICATE**

```
scattergraph_w1(Header, GT, XT, XV, XMax, XMin, YT, YVs, YMax, YMin, PAtts, Footer)
```

This predicate is quite similar to **scattergraph1_b1/13**, the differences between them are listed below:

• The plotting area background color is white.
• The cross hairs color is black.

Example:

```
scattergraph_w1('This is the header text',
   'Graph_title',
   'xaxistitle',
   [20,10,59],
   150,
   5,  
   'yaxixtitle',
   [[10,35,40],[25,20,60]],
   _,
   -10,  
   [['e1','Blue','Yellow'],[e2,'MediumTurquoise','Plum']],
   'footer').
```

Usage:
The following properties should hold at call time:

- **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
- **GT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
- **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
- **gengraph1:vector(XV)** (gengraph1:vector/1)
- **genbar1:axis_limit(XMax)** (genbar1:axis_limit/1)
- **genbar1:axis_limit(XMin)** (genbar1:axis_limit/1)
- **YT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
- **YVs** is a list of vectors. (basic_props:list/2)
- **genbar1:axis_limit(YMax)** (genbar1:axis_limit/1)
- **genbar1:axis_limit(YMin)** (genbar1:axis_limit/1)
- **PAtts** is a list of attributes. (basic_props:list/2)
- **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

---

**vector/1:**

```prolog
vector(X) :-
  list(X, number).
```

The type `vector` defines a list of numbers (integers or floats).

---

**smooth/1:**

```prolog
smooth(Smooth)
  smooth(linear).
  smooth(cubic).
  smooth(quadratic).
  smooth(step).
```

Specifies how connecting segments are drawn between data points. If `Smooth` is `linear`, a single line segment is drawn, connecting both data points. When `Smooth` is `step`, two line segments will be drawn, the first line is a horizontal line segment that steps the next X-coordinate and the second one is a vertical line, moving to the next Y-coordinate. Both `cubic` and `quadratic` generate multiple segments between data points. If `cubic` is used, the segments are generated using a cubic spline. If `quadratic`, a quadratic spline is used. The default is linear.

---

**attributes/1:**

```prolog
attributes([ElementName]) :-
  atomic(ElementName).
attributes([ElementName, OutLine, Color]) :-
  atomic(ElementName),
  color(OutLine),
  color(Color).
attributes([ElementName, Symbol, Size]) :-
  atomic(ElementName),
  symbol(Symbol),
  size(Size).
```
attributes([ElementName, OutLine, Color, Symbol, Size]) :-
  atomic(ElementName),
  color(OutLine),
  color(Color),
  symbol(Symbol),
  size(Size).

Each line or point dataset in the graph has its own attributes, which are defined by this type. The name of the dataset, specified in the ElementName argument, may be either a number or an atom. The second argument is the color of a thin line around each point in the dataset and the Color argument is the points and lines color. Both OutLine and Color must be a valid color (see available values in color/1), otherwise a random color according to the plot background color will be selected. The Symbol must be a valid symbol and the Size must be a number. Be careful if you want to specify the Symbol and the Size, otherwise the predicate will not work as you expect. If you don’t select a symbol and a size for a dataset the default values will be square and 1 pixel.

**symbol/1**:  
symbol(Symbol)
  symbol(square).
symbol(circle).
symbol(diamond).
symbol(plus).
symbol(cross).
symbol(splus).
symbol(scross).
symbol(triangle).

Symbol stands for the shape of the points whether in scatter graphs or in line graphs.

**size/1**:  
size(Size)
  size(Size) :-
    number(Size).

Size stands for the size in pixels of the points whether in scatter graphs or in line graphs.
191 Line graph widgets

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.11#149 (2004/1/1, 14:19:37 CET)

This module defines predicates which show line graph widgets. All eight predicates exported by this module plot two-variable data. Each point is defined by its X-Y coordinate values. Every predicate share the following features:

- A dataset is defined by three lists xvector, yvector and attributes. The arguments named XVectors (or XVs), YVectors (or YVs) and LAtts\(^1\) contain this information. Those arguments must be lists whose elements are also lists. The first dataset is defined by the firts element of the three lists, the second dataset is defined by the second element of the three lists and so on.
- Numeric values for the vector elements are needed. We will use two vectors to represent the X-Y coordinates of each set of data plotted. In these predicates the vectors can have different number of points. However, the number of elements in xvector and yvector that pertain to a certain dataset must be, obviously, equal.
- The active line color is blue, which means that when you select a legend element, the corresponding line turns into navyblue.
- The user can either select the appearance for the lines and the points or not. See the attributes/1 type definition. Thus, the user can call each predicate in different ways.
- The graph has a legend and one entry (symbol and label) per dataset.
- If you do not want to display text in the elements header, barchart title, xaxis title, yaxis title or footer, simply give ’’ as the value of the argument.
- The predicates check whether the format of the arguments is correct as well. The testing process involves some verifications. If one or both vectors are empty, the exception error2 will be thrown. If the vectors contains elements but are not correct, the exception error4 will be thrown.

191.1 Usage and interface (gengraph2)

- Library usage:
  ```prolog```
  :- use_module(library(gengraph2)).
  ```prolog```

- Exports:
  - Predicates:
    ```prolog```
    graph_b2/9, graph_b2/13, graph_w2/9, graph_w2/13, scattergraph_b2/8, scattergraph_b2/12, scattergraph_w2/8, scattergraph_w2/12.
    ```prolog```
  - Other modules used:
    ```prolog```
    chartlib/gengraph1, chartlib/genbar1, chartlib/bltclass, chartlib/color_pattern, chartlib/test_format, chartlib/install_utils, lists, random/random.
    ```prolog```

\(^1\) In scatter graphs the attribute that contains the features of a point dataset is PAtts.
191.2 Documentation on exports (gengraph2)

**graph_b2/9:**
```
graph_b2(Header, GTitle, XTitle, XVectors, YTitle, YVectors, LAtts, Footer, Sm)
```

Besides the features mentioned at the beginning of the module chapter, the displayed graph generated calling this predicate has the following distinguish characteristics:

- The plotting area background color is black.
- The cross hairs color is white.
- The axis limits are determined from the data.

Example:
```
graph_b2('This is the header text',
    'Graph_title',
    'xaxistitle',
    [[20,30,59],[25,50]],
    'yaxixtitle',
    [[10,35,40],[25,50]],
    [['line1','Blue','Yellow'],['line2']],
    'footer',
    'natural').
```

**Usage:**
- *The following properties should hold at call time:*
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **GTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XVectors** is a list of vectors. (basic_props:list/2)
  - **YTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **YVectors** is a list of vectors. (basic_props:list/2)
  - **LAtts** is a list of attributes. (basic_props:list/2)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
  - **gengraph1:smooth(Sm)** (gengraph1:smooth/1)

**graph_b2/13:**
```
graph_b2(Header, GT, XT, XVs, XMax, XMin, YT, YVs, YMax, YMin, LAtts, Footer, Smooth)
```

In addition to the features brought up at the beginning of the module chapter, this graph has the following:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the maximum and minimum values for the graph axes.

Example:
Chapter 191: Line graph widgets

graph_b2('This is the header text', 'Graph_title', 'xaxistitle', [[20,10,59],[15,30,35]], 50, 'yaxixtitle', [[10,35,40],[25,50,60]], 50.5, [[['line1','Blue','Yellow'],['line','MediumTurquoise','Plum']], 'footer', 'step').

Usage:
- The following properties should hold at call time:
  - Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - GT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - XVs is a list of vectors. (basic_props:list/2)
    - genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
    - genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
  - YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - YVs is a list of vectors. (basic_props:list/2)
    - genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
    - genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  - LAtts is a list of attributes. (basic_props:list/2)
  - Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
  - gengraph1:smooth(Smooth) (gengraph1:smooth/1)

graph_w2/9:

This predicate is quite similar to graph_b2/9. The difference lies in the graph appearance, as you can see below.
- The plotting area background color is white.
- The cross hairs color is black.

Example:

graph_w2('This is the header text', 'Graph_title', 'xaxistitle', [[10,30,59],[25,50]], 'yaxixtitle', [[10,35,40],[25,40]], [[['element1','Blue','DarkOrchid'],['element2','DarkOliveGreen','Firebrick']], 'footer', 'natural').

Usage:
The following properties should hold at call time:

- **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
- **GT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
- **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
- **XVectors** is a list of vectors.
- **YTitle** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
- **YVectors** is a list of vectors.
- **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
- **LAtts** is a list of attributes. (basic_props:list/2)
- **gengraph1:smooth(Smooth)** (gengraph1:smooth/1)

**graph_w2/13:**

\[
graph_w2(Header, GT, XT, XV, XMax, XMin, YT, YVs, YMax, YMin, LAtts, Footer, Smooth)\]

This predicate is comparable to **graph_b2/13**. The differences lie in the plot background color and in the cross hairs color, which are white and black respectively.

**Example:**

\[
\begin{align*}
&graph_w2('This is the header text', \\
&'Graph_title', \\
&'xaxistitle', \\
&[[10,30,59],[10,35,40]], \\
&80, \\
&', \\
&'yaxistitle', \\
&[[10,35,40],[25,50,60]], \\
&50, \\
&', \\
&[['element1','Blue','Green'],[['element2','Turquoise','Black']], \\
&'footer', \\
&'linear').
\end{align*}
\]

**Usage:**

- The following properties should hold at call time:
  
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **GT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XV** is a list of vectors. (basic_props:list/2)
  - **genbar1:axis_limit(XMax)** (genbar1:axis_limit/1)
  - **genbar1:axis_limit(XMin)** (genbar1:axis_limit/1)
  - **YT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **YVs** is a list of vectors. (basic_props:list/2)
  - **genbar1:axis_limit(YMax)** (genbar1:axis_limit/1)
  - **genbar1:axis_limit(YMin)** (genbar1:axis_limit/1)
  - **LAtts** is a list of attributes. (basic_props:list/2)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
  - **gengraph1:smooth(Smooth)** (gengraph1:smooth/1)
scattergraph_b2/8:

scattergraph_b2(Header, GT, XT, XVectors, YT, YVectors, PAtts, Footer)

Apart from the features brought up at the beginning of the chapter, the scatter graph displayed when invoking this predicate has the following features:

- The plotting area background color is black.
- The cross hairs color is white.
- The axis limits are determined from the data.

Example:

```
scattergraph_b2('This is the header text',
    'Graph_title',
    'xaxistitle',
    [[10,15,20],[8,30,40]],
    'yaxixtitle',
    [[10,35,20],[15,11,21]],
    ['element1','Blue','Yellow'],
    ['element2','MediumTurquoise','Plum'],
    'footer').
```

Usage:

- The following properties should hold at call time:
  - `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - `GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - `XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - `XVectors` is a list of vectors. (basic_props:list/2)
  - `YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - `YVectors` is a list of vectors. (basic_props:list/2)
  - `PAtts` is a list of attributes. (basic_props:list/2)
  - `Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_b2/12:

scattergraph_b2(Header, GT, XT, XVs, XMax, XMin, YT, YVs, YMax, YMin, PAtts, Footer)

The particular features related to this predicate are described below:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted.

Example:

```
scattergraph_b2('This is the header text',
    'Graph_title',
    'xaxistitle',
    [[20,30,50],[18,40,59]],
    50,
    'yaxixtitle',
    [[10,35,40],[25,50,60]],
    50,
```
- `[[point dataset1],[point dataset2]],
  'footer').

Usage:

- The following properties should hold at call time:

  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  GT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XVs is a list of vectors. (basic_pros:list/2)
  genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
  genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
  YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  YVs is a list of vectors. (basic_pros:list/2)
  genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
  genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  PAtts is a list of attributes. (basic_pros:list/2)
  Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_w2/8:

scattergraph_w2(Header, GTitle, XTitle, XVs, YTitle, YVs, PAtts, Footer)

This predicate is quite similar to scattergraph_w1/8 except in the following:

- The plotting area background color is black.
- The cross hairs color is white.
- If the user do not provide the colors of the points, they will be chosen among the lighter ones.

Example:

```
scattergraph_w2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[20,30,40,15,30,35,20,30]],
  'yaxixtitie',
  [[10,30,40,25,20,25,20,25]],
  [['set1','cross',4]],
  'footer').
```

Usage:

- The following properties should hold at call time:

  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  GTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  gengraph1:vector(XVs) (gengraph1:vector/1)
  YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  YVs is a list of vectors. (basic_pros:list/2)
  PAtts is a list of attributes. (basic_pros:list/2)
  Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
**scattergraph_w2/12:**

This predicate is comparable to `scattergraph_w2/13`, the differences between them are listed below:

- The plotting area background color is white.
- The cross hairs color is black.

Example:

```c
scattergraph_w2('This is the header text',
    'Graph_title',
    'xaxistitle',
    [[20,10,59],[15,30,50]],
    150,
    5,
    'yaxistitle',
    [[10,35,40],[25,20,60]],
    -, -10,
    ['e1','Blue','Yellow'],['e2','MediumTurquoise','Plum']],
    'footer')
```

**Usage:**

- *The following properties should hold at call time:*
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - **GT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **XVs** is a list of vectors. (basic_props:list/2)
  - **genbar1:axis_limit(XMax)** (genbar1:axis_limit/1)
  - **genbar1:axis_limit(XMin)** (genbar1:axis_limit/1)
  - **YT** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **YVs** is a list of vectors. (basic_props:list/2)
  - **genbar1:axis_limit(YMax)** (genbar1:axis_limit/1)
  - **genbar1:axis_limit(YMin)** (genbar1:axis_limit/1)
  - **PAtts** is a list of attributes. (basic_props:list/2)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
Chapter 192: Multi barchart widgets

Author(s): Isabel Martín García.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.11#154 (2004/1/7, 15:44:30 CET)

This module defines predicates which show barchart widgets. These bar charts are somewhat different from the bar charts generated by the predicates in modules genbar1, genbar2, genbar3 and genbar4. Predicates in the present module show different features of each dataset element in one chart at the same time. Each bar chart element is a group of bars, and the element features involve three vectors defined as follows:

- xvector is a list containing the names (atoms) of the bars (n elements). Each bar group will be displayed at uniform intervals.
- yvector is a list that contains m sublists, each one is composed of n elements. The i-sublist contains the y-values of the i-BarAttribute element for all of the XVector elements.
- bar_attributes is a list containing the appearance features of the bars (m elements). Each element of the list can be partial or complete, which means that you can define as bar attributes only the element name or by setting the element name, its background and foreground color and its stipple pattern.

Other relevant aspects about this widgets are:

- If you don’t want to display text in the elements header, barchart title, xaxis title, yaxis title or footer, simply type ‘’ as the value of the argument.
- The bar chart has a legend, and one entry (symbol and label) per feature group bar.
- The user can either select the appearance of the bars (background color, foreground color and stipple style) or not. See the multibar_attribute type definition.
- Data points can have their bar segments displayed in one of the following modes: stacked, aligned, overlapped or overlayed. They user can change the mode clicking in the checkboxes associated to each mode.
- The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception error2 will be thrown. If the vectors contains elements but are not correct, the exception error5 or error6 will be thrown, depending on what is incorrect. error5 means that XVector and each element of YVector do not contain the same number of elements or that YVector and BarsAtt do not contain the same number of elements, while error6 indicates that not all the BarsAtt elements contain a correct number of attributes.

The examples will help you to understand how these predicates should be called.

192.1 Usage and interface (genmultibar)

- Library usage:
  ```prolog
  :- use_module(library(genmultibar)).
  ```
- Exports:
  - Predicates:
    ```prolog
    multibarchart/8, multibarchart/10.
    ```
- Other modules used:
  - System library modules:
    ```prolog
    chartlib/genbar1, chartlib/bltclass, chartlib/color_pattern,
    chartlib/test_format, chartlib/install_utils, lists, random/random.
    ```
192.2 Documentation on exports (genmultibar)

multibarchart/8:

```
multibarchart(Header, BTitle, XTitle, XVector, YTitle, BarsAtts, YVector, Footer)
```

The x axis limits are autoarrange. The user can call the predicate in two ways. In the first example the user sets the appearance of the bars, in the second one the appearance features will be chosen by the library.

Example1:
```
multibarchart('This is the Header text',
'My BarchartTitle',
'Processors',
['processor1','processor2','processor3','processor4'],
'Time (seconds)',
[['setup time','MediumTurquoise','Plum','pattern2'],
['sleep time','Blue','Green','pattern5'],
['running time','Yellow','Plum','pattern1']],
[[20,30,40,50],[10,8,5,35],[60,100,20,50]],
'This is the Footer text').
```

Example2:
```
multibarchart('This is the Header text',
'My BarchartTitle',
'Processors',
['processor1','processor2','processor3','processor4'],
'Time (seconds)',
[['setup time'],['sleep time'],['running time']],
[[20,30,40,50],[10,8,5,35],[60,100,20,50]],
'This is the Footer text').
```

Usage:

- The following properties should hold at call time:
  - Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - BTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - XTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - XVector is a list of xelements. (basic_props:list/2)
  - YTitle is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - BarsAtts is a list of multibar_attributes. (basic_props:list/2)
  - YVector is a list of yelements. (basic_props:list/2)
  - Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

multibarchart/10:

```
multibarchart(Header, BT, XT, XVector, YT, BAtts, YVector, YMax, YMin, Footer)
```

This predicate is quite similar to multibarchart/8, except in that you can choose limits in the y axis. The part of the bars placed outside the limits will not be plotted.

Example2:
multibarchart('This is the Header text',
'My BarchartTitle',
'Processors',
['processor1','processor2','processor3','processor4'],
'Time (seconds)',
[['setup time'],['sleep time'],['running time']],
[20,30,40,50],[10,8,5,35],[60,100,20,50],
80,
0,
'This is the Footer text').

Usage:

- The following properties should hold at call time:

  Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  BT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  XVector is a list of xelements. (basic_props:list/2)
  YT is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  BAtts is a list of multibar_attributes. (basic_props:list/2)
  YVector is a list of yelements. (basic_props:list/2)
  genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
  genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
  Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

192.3 Documentation on internals (genmultibar)

multibar_attribute/1:  REGTYPE
  multibar_attribute([LegendElement]) :-
      atomic(LegendElement).
  multibar_attribute([LegendElement,ForegroundColor,BackgroundColor,StipplePattern]) :-
      atom(LegendElement),
      color(ForegroundColor),
      color(BackgroundColor),
      pattern(StipplePattern).

Defines the attributes of each feature bar along the different datasets.

LegendElement
Legend element name. It may be a number or an atom. Every LegendElement value of the list must be unique.

ForegroundColor
It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

BackgroundColor
It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.
StipplePattern
    It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.

\[
\text{xelement/1:}
\]
\[
\text{xelement(Label) :-}
\]
\[
\quad \text{atomic(Label)}.
\]

This type defines a dataset label. Although Label values may be numbers, the will be treated as atoms, So it will be displayed at uniform intervals along the x axis.
193 table_widget1 (library)

Author(s): Isabel Martín García.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#166 (2003/12/4, 17:39:45 CET)

In addition to the features explained in the introduction, the predicates exported by this module depict tables in which the font weight for the table elements is bold.

If the arguments are not in a correct format the exception error8 will be thrown.

193.1 Usage and interface (table_widget1)

- Library usage:
  
  ```prolog
  :- use_module(library(table_widget1)).
  ```

- Exports:
  - Predicates:
    - tablewidget1/4, tablewidget1/5.
  - Regular Types:
    - table/1, image/1.

- Other modules used:
  - System library modules:
    - chartlib/genbar1, chartlib/bltclass, chartlib/test_format, chartlib/install_utils, lists.

193.2 Documentation on exports (table_widget1)

tablewidget1/4: PREDICATE

```prolog
tablewidget1(Title, Header, ElementTable, Footer)
```

Shows a regular table in a window. The user does not choose a background image.

Example:

```prolog
tablewidget1('This is the title',
  'Header text',
  [['Number of processors','8'],['Average processors','95'],
   ['Average Tasks per fork','7.5']],
  'Footer text').
```

Usage:

- The following properties should hold at call time:
  - Title is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - table_widget1:table(ElementTable) (table_widget1:table/1)
  - Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
**tablewidget1/5:**

**PREDICATE**

`tablewidget1(Title, Header, ElementTable, Footer, BackgroundImage)`

Shows a regular table in a window. The user must set a background image. See the `image/1` type definition.

Example:

```
  tablewidget1('This is the title',
               'Header text',
               [['Number of processors','8'],
                ['Average processors','95'],
                ['Average Tasks per fork','7.5']],
               'Footer text',
               './images/rain.gif')
```

**Usage:**

- The following properties should hold at call time:
  - **Title** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  - **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)
  - `table_widget1:table(ElementTable)` (table_widget1:table/1)
  - **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
  - `table_widget1:image(BackgroundImage)` (table_widget1:image/1)

**table/1:**

**REGTYPE**

A table is a list of rows, each row must contain the same number of elements, otherwise the table wouldn't be regular and an exception will be thrown by the library. The rows list may not be empty.

```
table([X]) :-
    row(X).
table([X|Xs]) :-
    row(X),
    table(Xs).
```

**image/1:**

**REGTYPE**

Some predicates allow the user to set the widget background image, whose is what this type is intended for. The user has to take into account the following restrictions:

- The image must be in gif format.
- The file path must be absolute.

193.3 Documentation on internals (**table_widget1**)

**row/1:**

**REGTYPE**

```
row([X]) :-
    cell_value(X).
row([X|Xs]) :-
    cell_value(X),
    row(Xs).
```

Each row is a list of elements whose type is `cell_value/1`. A row cannot be an empty list, as you can see in the definition type.
row/1:  
\[
\text{row}([X]) :- \\
\text{cell\_value}(X). \\
\text{row}([X|Xs]) :- \\
\text{cell\_value}(X), \\
\text{row}(Xs).
\]

Each row is a list of elements whose type is \text{cell\_value}/1. A row cannot be an empty list, as you can see in the definition type.

cell\_value/1:

This type defines the possible values that a table element have. If any cell value is ‘’’, the cell will be displayed empty.

\[
\text{cell\_value}(X) :- \\
\text{atomic}(X).
\]
194  table_widget2 (library)

**Author(s):** Isabel Martín García.

**Version:** 1.11#222 (2004/5/24, 13:8:7 CEST)

**Version of last change:** 1.9#167 (2003/12/4, 17:39:48 CET)

In addition to the features explained in the introduction, predicates exported by this module display tables in which the font weight for the elements placed in the first row is bold. The remaining elements are in medium weight font.

If the arguments are not in a correct format the exception error8 will be thrown.

194.1 Usage and interface (table_widget2)

- **Library usage:**
  
  ```prolog
  :- use_module(library(table_widget2)).
  ```

- **Exports:**
  
  **Predicates:**
  
  tablewidget2/4, tablewidget2/5.

- **Other modules used:**
  
  **System library modules:**
  
  chartlib/genbar1, chartlib/bltclass, chartlib/table_widget1, chartlib/test_format, chartlib/install_utils, lists.

194.2 Documentation on exports (table_widget2)

tablewidget2/4:

**Predicate**

```
tablewidget2(Title, Header, ElementTable, Footer)
```

Shows a regular table in a window. The system sets a default background image for the widget.

Example:

```
tablewidget2('COM Features',
             'Extracted from "Inside COM" book ',
             [['Feature','Rich people','Bean Plants','C++','COM'],
              ['Edible','Yes','Yes','No','No'],
              ['Supports inheritance','Yes','Yes','Yes','Yes and No'],
              ['Can run for President','Yes','No','No','No']],
             'What do you think about COM?').
```

**Usage:**

- The following properties should hold at call time:
  
  **Title** is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  **Header** is a text (an atom) describing the header of the graph. (genbar1:header/1)

  **table_widget1:table(ElementTable)** (table_widget1:table/1)

  **Footer** is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
tablewidget2/5:

predicate

\[
\text{tablewidget2(Title, Header, ElementTable, Footer, BackgroundImage)}
\]

This predicate and \text{tablewidget2/4} are quite similar, except that in the already one defined you must set the background image.

Example:

\[
\text{tablewidget2('COM Features', 'Extracted from "Inside COM" book', [['Feature','Rich people','Bean Plants','C++','COM'], ['Edible','Yes','Yes','No','No'], ['Supports inheritance','Yes','Yes','Yes','Yes and No'], ['Can run for President','Yes','No','No','No']], 'What do you think about COM?', './images/rain.gif').}
\]

Usage:

\text{The following properties should hold at call time:}

- \text{Title} is a text (an atom) to be used as label, usually not very long. 
  \text{(genbar1:title/1)}
- \text{Header} is a text (an atom) describing the header of the graph. \text{(genbar1:header/1)}
- \text{table_widget1:table(ElementTable)} \text{(table_widget1:table/1)}
- \text{Footer} is a text (an atom) describing the footer of the graph. \text{(genbar1:footer/1)}
- \text{table_widget1:image(BackgroundImage)} \text{(table_widget1:image/1)}
Chapter 195: table_widget3 (library)

Author(s): Isabel Martín García.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#168 (2003/12/4, 17:39:51 CET)

The predicates exported by this module display data in a regular table, as we brought up in
the introduction. Both predicates have in common that the font weight for the elements placed
in the first column is bold and the remaining elements are in medium font weight.

If the arguments are not in a correct format the exception error8 will be thrown.

195.1 Usage and interface (table_widget3)

- Library usage:
  :- use_module(library(table_widget3)).
- Exports:
  - Predicates:
tablewidget3/4, tablewidget3/5.
- Other modules used:
  - System library modules:
    chartlib/genbar1, chartlib/bltclass, chartlib/table_widget1,
    chartlib/test_format, chartlib/install_utils, lists.

195.2 Documentation on exports (table_widget3)

tablewidget3/4: PREDICATE

  tablewidget3(Title, Header, ElementTable, Footer)

  Shows a regular table in a window. The user does not choose a background image.

  Example:
  
  tablewidget3('This is the title',
               'Header text',
               [['Number of processors','8'],['Average processors','95'],
                ['Tasks per fork','7.5']],
               'Footer text').

  Usage:
  - The following properties should hold at call time:

    Title is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
    Header is a text (an atom) describing the header of the graph. (genbar1:header/1)
    table_widget1:table(ElementTable) (table_widget1:table/1)
    Footer is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
tablewidget3/5:  

\[ \text{tablewidget3}(\text{Title}, \text{Header}, \text{ElementTable}, \text{Footer}, \text{BackgroundImage}) \]

Shows a regular table in a window. The user must set a background image.

Example:

\[
\text{tablewidget3}(\text{'This is the title'}, \\
\text{'Header text'}, \\
[\text{['Number of processors','8']},\text{['Average processors','95']}, \\
\text{[['Average Tasks per fork','7.5']]}], \\
\text{'Footer text'}, \\
\text{'./images/rain.gif'})
\]
Chapter 196: table_widget4 (library)

Author(s): Isabel Martín García.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#169 (2003/12/4, 17:39:54 CET)

In addition to the features explained in the introduction, predicates exported by this module display tables in which the font weight for the elements placed in the first row and column is bold. The remaining elements are in medium weight font.

If the arguments are not in a correct format the exception error8 will be thrown.

196.1 Usage and interface (table_widget4)

- Library usage:
  
  :- use_module(library(table_widget4)).

- Exports:
  - Predicates:
    
    tablewidget4/4, tablewidget4/5.

- Other modules used:
  - System library modules:
    
    chartlib/genbar1, chartlib/bltclass, chartlib/table_widget1,
    chartlib/test_format, chartlib/install_utils, lists.

196.2 Documentation on exports (table_widget4)

**tablewidget4/4:**

**PREDICATE**

**tablewidget4(Title, Header, ElementTable, Footer)**

Shows a regular table in a window. The system sets a default background image for the widget.

Example:

```
   tablewidget4('Some sterEUtypes',
     'Source: Eurostat yearbook, 1999',
     [['Country','Adult alcohol intake per year (litres)',
       'Cigarettes smoked per day per adult',
       'Suicides per 100000 people'],
      ['Finland','8.4','2.2','26.3'],
      ['Spain','11.4','5.3','7.5'],
      ['Austria','11.9','4.6','20.7'],
      ['Britain','9.4','4.2','7.1'],
      ['USA','4.7','4.9','13'],
      ['European Union','11.1','4.5','11.9']]
     ,
     'This is part of the published table').
```

**Usage:**

- The following properties should hold at call time:
  - **Title** is a text (an atom) to be used as label, usually not very long. (**genbar1:title/1**)
  - **Header** is a text (an atom) describing the header of the graph. (**genbar1:header/1**)
  - **table_widget1:table(ElementTable)** (**table_widget1:table/1**)
  - **Footer** is a text (an atom) describing the footer of the graph. (**genbar1:footer/1**)
tablewidget4/5: PREDICATE

\[ \text{tablewidget4(Title, Header, ElementTable, Footer, BackgroundImage)} \]

This predicate and \text{tablewidget4/4} are comparable, except that in the already defined you must set the background image.

Example:

\[
\text{tablewidget4('Some sterEUtypes', 'Source: Eurostat yearbook, 1999', }
\\text{[['Country','Adult alcohol intake per year (litres)',}
\\text{'Cigarettes smoked per day per adult',}
\\text{'Suicides per 100000 people']},
\\text{[['Finland','8.4','2.2','26.3'],[['Spain','11.4','5.3','7.5'],
\\text{[['Austria','11.9','4.6','20.7'],[['Britain','9.4','4.2','7.1'],
\\text{[['USA','4.7','4.9','13'],[['European Union','11.1','4.5','11.9']},
\\text{'This is part of the published table',}
\\text{'./images/rain.gif'}).\}
\]

Usage:

- \text{The following properties should hold at call time:}

  \begin{itemize}
  \item \text{Title} is a text (an atom) to be used as label, usually not very long. (\text{genbar1:title/1})
  \item \text{Header} is a text (an atom) describing the header of the graph. (\text{genbar1:header/1})
  \item \text{table_widget1:table(ElementTable)} (\text{table_widget1:table/1})
  \item \text{Footer} is a text (an atom) describing the footer of the graph. (\text{genbar1:footer/1})
  \item \text{table_widget1:image(BackgroundImage)} (\text{table_widget1:image/1})
  \end{itemize}
197 test_format (library)

Author(s): Isabel Martín García.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#170 (2003/12/4, 17:39:57 CET)

Most of the predicates exported by this module perform some checks to determine whether the arguments attain some conditions or not. In the second case an exception will be thrown. To catch the exceptions you can use the following metapredicates when invoking chartlib exported predicates:

- chartlib_text_error_protect/1
- chartlib_text_error_protect/1

Both metapredicates are defined in the chartlib.errhandle module that comes with this library. Some of the predicates have a Predicate argument which will be used in case of error to show which chartlib predicate causes the error.

197.1 Usage and interface (test_format)

- Library usage:
  ```prolog
  :- use_module(library(test_format)).
  ```
- Exports:
  
  **Predicates:**
  
  - equalnumber/3, not_empty/4, not_empty/3, check_sublist/4, valid_format/4, vectors_format/4, valid_vectors/4, valid_attributes/2, valid_table/2.

- Other modules used:
  
  **System library modules:**
  
  - chartlib/bltclass, lists.

197.2 Documentation on exports (test_format)

equalnumber/3:

```
monic number(X, Y, Predicate)
```

Test whether the list X and the list Y contain the same number of elements.

Usage:

- The following properties should hold at call time:
  
  X is a list.
  
  Y is a list.
  
  Predicate is a term which represents a goal, i.e., an atom or a structure.

not_empty/4:

```
monic empty(X, Y, Z, Predicate)
```

Tests whether at least one the lists X, Y or Z are empty.

Usage:
The following properties should hold at call time:

- X is a list. (basic_props:list/1)
- Y is a list. (basic_props:list/1)
- Z is a list. (basic_props:list/1)
- Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

**not_empty/3:**

```prolog
not_empty(X, Y, Predicate)
```
Tests whether the lists X or Y are empty.

**Usage:**

- The following properties should hold at call time:
  - X is a list. (basic_props:list/1)
  - Y is a list. (basic_props:list/1)
  - Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

**check_sublist/4:**

```prolog
check_sublist(List, Number, Number, Predicate)
```
Tests if the number of elements in each sublist of List is Number1 or Number2.

**Usage:**

- The following properties should hold at call time:
  - List is a list. (basic_props:list/1)
  - Number is currently instantiated to an integer. (term_typing:integer/1)
  - Number is currently instantiated to an integer. (term_typing:integer/1)
  - Predicate is an atom. (basic_props:atm/1)

**valid_format/4:**

```prolog
valid_format(XVector, YVector, BarsAttributes, Predicate)
```
Tests the following restrictions:

- The XVector number of elements is the same as each YVector sublist number of elements.
- The YVector length is equal to BarsAttributes length.

**Usage:**

- The following properties should hold at call time:
  - XVector is a list. (basic_props:list/1)
  - YVector is a list. (basic_props:list/1)
  - BarsAttributes is a list. (basic_props:list/1)
  - Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
vectors_format/4: PREDICATE
vectors_format(XVector, YVectors, LinesAttributes, Predicate)
Tests the following conditions:
• YVectors list and LinesAttributes list have the same number of elements.
• XVector list and each YVectors element have the same number of elements.
• Each sublist of LinesAttributes is composed of 5, 3 or 1 elements.
Usage:
  - The following properties should hold at call time:
    XVector is a list. (basic_props:list/1)
    YVectors is a list. (basic_props:list/1)
    LinesAttributes is a list. (basic_props:list/1)
    Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

valid_vectors/4: PREDICATE
valid_vectors(XVector, YVectors, LinesAttributes, Predicate)
Tests the following conditions:
• XVector list, YVectors list and LinesAttributes list have the same number of elements.
• Each sublist of LinesAttributes is composed of 5, 3 or 1 element.
Usage:
  - The following properties should hold at call time:
    XVector is a list. (basic_props:list/1)
    YVectors is a list. (basic_props:list/1)
    LinesAttributes is a list. (basic_props:list/1)
    Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

valid_attributes/2: PREDICATE
valid_attributes(BarsAttibuttes, Predicate)
Check if each BarsAttibuttes element is a list composed of one or four elements.
Usage:
  - The following properties should hold at call time:
    BarsAttibuttes is a list. (basic_props:list/1)
    Predicate is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

valid_table/2: PREDICATE
valid_table(ElementTable, Predicate)
All of the ElementTable sublists have the same number of elements and are not empty.
Usage:
The following properties should hold at call time:

ElementTable is a list. \hfill (basic.props:list/1)

Predicate is a term which represents a goal, i.e., an atom or a structure. \hfill (basic.props:callable/1)
198 ProVRML - a Prolog interface for VRML

Author(s): Göran Smedbäck, (Some changes by Manuel Carro), clip@dia.fi.upm.es, http://www.clip.dia.fi.upm.es/, The CLIP Group, Facultad de Informática, Universidad Politécnica de Madrid.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 0.1#1 (1998/12/10, 16:19:45 MET)

ProVRML is Prolog library to handle VRML code. The library consists of modules to handle the tokenising, that is breaking the VRML code into smaller parts that can be analysed further. The further analysis will be the parsing. This is a complex part of the library and consists of several modules to handle errors and value check. When the parsing is done we have the Prolog terms of the VRML code. The terms are quite similar to the origin VRML code and can easily be read if you recognise that syntax.

This Prolog terms of the VRML code is then possible to use for analysis, reconstruction, reverse engineering, building blocks for automatic generation of VRML code. There are several possibilities and these are only some of them.

When you are done with the Prolog terms for the code, you would probably want to reverse the action and return to VRML code. This is done with the code generation modules. These are built up in more or less the same manner as the parser modules.

198.1 Usage and interface (provrml)

- Library usage:
  ```prolog```
  :- use_module(library(provrml)).

- Exports:
  - Predicates:
    ```prolog```
    vrml_web_to_terms/2, vrml_file_to_terms/2, vrml_web_to_terms_file/2, vrml_file_to_terms_file/2, terms_file_to_vrml/2, terms_file_to_vrml_file/2, terms_to_vrml/2, terms_to_vrml_file/2, vrml_to_terms/2, vrml_to_terms_file/2, vrml_http_access/2.

- Other modules used:
  - System library modules:
    ```prolog```

198.2 Documentation on exports (provrml)

vrml_web_to_terms/2:

```prolog```
vrml_web_to_terms(WEBAddress, Terms)

- Description: Given a address to a VRML-document on the Internet, the predicate will return the prolog-terms.
- Call and exit should be compatible with:
  ```prolog```
  WEBAddress is an atom. (basic_props:atm/1)
  Terms is a string (a list of character codes). (basic_props:string/1)
The following properties should hold at call time:

WEBAddress is currently a term which is not a free variable. \(\text{(term_typing:nonvar/1)}\)

Term is a free variable. \(\text{(term_typing:var/1)}\)


\begin{verbatim}
vrml_file_to_terms/2:
Usage 1: vrml_file_to_terms(FileName, Term)
- Description: Given a filename containing a VRML-file the predicate returns the prolog terms corresponding.
- Call and exit should be compatible with:
  FileName is an atom. \(\text{(basic_props:atm/1)}\)
  Term is an atom. \(\text{(basic_props:atm/1)}\)
- The following properties should hold at call time:
  FileName is currently a term which is not a free variable. \(\text{(term_typing:nonvar/1)}\)
  Term is a free variable. \(\text{(term_typing:var/1)}\)

Usage 2: vrml_file_to_terms(FileName, Terms)
- Description: Given a filename containing a VRML-file and a filename, the predicate write the prolog terms corresponding to the filename.
- Call and exit should be compatible with:
  FileName is an atom. \(\text{(basic_props:atm/1)}\)
  Terms is an atom. \(\text{(basic_props:atm/1)}\)
- The following properties should hold at call time:
  FileName is currently a term which is not a free variable. \(\text{(term_typing:nonvar/1)}\)
  Terms is currently a term which is not a free variable. \(\text{(term_typing:nonvar/1)}\)
\end{verbatim}


\begin{verbatim}
vrml_web_to_terms_file/2:
Usage: vrml_web_to_terms_file(WEBAddress, FileName)
- Description: Given a address to a VRML-document on the Internet and a filename, the predicate will write the prolog terms to the file.
- Call and exit should be compatible with:
  WEBAddress is an atom. \(\text{(basic_props:atm/1)}\)
  FileName is an atom. \(\text{(basic_props:atm/1)}\)
- The following properties should hold at call time:
  WEBAddress is currently a term which is not a free variable. \(\text{(term_typing:nonvar/1)}\)
  FileName is currently a term which is not a free variable. \(\text{(term_typing:nonvar/1)}\)
\end{verbatim}


\begin{verbatim}
vrml_file_to_terms_file/2:
No further documentation available for this predicate.
\end{verbatim}


\begin{verbatim}
terms_file_to_vrml/2:
Usage: terms_file_to_vrml(FileName, List)
\end{verbatim}
Description: From a given filename with prologterms on the special format, the predicate returns the corresponding VRML-code.

- Call and exit should be compatible with:
  - FileName is an atom. (basic_props:atm/1)
  - List is a string (a list of character codes). (basic_props:string/1)

- The following properties should hold at call time:
  - FileName is currently a term which is not a free variable. (term_typing:nonvar/1)
  - List is a free variable. (term_typing:var/1)

PREDICATE terms_file_to_vrml_file/2:
Usage: terms_file_to_vrml_file(Atom, Atom)
- Description: From a given filename with prologterms on the special format, the predicate writes the corresponding VRML-code to second filename.
- Call and exit should be compatible with:
  - Atom is an atom. (basic_props:atm/1)
  - Atom is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  - Atom is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Atom is currently a term which is not a free variable. (term_typing:nonvar/1)

PREDICATE terms_to_vrml_file/2:
Usage: terms_to_vrml_file(Term, FileName)
- Description: Given prolog-terms the predicate writes the corresponding VRML-code to the given file.
- Call and exit should be compatible with:
  - Term is an atom. (basic_props:atm/1)
  - FileName is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  - Term is currently a term which is not a free variable. (term_typing:nonvar/1)
  - FileName is currently a term which is not a free variable. (term_typing:nonvar/1)

PREDICATE terms_to_vrml/2:
Usage: terms_to_vrml(Term, VRMLCode)
- Description: Given prolog-terms the predicate returns a list with the corresponding VRML-code.
- Call and exit should be compatible with:
  - Term is an atom. (basic_props:atm/1)
  - VRMLCode is a string (a list of character codes). (basic_props:string/1)
- The following properties should hold at call time:
  - Term is currently a term which is not a free variable. (term_typing:nonvar/1)
  - VRMLCode is a free variable. (term_typing:var/1)
vrml_to_terms/2:  

Usage: vrml_to_terms(VRMLCode, Terms)

- Description: Given a list with VRML-code the predicate will return the corresponding prolog-terms.

Call and exit should be compatible with:

VRMLCode is a string (a list of character codes). (basic_props:string/1)
Terms is an atom. (basic_props:atm/1)

The following properties should hold at call time:

VRMLCode is currently a term which is not a free variable. (term_typing:nonvar/1)
Terms is a free variable. (term_typing:var/1)

vrml_in_out/2:  

Usage: vrml_in_out(FileName, FileName)

- Description: This is a controll-predicate that given a filename to a VRML-file and a filename, the predicate will read the VRML-code. Transform it to prolog-terms and then transform it back to VRML-code and write it to the latter file.

Call and exit should be compatible with:

FileName is an atom. (basic_props:atm/1)
FileName is an atom. (basic_props:atm/1)

The following properties should hold at call time:

FileName is currently a term which is not a free variable. (term_typing:nonvar/1)
FileName is currently a term which is not a free variable. (term_typing:nonvar/1)

vrml_http_access/2:  

Usage: vrml_http_access(ReadFilename, BaseFilename)

- Description: Given a web-address to a VRML-file the predicate will load the code, write it first to the second argument with extension '_first.wrl'. Then it transform the code to prolog terms and write it with the extension '.term'. Transform it back to VRML-code and write it to the filename with '.wrl. A good test-predicate.

Call and exit should be compatible with:

ReadFilename is an atom. (basic_props:atm/1)
BaseFilename is an atom. (basic_props:atm/1)

The following properties should hold at call time:

ReadFilename is currently a term which is not a free variable. (term_typing:nonvar/1)
BaseFilename is currently a term which is not a free variable. (term_typing:nonvar/1)

198.3 Documentation on internals (provrml)

read_page/2:  

Usage: read_page(WEBAddress, Data)

- Description: This routine reads a page on the web using pillow routines.
Call and exit should be compatible with:

WEBAddress is an atom. (basic_props:atm/1)

Data is a string (a list of character codes). (basic_props:string/1)

The following properties should hold at call time:

WEBAddress is currently a term which is not a free variable. (term_typing:nonvar/1)

Data is a free variable. (term_typing:var/1)
199 boundary (library)

Author(s): Göran Smedbäck.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.9#171 (2003/12/4, 17:46:50 CET)

This module offers predicate to check values according to their boundaries and offers lists of possible node ascendents.

199.1 Usage and interface (boundary)

Library usage:
:- use_module(library(boundary)).

Exports:
- Predicates:
  - boundary_check/3, boundary_rotation_first/2, boundary_rotation_last/2,
  - reserved_words/1, children_nodes/1.

Other modules used:
- System library modules:
  - aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read,
  - write, provrml/internal_types, provrml/error.

199.2 Documentation on exports (boundary)

boundary_check/3:
Usage: boundary_check(Value_to_check, Init_value, Bound)
- Description: This predicate check the boundaries of the given value according to the boundary values. If the value is wrong according to the boundaries, the value is checked according to the initial value given. If the value continues to be wrong, an error will be raised accordingly.
- Call and exit should be compatible with:
  Value_to_check is an atom. (basic_props:atm/1)
  Init_value is a list of atms. (basic_props:list/2)
  Bound is a variable interval. (internal_types:bound/1)
- The following properties should hold at call time:
  Value_to_check is currently a term which is not a free variable. (term_typing:nonvar/1)
  Init_value is currently a term which is not a free variable. (term_typing:nonvar/1)
  Bound is currently a term which is not a free variable. (term_typing:nonvar/1)

boundary_rotation_first/2:
Usage: boundary_rotation_first(Bound_double, Bound)
- Description: The predicate will extract the first bounds from a double bound.
– *Call and exit should be compatible with:*
  Bound_double is a variable interval. (internal_types:bound_double/1)
  Bound is a variable interval. (internal_types:bound/1)

– *The following properties should hold at call time:*
  Bound_double is currently a term which is not a free variable. (term_typing:nonvar/1)
  Bound is a free variable. (term_typing:var/1)

**boundary_rotation_last/2:**

**Usage:** boundary_rotation_last(Bound_double, Bound)

– *Description:* The predicate will extract the last bounds from a double bound.

– *Call and exit should be compatible with:*
  Bound_double is a variable interval. (internal_types:bound_double/1)
  Bound is a variable interval. (internal_types:bound/1)

– *The following properties should hold at call time:*
  Bound_double is currently a term which is not a free variable. (term_typing:nonvar/1)
  Bound is a free variable. (term_typing:var/1)

**reserved_words/1:**

**Usage:** reserved_words(List)

– *Description:* Returns a list with the reserved words, words prohibited to use in cases not appropriated.

– *Call and exit should be compatible with:*
  List is a list of atms. (basic_props:list/2)

– *The following properties should hold at call time:*
  List is a free variable. (term_typing:var/1)

**children_nodes/1:**

**Usage:** children_nodes(List)

– *Description:* Returns a list of all nodes possible as children nodes.

– *Call and exit should be compatible with:*
  List is a list of atms. (basic_props:list/2)

– *The following properties should hold at call time:*
  List is a free variable. (term_typing:var/1)
200 dictionary (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#172 (2003/12/4, 17:47:7 CET)

This module contains the fixed dictionary. All the nodes in VRML with their associated fields.

200.1 Usage and interface (dictionary)

- Library usage:
  :- use_module(library(dictionary)).
- Exports:
  - Predicates:
    dictionary/6.
- Other modules used:
  - System library modules:
    aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read, write, lists, provrml/internal_types.

200.2 Documentation on exports (dictionary)

dictionary/6: PREDICATE

Usage 1: dictionary(NodeTypeId, AccessType, FieldTypeId, FieldId, Init_value, Boundary)

  Description: To lookup information about the nodes, getting their properties. Note that the type returned for the bound can be of two different types bound or bound_double. The rotation type have one bound for the directions and one for the degree of rotation.

  Call and exit should be compatible with:
  - NodeTypeId is an atom. (basic_props:atm/1)
  - AccessType is an atom. (basic_props:atm/1)
  - FieldTypeId is an atom. (basic_props:atm/1)
  - FieldId is an atom. (basic_props:atm/1)
  - Init_value is a list of atms. (basic_props:list/2)
  - Boundary is a variable interval. (internal_types:bound/1)

  The following properties should hold at call time:
  - Init_value is a free variable. (term_typing:var/1)
  - Boundary is a free variable. (term_typing:var/1)

Usage 2: dictionary(NodeTypeId, AccessType, FieldTypeId, FieldId, Init_value, Boundary)
Description: To lookup information about the nodes, getting their properties. Note that the type returned for the bound can be of two different types bound or bound_double. The rotation type have one bound for the directions and one for the degree of rotation.

Call and exit should be compatible with:

- NodeType is an atom. (basic_props:atm/1)
- AccessType is an atom. (basic_props:atm/1)
- FieldType is an atom. (basic_props:atm/1)
- FieldId is an atom. (basic_props:atm/1)
- Init_value is a list of atms. (basic_props:list/2)
- Boundary is a variable interval. (internal_types:bound_double/1)

The following properties should hold at call time:

- Init_value is a free variable. (term_typing:var/1)
- Boundary is a free variable. (term_typing:var/1)
201 dictionary_tree (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#173 (2003/12/4, 17:47:16 CET)

This module offers a dynamic tree structured dictionary a bit combined with predicates that gives it the usability to be the dictionary for the parser.

201.1 Usage and interface (dictionary_tree)

- Library usage:
  `:- use_module(library(dictionary_tree)).`
- Exports:
  - Predicates:
    `create_dictionaries/1`, `is_dictionaries/1`, `get_definition_dictionary/2`, `get_prototype_dictionary/2`, `dictionary_insert/5`, `dictionary_lookup/5`, `merge_tree/2`.
- Other modules used:
  - System library modules:
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `lists`, `provrml/internal_types`.

201.2 Documentation on exports (dictionary_tree)

create_dictionaries/1:

Usage: `create_dictionaries(Dictionary)`
- Description: Returns a dictionary. A general name was used if the user would like to change the code to include more dictionaries.
- Call and exit should be compatible with:
  Dictionary is a dictionary. (`internal_types:dictionary/1`)
- The following properties should hold at call time:
  Dictionary is a free variable. (`term_typing:var/1`)

is_dictionaries/1:

Usage: `is_dictionaries(Dictionary)`
- Description: Is the argument a dictionary is solved by this predicate.
- Call and exit should be compatible with:
  Dictionary is a dictionary. (`internal_types:dictionary/1`)
**get_definition_dictionary/2:**

Usage: `get_definition_dictionary(Dictionary, Tree)`

- Description: Returns the definition dictionary (for the moment there is only one dictionary), which is a tree representation.
- Call and exit should be compatible with:
  - `Dictionary` is a dictionary. (internal_types:dictionary/1)
  - `Tree` is a tree structure. (internal_types:tree/1)
- The following properties should hold at call time:
  - `Dictionary` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Tree` is a free variable. (term_typing:var/1)

**get_prototype_dictionary/2:**

Usage: `get_prototype_dictionary(Dictionary, Tree)`

- Description: Returns the prototype dictionary (for the moment there is only one dictionary), which is a tree representation.
- Call and exit should be compatible with:
  - `Dictionary` is a dictionary. (internal_types:dictionary/1)
  - `Tree` is a tree structure. (internal_types:tree/1)
- The following properties should hold at call time:
  - `Dictionary` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Tree` is a free variable. (term_typing:var/1)

**dictionary_insert/5:**

Usage: `dictionary_insert(Key, Type, Field, Dictionary, Info)`

- Description: The predicate will search for the place for the Key and return Info, if the element inserted had a post before (same key value) multiple else new. The dictionary is dynamic and do not need output because of using unbinded variables.
- Call and exit should be compatible with:
  - `Key` is an atom. (basic_props:atm/1)
  - `Type` is an atom. (basic_props:atm/1)
  - `Field` is any term. (basic_props:term/1)
  - `Dictionary` is a tree structure. (internal_types:tree/1)
  - `Info` is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  - `Key` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Type` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Field` is currently a term which is not a free variable. (term_typing:nonvar/1)
  - `Dictionary` is currently a term which is not a free variable. (term_typing:nonvar/1)

**dictionary_lookup/5:**

Usage: `dictionary_lookup(Key, Type, Field, Dictionary, Info)`
Description: The predicate will search for the Key and return Info: defined or undefined accordingly. If defined the fields will be filled as well. The predicate do not insert the element.

Call and exit should be compatible with:
- Key is an atom. (basic_props:atm/1)
- Type is an atom. (basic_props:atm/1)
- Field is any term. (basic_props:term/1)
- Dictionary is a dictionary. (internal_types:dictionary/1)
- Info is an atom. (basic_props:atm/1)

The following properties should hold at call time:
- Key is currently a term which is not a free variable. (term_typing:nonvar/1)
- Dictionary is currently a term which is not a free variable. (term_typing:nonvar/1)
- Info is a free variable. (term_typing:var/1)

merge_tree/2:

Usage: merge_tree(Tree, Tree)

Description: The predicate can be used for adding a tree dictionary to another one (the second). It will remove equal posts but posts with a slight difference will be inserted. The resulting tree will be the second tree.

Call and exit should be compatible with:
- Tree is a tree structure. (internal_types:tree/1)
- Tree is a tree structure. (internal_types:tree/1)

The following properties should hold at call time:
- Tree is currently a term which is not a free variable. (term_typing:nonvar/1)
- Tree is currently a term which is not a free variable. (term_typing:nonvar/1)
Chapter 202: error (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#179 (2003/12/4, 19:25:17 CET)
This file implements error predicates of different types.

202.1 Usage and interface (error)

- Library usage:
  :- use_module(library(error)).
- Exports:
  - Predicates:
    error_vrml/1, output_error/1.
- Other modules used:
  - System library modules:
    write.

202.2 Documentation on exports (error)

error_vrml/1:  
Usage: error_vrml(Structure)
- Description: Given a structure with the error type as its head with possible arguments, it will write the associated error-text.
- Call and exit should be compatible with:
  Structure is any term. (basic_props:term/1)
- The following properties should hold at call time:
  Structure is currently a term which is not a free variable. (term_typing:nonvar/1)

output_error/1:  
Usage: output_error(Message)
- Description: This predicate will print the error message given as the argument. This predicate is used for warnings that only needs to be given as information and not necessarily give an error by the VRML browser.
- Call and exit should be compatible with:
  Message is a list of atoms. (basic_props:list/2)
- The following properties should hold at call time:
  Message is currently a term which is not a free variable. (term_typing:nonvar/1)
203 field_type (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#174 (2003/12/4, 17:47:35 CET)

203.1 Usage and interface (field_type)

- Library usage:
  :- use_module(library(field_type)).
- Exports:
  - Predicates:
    fieldType/1.

203.2 Documentation on exports (field_type)

fieldType/1: PREDICATE
  Usage: fieldType(FieldTypeId)
  - Description: Boolean predicate used to check the FieldTypeId with the defined.
  - Call and exit should be compatible with:
    FieldTypeId is an atom. (basic_props:atm/1)
  - The following properties should hold at call time:
    FieldTypeId is currently a term which is not a free variable. (term_typing:nonvar/1)
204 field_value (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#175 (2003/12/4, 17:47:48 CET)

204.1 Usage and interface (field_value)

- Library usage:
  
  ```prolog
  :- use_module(library(field_value)).
  ```

- Exports:
  - Predicates:
    ```prolog
    fieldValue/6, mfstringValue/5.
    ```
  - Properties:
    ```prolog
    parse/1.
    ```

- Other modules used:
  - System library modules:
    ```prolog
    lists, provrml/parser, provrml/parser_util, provrml/error.
    ```

204.2 Documentation on exports (field_value)

**fieldValue/6:**

No further documentation available for this predicate.

**mfstringValue/5:**

No further documentation available for this predicate.

**parse/1:**

A property, defined as follows:

```prolog
parse(_1).
parse(parse(In,Out,Env,Dic)) :-
  list(In),
  list(Out),
  environment(Env),
  dictionary(Dic).
```
205 field_value_check (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#176 (2003/12/4, 17:47:53 CET)

205.1 Usage and interface (field_value_check)

- Library usage:
  :- use_module(library(field_value_check)).

- Exports:
  - Predicates:
    fieldValue_check/8, mfstringValue/7.

- Other modules used:
  - System library modules:
    lists, provrml/io, provrml/generator_util, provrml/boundary, provrml/tokeniser, provrml/generator, provrml/parser_util.

205.2 Documentation on exports (field_value_check)

fieldValue_check/8:
No further documentation available for this predicate.

mfstringValue/7:
No further documentation available for this predicate.
206 generator (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

206.1 Usage and interface (generator)

- Library usage:
  \texttt{:- use\_module(library(generator)).}

- Exports:
  - 	exttt{Predicates:}
  - 	exttt{generator/2, nodeDeclaration/4.}

- Other modules used:
  - 	exttt{System library modules:}
    - 	exttt{aggregates, dynamic, iso\_misc, iso\_byte\_char, iso\_incomplete, operators, read, write, provrml\_lookup, provrml\_io, provrml\_generator\_util, provrml\_parser\_util, provrml\_error, provrml\_internal\_types.}

206.2 Documentation on exports (generator)

**generator/2:**
Usage: \texttt{generator(Terms, VRML)}

- Description: This predicate is the generator of VRML code. It accepts a list of terms that is correct VRML code, other kind of terms will be rejected with an error message accordingly. The output is a string of correct VRML code, acceptable for VRML browsers.
- Call and exit should be compatible with:
  - \texttt{Terms} is a list of \texttt{terms}.
  - \texttt{VRML} is a string (a list of character codes).
- The following properties should hold at call time:
  - \texttt{Terms} is currently a term which is not a free variable.
  - \texttt{VRML} is a free variable.

**nodeDeclaration/4:**
No further documentation available for this predicate.
207 generator_util (library)

Author(s): Göran Smedbäck.

207.1 Usage and interface (generator_util)

- **Library usage:**
  
  ```prolog
  :- use_module(library(generator_util)).
  ```

- **Exports:**
  
  - **Predicates:**
    
    - reading/4, reading/5, reading/6, open_node/6, close_node/5, close_nodeGut/4, open_PROTO/4, close_PROTO/6, open_EXTERNPROTO/5, close_EXTERNPROTO/6, open_DEF/5, close_DEF/5, open_Script/5, close_Script/5, decompose_field/3, indentation_list/2, start_vrmlScene/4, remove_comments/4.

  - **Other modules used:**

    - **System library modules:**
      
      - provrml/error, lists, provrml/io, provrml/field_value, provrml/field_value_check, provrml/lookup, provrml/parser_util.

207.2 Documentation on exports (generator_util)

**reading/4:**

**Usage 1:** reading(IS, NodeTypeId, ParseIn, ParseOut)

- **Description:** This predicate will refer to a formerly introduced interface. We do a checkup of the access type and output the values.

- **Call and exit should be compatible with:**

  - IS is an atom. (basic_props:atm/1)
  - NodeTypeId is an atom. (basic_props:atm/1)
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)

- **The following properties should hold at call time:**

  - IS is currently a term which is not a free variable. (term_typing:nonvar/1)
  - NodeTypeId is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseOut is a free variable. (term_typing:var/1)

**Usage 2:** reading(NodeGut, NodeName, ParseIn, ParseOut)

- **Description:** This predicate will read a node gut and will check the field according to the name.

- **Call and exit should be compatible with:**

  - NodeGut is an atom. (basic_props:atm/1)
  - NodeName is an atom. (basic_props:atm/1)
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)
The following properties should hold at call time:

- NodeGut is currently a term which is not a free variable. (term_typing:nonvar/1)
- NodeName is currently a term which is not a free variable. (term_typing:nonvar/1)
- ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
- ParseOut is a free variable. (term_typing:var/1)

**reading/5:**

No further documentation available for this predicate.

**reading/6:**

No further documentation available for this predicate.

**open_node/6:**

No further documentation available for this predicate.

**close_node/5:**

No further documentation available for this predicate.

**close_nodeGut/4:**

No further documentation available for this predicate.

**openPROTO/4:**

No further documentation available for this predicate.

**closePROTO/6:**

No further documentation available for this predicate.

**open_EXTERNPROTO/5:**

No further documentation available for this predicate.

**close_EXTERNPROTO/6:**

No further documentation available for this predicate.

**open_DEF/5:**

No further documentation available for this predicate.
**close_DEF/5:**
No further documentation available for this predicate.

**open_Script/5:**
No further documentation available for this predicate.

**close_Script/5:**
No further documentation available for this predicate.

**decompose_field/3:**
No further documentation available for this predicate.

**indentation_list/2:**
Usage: indentation_list(Parse, IndList)

- **Description:** This predicate will construct a list with indentations to be output before text. The information of the indentations is inside the parse structure.
- **Call and exit should be compatible with:**
  Parse is a parse structure. (internal_types:parse/1)
  IndList is a list of atms. (basic_props:list/2)
- **The following properties should hold at call time:**
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  IndList is a free variable. (term_typing:var/1)

**start_vrmlScene/4:**
No further documentation available for this predicate.

**remove_comments/4:**
Usage: remove_comments(Value, CommentsBefore, ValueClean, CommentsAfter)

- **Description:** The predicate will remove comments and return the comments before and after the pure value.
- **Call and exit should be compatible with:**
  Value is a list of atms. (basic_props:list/2)
  CommentsBefore is a list of atms. (basic_props:list/2)
  ValueClean is an atom. (basic_props:atm/1)
  CommentsAfter is a list of atms. (basic_props:list/2)
- **The following properties should hold at call time:**
  Value is currently a term which is not a free variable. (term_typing:nonvar/1)
  CommentsBefore is a free variable. (term_typing:var/1)
  ValueClean is a free variable. (term_typing:var/1)
  CommentsAfter is a free variable. (term_typing:var/1)
The Ciao Prolog System
208 internal_types (library)

Author(s): Göran Smedbäck.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#177 (2003/12/4, 17:48:37 CET)

These are the internal data types used in the predicates. They are only used to simplify this documentation and make it more understandable.

Implemented by Göran Smedbäck

208.1 Usage and interface (internal_types)

- Library usage:
  :- use_module(library(internal_types)).
- Exports:
  - Regular Types:
    bound/1, bound_double/1, dictionary/1, environment/1, parse/1, tree/1, whitespace/1.

208.2 Documentation on exports (internal_types)

bound/1: REGTYPE
Min is a number or an atom that indicates the minimal value, Max indicates the maximal.

bound(bound(Min,Max)) :-
atm(Min),
atm(Max).

Usage: bound(Bound)
  - Description: Bound is a variable interval.

bound_double/1: REGTYPE
Min is a number or an atom that indicates the minimal value, Max indicates the maximal.
The first two for some value and the second pair for some other. Typically used for types that are compound, e.g., rotationvalue.

bound_double(bound(Min0,Max0,Min1,Max1)) :-
atm(Min0),
atm(Max0),
atm(Min1),
atm(Max1).

Usage: bound_double(Bound)
  - Description: Bound is a variable interval.
dictionary/1:  
Dic is a tree structure and is used as the internal representation of the dictionary.

\[
dictionary(dic(Dic)) :-
\]

\[
\text{tree}(Dic).
\]

Usage: dictionary(Dictionary)
\- Description: Dictionary is a dictionary.

environment/1:  
EnvironmentType one of 'DEF','PROTO','EXTERNPROTO' with the name Name. Whitespace is a structure with whitespace information.

\[
environment(env(Env,Name,WhiteSpace)) :-
\]

\[
\text{atm}(Env),
\text{atm}(Name),
\text{whitespace}(WhiteSpace).
\]

Usage: environment(Environment)
\- Description: Environment is an environment structure.

parse/1:  
In is the list of tokens to parse and Out is the resulting list after the parsing. Env is of type env and is the environment-structure. The dictionary Dic contains created information and structures.

\[
parse(parse(In,Out,Env,Dic)) :-
\]

\[
\text{list}(In),
\text{list}(Out),
\text{environment}(Env),
\text{dictionary}(Dic).
\]

Usage: parse(Parse)
\- Description: Parse is a parse structure.

tree/1:  
Key is the search-key, Leaf is the information, Left and Right are more dictionary posts, where Left have less Key-value.

\[
tree(tree(Key,Leaf,Left,Right)) :-
\]

\[
\text{atm}(Key),
\text{leaf}(Leaf),
\text{tree}(Left),
\text{tree}(Right).
\]

Usage: tree(Tree)
\- Description: Tree is a tree structure.

whitespace/1:  
The Row and Indentation information. The row information used when parsing the VRML code to give accurate error position and the indentation is used when generating VRML code from terms.
whitespace(\texttt{w(Row,Indentation)}) :-
    number(Row),
    number(Indentation).

Usage: whitespace(Whitespace)

- Description: \texttt{Whitespace} is a whitespace structure.
209 io (library)

Author(s): Göran Smedbäck.
Version: 0.1#2 (1998/12/2)
This file implements I/O predicates of different types.
Implemented by Göran Smedbäck

209.1 Usage and interface (io)

- Library usage:
  :- use_module(library(io)).
- Exports:
  - Predicates:
    out/1, out/3, convert_atoms_to_string/2, read_terms_file/2, write_terms_file/2, read_vrml_file/2, write_vrml_file/2.
- Other modules used:
  - System library modules:
    aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read, write, lists, format.

209.2 Documentation on exports (io)

out/1:
Usage: out(ListOfOutput)
  - Description: The predicate used is out/3 (DCG) where we will 'save' the output in the second argument. The third argument is the rest, nil.
  - Call and exit should be compatible with:
    ListOfOutput is a list of atms. (basic_props:list/2)
  - The following properties should hold at call time:
    ListOfOutput is currently a term which is not a free variable. (term_typing:nonvar/1)

out/3:
No further documentation available for this predicate.

convert_atoms_to_string/2:
Usage: convert_atoms_to_string(Atoms, String)
  - Description: The predicate transforms a list of atoms to a string.
  - Call and exit should be compatible with:
    Atoms is a list of atms. (basic_props:list/2)
    String is a list of nums. (basic_props:list/2)
The following properties should hold at call time:

Atoms is currently a term which is not a free variable.  
(String is a free variable.)

read_terms_file/2:  
Usage: read_terms_file(Filename, Term)  
Description: Given a filename to a file with terms, the predicate reads the terms and are returned in the second argument. Filename is an atom and Term is the read prolog terms.

Call and exit should be compatible with:

Filename is an atom.  
Term is an atom.

The following properties should hold at call time:

Filename is currently a term which is not a free variable.  
Term is a free variable.

write_terms_file/2:  
Usage: write_terms_file(FileName, List)  
Description: Given a filename and a list of terms the predicate will write them down to the file.

Call and exit should be compatible with:

FileName is an atom.  
List is a list of atoms.

The following properties should hold at call time:

FileName is currently a term which is not a free variable.  
List is currently a term which is not a free variable.

read_vrml_file/2:  
Usage: read_vrml_file(FileName, Data)  
Description: Given a filename, the predicate returns the substance.

Call and exit should be compatible with:

FileName is an atom.  
Data is a string (a list of character codes).

The following properties should hold at call time:

FileName is currently a term which is not a free variable.  
Data is a free variable.

write_vrml_file/2:  
Usage: write_vrml_file(FileName, Data)  
Description: Given a filename and data in form of a string, the predicate will write the data to the named file.
Call and exit should be compatible with:

- `FileName` is an atom.  
  (basicProps:atm/1)
- `Data` is a string (a list of character codes).  
  (basicProps:string/1)

The following properties should hold at call time:

- `FileName` is currently a term which is not a free variable.  
  (term_typing:nonvar/1)
- `Data` is currently a term which is not a free variable.  
  (term_typing:nonvar/1)
210 lookup (library)

Author(s): G"oran Smedb"ack.
Version: 0.1 (1999/1/14, 13:30:46 MET)

210.1 Usage and interface (lookup)

- **Library usage:**
  ```prolog```
  ```:- use_module(library(lookup)).```  
- **Exports:**
  - **Predicates:**
    ```prolog```
    ```create_proto_element/3, get_prototype_interface/2, get_prototype_definition/2, lookup_check_node/4, lookup_check_field/6, lookup_check_interface_fieldValue/8, lookup_field/4, lookup_route/5, lookup_fieldTypeId/1, lookup_get_fieldType/4, lookup_field_access/4, lookup_set_def/3, lookup_set_prototype/4, lookup_setExtern_prototype/4.```  
- **Other modules used:**
  - **System library modules:**
    ```prolog```
    ```aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read, write, lists, provrml/error, provrml/internal_types, provrml/io, provrml/parser_util, provrml/parser, provrml/dictionary, provrml/dictionary_tree, provrml/field_value_check, provrml/boundary, provrml/generator_util, provrml/field_type, provrml/field_value.```  

210.2 Documentation on exports (lookup)

`create_proto_element/3`:
- **PREDICATE**
- **Usage:** `create_proto_element(Interface, Definition, Proto)`
- **Description:** The predicate will construct a proto structure containing the interface and the definition.
- **Call and exit should be compatible with:**
  - `Interface` is any term.  
  - `Definition` is any term.  
  - `Proto` is any term.  
- **The following properties should hold at call time:**
  - `Interface` is currently a term which is not a free variable.  
  - `Definition` is currently a term which is not a free variable.  
  - `Proto` is a free variable.

`get_prototype_interface/2`:
- **PREDICATE**
- **Usage:** `get_prototype_interface(Proto, Interface)`
- **Description:** The predicate will return the interface from a proto structure.
The Ciao Prolog System

Call and exit should be compatible with:
Proto is any term. (basic_props:term/1)
Interface is any term. (basic_props:term/1)

The following properties should hold at call time:
Proto is currently a term which is not a free variable. (term_typing:nonvar/1)
Interface is a free variable. (term_typing:var/1)

get_prototype_definition/2:
Usage: get_prototype_definition(Proto, Definition)
Description: The predicate will return the definition from a proto structure.

Call and exit should be compatible with:
Proto is any term. (basic_props:term/1)
Definition is any term. (basic_props:term/1)

The following properties should hold at call time:
Proto is currently a term which is not a free variable. (term_typing:nonvar/1)
Definition is a free variable. (term_typing:var/1)

lookup_check_node/4:
No further documentation available for this predicate.

lookup_check_field/6:
No further documentation available for this predicate.

lookup_check_interface_fieldValue/8:
Usage: lookup_check_interface_fieldValue(ParseIn, ParseOut, AccessType, FieldType, Id, FieldValue, DCGIn, DCGOut)
Description: The predicate formats the output for the interface part of the prototype. It also checks the values for the fields.

Call and exit should be compatible with:
field_value:parse(ParseIn) (field_value:parse/1)
field_value:parse(ParseOut) (field_value:parse/1)
AccessType is an atom. (basic_props:atm/1)
FieldType is any term. (basic_props:term/1)
Id is an atom. (basic_props:atm/1)
FieldValue is any term. (basic_props:term/1)
DCGIn is a string (a list of character codes). (basic_props:string/1)
DCGOut is a string (a list of character codes). (basic_props:string/1)

The following properties should hold at call time:
ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
ParseOut is a free variable. (term_typing:var/1)
AccessType is currently a term which is not a free variable. (term_typing:nonvar/1)
FieldType is currently a term which is not a free variable. (term_typing:nonvar/1)
Id is currently a term which is not a free variable. (term_typing:nonvar/1)
FieldValue is currently a term which is not a free variable. (term_typing:nonvar/1)
PREDICATE
lookup_field/4:
Usage: lookup_field(Parse, FieldTypeId, FieldId0, FieldId1)

- Description: The predicate will control that the two connected Fields are of the same type, e.g., SFColor - SFColor, MFVec3f - MFVec3f.
- Call and exit should be compatible with:
  - field_value:parse(Parse) (field_value:parse/1)
  - FieldTypeId is an atom. (basic_props:atm/1)
  - FieldId0 is an atom. (basic_props:atm/1)
  - FieldId1 is an atom. (basic_props:atm/1)

- The following properties should hold at call time:
  - Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  - FieldTypeId is currently a term which is not a free variable. (term_typing:nonvar/1)
  - FieldId0 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - FieldId1 is currently a term which is not a free variable. (term_typing:nonvar/1)

PREDICATE
lookup_route/5:
Usage: lookup_route(Parse, NodeTypeId0, FieldId0, NodeTypeId1, FieldId1)

- Description: The predicate will check the routing behaviour for two given fields. They will be checked according to the binding rules, like name changes access properties. The node types for the field must of course be given for the identification.
- Call and exit should be compatible with:
  - field_value:parse(Parse) (field_value:parse/1)
  - NodeTypeId0 is an atom. (basic_props:atm/1)
  - FieldId0 is an atom. (basic_props:atm/1)
  - NodeTypeId1 is an atom. (basic_props:atm/1)
  - FieldId1 is an atom. (basic_props:atm/1)

- The following properties should hold at call time:
  - Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  - NodeTypeId0 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - FieldId0 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - NodeTypeId1 is currently a term which is not a free variable. (term_typing:nonvar/1)
  - FieldId1 is currently a term which is not a free variable. (term_typing:nonvar/1)

PREDICATE
lookup_fieldTypeId/1:
Usage: lookup_fieldTypeId(FieldTypeId)

- Description: The predicate just make a check to see if the given FieldType id is among the allowed. You can not construct own ones and the check is nearly a spellcheck.
- Call and exit should be compatible with:
  - FieldTypeId is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  - FieldTypeId is currently a term which is not a free variable. (term_typing:nonvar/1)
lookup_get_fieldType/4:

Usage: lookup_get_fieldType(Parse, NodeTypeId, Arg3, FieldType)

- Description: The predicate will return the given field's type. It will start the search in the ordinar dictionary and then to the personal dictionary starting off with 'PROTO'. After it will go for 'DEF' and 'EXTERNPROTO'.
- Call and exit should be compatible with:
  field_value:parse(Parse) (field_value:parse/1)
  NodeTypeId is an atom. (basic_props:atm/1)
  Arg3 is an atom. (basic_props:atm/1)
  FieldType is an atom. (basic_props:atm/1)
- The following properties should hold at call time:
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  NodeTypeId is currently a term which is not a free variable. (term_typing:nonvar/1)
  parser:field_Id(Arg3) (parser:field_Id/1)
  FieldType is a free variable. (term_typing:var/1)

lookup_field_access/4:

Usage: lookup_field_access(Parse, NodenameId, FieldId, FieldId)

- Description: The predicate will control that the access properties are correct according to the certain rules that we have. It makes a check to see if the fields are of the same access type or if one of them is an exposedField. It is not doing a route check up to control that behaviour entirely.
- Call and exit should be compatible with:
  field_value:parse(Parse) (field_value:parse/1)
  NodenameId is an atom. (basic_props:atm/1)
  FieldId is an atom. (basic_props:atm/1)
  FieldId is an atom. (basic_props:atm/1)
  FieldId is an atom. (basic Props:atm/1)
- The following properties should hold at call time:
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  NodenameId is currently a term which is not a free variable. (term_typing:nonvar/1)
  FieldId is currently a term which is not a free variable. (term_typing:nonvar/1)
  FieldId is currently a term which is not a free variable. (term_typing:nonvar/1)

lookup_set_def/3:

Usage: lookup_set_def(Parse, Name, Node)

- Description: The predicate will enter a new post in the personal dictionary for the node definition.
- Call and exit should be compatible with:
  field_value:parse(Parse) (field_value:parse/1)
  Name is an atom. (basic_props:atm/1)
  Node is any term. (basic_props:term/1)
- The following properties should hold at call time:
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  Name is currently a term which is not a free variable. (term_typing:nonvar/1)
  Node is currently a term which is not a free variable. (term_typing:nonvar/1)
lookup_set_prototype/4:  
**Usage:** lookup_set_prototype(Parse, Name, Interface, Definition)  

- **Description:** The predicate will insert the prototype definition in the personal dictionary and will give a warning if there is a multiple name given.
- **Call and exit should be compatible with:**
  
  \[
  \text{field\_value:parse(Parse)} \quad \text{(field\_value:parse/1)} \\
  \text{Name is an atom.} \quad \text{(basic\_props:atm/1)} \\
  \text{Interface is any term.} \quad \text{(basic\_props:term/1)} \\
  \text{Definition is any term.} \quad \text{(basic\_props:term/1)} \\
  \]

- **The following properties should hold at call time:**
  
  Parse is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)  
  Name is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)  
  Interface is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)  
  Definition is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)

lookup_setExtern_prototype/4:  
**Usage:** lookup_setExtern_prototype(Parse, Name, Interface, Strings)

- **Description:** The predicate will insert the external prototype definition in the personal dictionary and will give a warning if there is a multiple name given.
- **Call and exit should be compatible with:**
  
  \[
  \text{field\_value:parse(Parse)} \quad \text{(field\_value:parse/1)} \\
  \text{Name is an atom.} \quad \text{(basic\_props:atm/1)} \\
  \text{Interface is any term.} \quad \text{(basic\_props:term/1)} \\
  \text{Strings is any term.} \quad \text{(basic\_props:term/1)} \\
  \]

- **The following properties should hold at call time:**
  
  Parse is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)  
  Name is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)  
  Interface is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)  
  Strings is currently a term which is not a free variable. \hspace{1cm} (term\_typing:nonvar/1)
Chapter 211: parser (library)

Author(s): Göran Smedbäck.

211.1 Usage and interface (parser)

- **Library usage:**
  
  ```prolog
  :- use_module(library(parser)).
  ```

- **Exports:**
  - **Predicates:**
    - parser/2, nodeDeclaration/4.
  - **Properties:**
    - field_Id/1.

- **Other modules used:**
  - **System library modules:**
    - aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read, write, lists, provrml/lookup, provrml/field_value, provrml/tokeniser, provrml/parser_util, provrml/possible, provrml/error.

211.2 Documentation on exports (parser)

PREDICATE parser/2:

**Usage:** parser(VRML, Terms)

- **Description:** The parser uses a tokeniser to read the input text string of VRML code and returns a list with the corresponding terms. The tokens will be read in this parser as the grammar says. The parser is according to the specification of the VRML grammar, accept that it is performed over tokens in stead of the actual code.

- **Call and exit should be compatible with:**
  - VRML is a string (a list of character codes).
  - Terms is a list of terms.

- **The following properties should hold at call time:**
  - VRML is currently a term which is not a free variable.
  - Terms is a free variable.

PREDICATE nodeDeclaration/4:

No further documentation available for this predicate.

PROPERTY field_Id/1:

A property, defined as follows:

```prolog
field_Id(_1).
```
212 parser_util (library)

Author(s): Göran Smedbäck.

212.1 Usage and interface (parser_util)

- **Library usage:**
  
  ```prolog
  :- use_module(library(parser_util)).
  ```

- **Exports:**
  
  - **Predicates:**
    
    ```prolog
    at_least_one/4, at_least_one/5, fillout/4, fillout/5, create_node/3, create_field/3, create_field/4, create_field/5, create_directed_field/5, correct_commenting/4, create_parse_structure/1, create_parse_structure/2, create_parse_structure/3, create_environment/4, insert_comments_in_beginning/3, get_environment_name/2, get_environment_type/2, get_row_number/2, add_environment_whitespace/3, get_indentation/2, inc_indentation/2, dec_indentation/2, push_whitespace/3, push_dictionaries/3, get_parsed/2, get_environment/2, inside_proto/1, get_dictionaries/2, strip_from_list/2, strip_from_term/2, strip_clean/2, strip_exposed/2, strip_restricted/2, strip_interface/2, set_parsed/3, set_environment/3, insert_parsed/3, reverse_parsed/2, stop_parse/2, look_first_parsed/2, get_first_parsed/3, remove_code/3, look_ahead/3.
    ```

- **Other modules used:**
  
  - **System library modules:**
    
    ```prolog
    aggregates, dynamic, iso_misc, iso_byte_char, iso_incomplete, operators, read, write, lists, provrml/dictionary_tree, provrml/internal_types.
    ```

212.2 Documentation on exports (parser_util)

---

**at_least_one/4:**

No further documentation available for this predicate.

---

**at_least_one/5:**

No further documentation available for this predicate.

---

**fillout/4:**

No further documentation available for this predicate.

---

**fillout/5:**

No further documentation available for this predicate.
create_node/3:

Usage: create_node(NodeTypeId, Parse, Node)

- Description: The predicate will construct a node term with the read guts which is inside the parse structure. A node consists of its name and one argument, a list of its fields.

- Call and exit should be compatible with:

  NodeTypeId is an atom. (basic_props:atm/1)
  Parse is a parse structure. (internal_types:parse/1)
  Node is any term. (basic_props:term/1)

- The following properties should hold at call time:

  NodeTypeId is currently a term which is not a free variable. (term_typing:nonvar/1)
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  Node is a free variable.

create_field/3:

Usage: create_field(FieldNameId, Arguments, Field)

- Description: The predicate will construct a field with the Id as the fieldname and the arguments as they are, in a double list, which results in a single list or a single list which will result in free arguments.

- Call and exit should be compatible with:

  FieldNameId is an atom. (basic_props:atm/1)
  Arguments is any term. (basic_props:term/1)
  Field is any term. (basic_props:term/1)

- The following properties should hold at call time:

  FieldNameId is currently a term which is not a free variable. (term_typing:nonvar/1)
  Arguments is currently a term which is not a free variable. (term_typing:nonvar/1)
  Field is a free variable.

create_field/4:

Usage: create_field(FieldAccess, FieldType, FieldId, Field)

- Description: The predicate will construct a field with its access type as the name with type and id as arguments.

- Call and exit should be compatible with:

  FieldAccess is an atom. (basic_props:atm/1)
  FieldType is an atom. (basic_props:atm/1)
  FieldId is an atom. (basic_props:atm/1)
  Field is any term. (basic_props:term/1)

- The following properties should hold at call time:

  FieldAccess is currently a term which is not a free variable. (term_typing:nonvar/1)
  FieldType is currently a term which is not a free variable. (term_typing:nonvar/1)
  FieldId is currently a term which is not a free variable. (term_typing:nonvar/1)
  Field is a free variable.
create_field/5:  
Usage: create_field(FieldAccess, FieldType, FieldId, Fieldvalue, Field)  

- **Description:** The predicate will construct a field with its access type as the name with type, id and value as arguments.  
- **Call and exit should be compatible with:**  
  - FieldAccess is an atom.  
  - FieldType is an atom.  
  - FieldId is an atom.  
  - Fieldvalue is any term.  
  - Field is any term.  

- **The following properties should hold at call time:**  
  - FieldAccess is currently a term which is not a free variable.  
  - FieldType is currently a term which is not a free variable.  
  - FieldId is currently a term which is not a free variable.  
  - Fieldvalue is currently a term which is not a free variable.  
  - Field is a free variable.

create_directed_field/5:  
Usage: create_directed_field(Access, Type, Id0, Id1, Field)  

- **Description:** The predicate will construct a directed field with the key word IS in the middle. Its access type as the name with type, from id0 and to id1 as arguments.  
- **Call and exit should be compatible with:**  
  - Access is an atom.  
  - Type is an atom.  
  - Id0 is an atom.  
  - Id1 is an atom.  
  - Field is any term.  

- **The following properties should hold at call time:**  
  - Access is currently a term which is not a free variable.  
  - Type is currently a term which is not a free variable.  
  - Id0 is currently a term which is not a free variable.  
  - Id1 is currently a term which is not a free variable.  
  - Field is a free variable.

correct_commenting/4:  
Usage: correct_commenting(Place, Comment, ParsedIn, ParsedOut)  

- **Description:** The predicate places the comment 'before' or 'after' the parsed term. This results in a list with the term and the comment or in just returning the term.  
- **Call and exit should be compatible with:**  
  - Place is an atom.  
  - Comment is a compound term.  
  - ParsedIn is any term.  
  - ParsedOut is any term.
The following properties should hold at call time:

Place is currently a term which is not a free variable.  \(\text{term\_typing:nonvar/1}\)
Comment is currently a term which is not a free variable.  \(\text{term\_typing:nonvar/1}\)
ParsedIn is currently a term which is not a free variable.  \(\text{term\_typing:nonvar/1}\)
ParsedOut is a free variable.  \(\text{term\_typing:var/1}\)

create\_parse\_structure/1:

**PREDICATE**

**Usage:** create\_parse\_structure(Parse)

**Description:** The predicate will construct the parse structure with its three fields: the parsing list, the environment structure, and the dictionaries.

**Call and exit should be compatible with:**

Parse is a parse structure.  \(\text{internal\_types:parse/1}\)

**The following properties should hold at call time:**

Parse is a free variable.  \(\text{term\_typing:var/1}\)

create\_parse\_structure/2:

**PREDICATE**

**Usage 1:** create\_parse\_structure(ParseIn, ParseOut)

**Description:** The predicate will construct a parse structure with its three fields: the parsing list, the environment structure, and the dictionaries. It will use the list of parsed items in its structure.

**Call and exit should be compatible with:**

ParseIn is a parse structure.  \(\text{internal\_types:parse/1}\)
ParseOut is a parse structure.  \(\text{internal\_types:parse/1}\)

**The following properties should hold at call time:**

ParseIn is currently a term which is not a free variable.  \(\text{term\_typing:nonvar/1}\)
ParseOut is a free variable.  \(\text{term\_typing:var/1}\)

**Usage 2:** create\_parse\_structure(ParsedList, ParseOut)

**Description:** The predicate will construct a parse structure with its three fields: the parsing list, the environment structure, and the dictionaries. It will use the list of parsed items in its structure.

**Call and exit should be compatible with:**

ParsedList is a list of terms.  \(\text{basic\_props:list/2}\)
ParseOut is a parse structure.  \(\text{internal\_types:parse/1}\)

**The following properties should hold at call time:**

ParsedList is currently a term which is not a free variable.  \(\text{term\_typing:nonvar/1}\)
ParseOut is a free variable.  \(\text{term\_typing:var/1}\)

create\_parse\_structure/3:

**PREDICATE**

**Usage:** create\_parse\_structure(ParsedList, ParseIn, ParseOut)

**Description:** The predicate will construct a parse structure with its three fields: the parsing list, the environment structure, and the dictionaries. It will use the list of parsed items in its structure and the environment and the dictionary from the parse structure given.
– **Call and exit should be compatible with:**
  - ParsedList is a list of terms. (basic_props:list/2)
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)

– **The following properties should hold at call time:**
  - ParsedList is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseOut is a free variable. (term_typing:var/1)

**create_environment/4:**

**Usage:** create_environment(Parse, EnvType, Name, EnvStruct)

– **Description:** The predicate will construct an environment structure based on the information in the parse structure. Well only the white-space information will be reused. The are three types of environments 'PROTO', 'EXTERNPROTO', and 'DEF'.

– **Call and exit should be compatible with:**
  - Parse is a parse structure. (internal_types:parse/1)
  - EnvType is an atom. (basic_props:atm/1)
  - Name is an atom. (basic_props:atm/1)
  - EnvStruct is an environment structure. (internal_types:environment/1)

– **The following properties should hold at call time:**
  - Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  - EnvType is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Name is currently a term which is not a free variable. (term_typing:nonvar/1)
  - EnvStruct is a free variable. (term_typing:var/1)

**insert_comments_in_beginning/3:**

**Usage:** insert_comments_in_beginning(Comment, ParseIn, ParseOut)

– **Description:** We add the comment in the beginning of the parsed, to get the proper look.

– **Call and exit should be compatible with:**
  - Comment is a compound term. (basic_props:struct/1)
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)

– **The following properties should hold at call time:**
  - Comment is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseOut is a free variable. (term_typing:var/1)

**get_environment_name/2:**

**Usage:** get_environment_name(Environment, Name)

– **Description:** The predicate will return the environment name.
get_environment_type/2:

Usage: get_environment_type(Environment, Type)

- Description: The predicate will return the environment type, one of the three: 'PROTO', 'EXTERNPROTO', and 'DEF'.
- Call and exit should be compatible with:
  Environment is an environment structure.  (internal_types:environment/1)
  Name is an atom.                     (basic_props:atm/1)
- The following properties should hold at call time:
  Environment is currently a term which is not a free variable.   (termotyping:nonvar/1)
  Name is a free variable.                                (term_typing:var/1)

get_row_number/2:

Usage: get_row_number(Parse, Row)

- Description: The predicate will return the row number from the parse structure. The row number is not fully implemented.
- Call and exit should be compatible with:
  Parse is a parse structure.                         (internal_types:parse/1)
  Row is a number.                                    (basic_props:num/1)
- The following properties should hold at call time:
  Parse is currently a term which is not a free variable.  (term_typing:nonvar/1)
  Row is a free variable.                             (term_typing:var/1)

add_environment_whitespace/3:

Usage: add_environment_whitespace(EnvIn, WhiteSpaceList, EnvOut)

- Description: The predicate will add the new whitespace that is collected in a list of whitespaces to the environment.
- Call and exit should be compatible with:
  EnvIn is an environment structure.                     (internal_types:environment/1)
  WhiteSpaceList is a list of atms.                      (basic_props:list/2)
  EnvOut is an environment structure.                    (internal_types:environment/1)
- The following properties should hold at call time:
  EnvIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  WhiteSpaceList is currently a term which is not a free variable. (term_typing:nonvar/1)
  EnvOut is a free variable.                            (term_typing:var/1)
get indentation/2:

Usage 1: get_indentation(Whitespace, Indentation)

- **Description:** The predicate will return the indentation depth from a whitespace structure.
- **Call and exit should be compatible with:**
  - Whitespace is a whitespace structure. (internal_types:whitespace/1)
  - Indentation is a number. (basic_props:num/1)
- **The following properties should hold at call time:**
  - Whitespace is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Indentation is a free variable. (term_typing:var/1)

Usage 2: get_indentation(Parse, Indentation)

- **Description:** The predicate will return the indentation depth from a parse structure.
- **Call and exit should be compatible with:**
  - Parse is a parse structure. (internal_types:parse/1)
  - Indentation is a number. (basic_props:num/1)
- **The following properties should hold at call time:**
  - Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Indentation is a free variable. (term_typing:var/1)

inc indentation/2:

Usage: inc_indentation(ParseIn, ParseOut)

- **Description:** Will increase the indentation with one step to a parse structure.
- **Call and exit should be compatible with:**
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)
- **The following properties should hold at call time:**
  - ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseOut is a free variable. (term_typing:var/1)

dec indentation/2:

Usage: dec_indentation(ParseIn, ParseOut)

- **Description:** Will decrease the indentation with one step to a parse structure.
- **Call and exit should be compatible with:**
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)
- **The following properties should hold at call time:**
  - ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseOut is a free variable. (term_typing:var/1)

add indentation/3:

No further documentation available for this predicate.
reduce_indentation/3:
No further documentation available for this predicate.

push_whitespace/3:
Usage: push_whitespace(ParseWithWhitespace, ParseIn, ParseOut)
- Description: The predicate will add the whitespace values from one parse structure to another one, resulting in the output, with the values from the second parse structure with the whitespace from the first added.
- Call and exit should be compatible with:
  ParseWithWhitespace is a parse structure. (internal_types:parse/1)
  ParseIn is a parse structure. (internal_types:parse/1)
  ParseOut is a parse structure. (internal_types:parse/1)
- The following properties should hold at call time:
  ParseWithWhitespace is currently a term which is not a free variable. (term_typing:nonvar/1)
  ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  ParseOut is a free variable. (term_typing:var/1)

push_dictionaries/3:
Usage: push_dictionaries(Parse, Parse, Parse)
- Description: The predicate will set the first parse structure’s directory to the second parsing structure argument. The resulting parsing structure will be returned.
- Call and exit should be compatible with:
  Parse is a parse structure. (internal_types:parse/1)
  Parse is a parse structure. (internal_types:parse/1)
  Parse is a parse structure. (internal_types:parse/1)
- The following properties should hold at call time:
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  Parse is a free variable. (term_typing:var/1)

get_parsed/2:
Usage 1: get_parsed(ParseStructure,ListOfParsed)
- Description: The predicate will return a list of the parsed terms that is inside the parse structure.
- Call and exit should be compatible with:
  ParseStructure is a parse structure. (internal_types:parse/1)
  ListOfParsed is a list of terms. (basic_props:list/2)
- The following properties should hold at call time:
  ParseStructure is currently a term which is not a free variable. (term_typing:nonvar/1)
  ListOfParsed is a free variable. (term_typing:var/1)
Usage 2: get_parsed(ParseStructure, EnvironmentStructure)

- **Description:** The predicate will return the environment of the parse structure.
- **Call and exit should be compatible with:**
  - ParseStructure is a parse structure. (internal_types:parse/1)
  - EnvironmentStructure is an environment structure. (internal_types:environment/1)
- **The following properties should hold at call time:**
  - ParseStructure is currently a term which is not a free variable. (term_typing:nonvar/1)
  - EnvironmentStructure is a free variable. (term_typing:var/1)

Usage 3: get_parsed(ParseStructure, Dictionaries)

- **Description:** The predicate will return dictionary used within the parse structure.
- **Call and exit should be compatible with:**
  - ParseStructure is a parse structure. (internal_types:parse/1)
  - Dictionaries is a dictionary. (internal_types:dictionary/1)
- **The following properties should hold at call time:**
  - ParseStructure is currently a term which is not a free variable. (term_typing:nonvar/1)
  - Dictionaries is a free variable. (term_typing:var/1)

**get_environment/2:**

No further documentation available for this predicate.

**inside_proto/1:**

**Usage:** inside_proto(Parse)

- **Description:** The predicate will answer to the question: are we parsing inside a PROTO/EXTERNPROTO.
- **Call and exit should be compatible with:**
  - Parse is a parse structure. (internal_types:parse/1)
- **The following properties should hold at call time:**
  - Parse is currently a term which is not a free variable. (term_typing:nonvar/1)

**get_dictionaries/2:**

No further documentation available for this predicate.

**strip_from_list/2:**

**Usage:** strip_from_list(ListWithComments, CleanList)

- **Description:** The predicate will strip the list from comments and return a list without any comments.
- **Call and exit should be compatible with:**
  - ListWithComments is a list of terms. (basic_props:list/2)
  - CleanList is a list of terms. (basic_props:list/2)
– The following properties should hold at call time:
  ListWithComments is currently a term which is not a free variable.  \( \text{term_typing:nonvar/1} \)
  CleanList is a free variable.  \( \text{term_typing:var/1} \)

strip_from_term/2:
Usage: strip_from_term(Term, Stripped)
– Description: The predicate will remove comments from a term, it will reduce its arguments if there are comments as arguments.
– Call and exit should be compatible with:
  Term is any term.  \( \text{basic_props:term/1} \)
  Stripped is any term.  \( \text{basic_props:term/1} \)
– The following properties should hold at call time:
  Term is currently a term which is not a free variable.  \( \text{term_typing:nonvar/1} \)
  Stripped is a free variable.  \( \text{term_typing:var/1} \)

strip_clean/2:
Usage: strip_clean(ParsedIn, ParsedOut)
– Description: The predicate will return a striped list or a single atom if there was no comments and no more items in the list. It will also return a atom if there is comments and only one other element.
– Call and exit should be compatible with:
  ParsedIn is any term.  \( \text{basic_props:term/1} \)
  ParsedOut is any term.  \( \text{basic_props:term/1} \)
– The following properties should hold at call time:
  ParsedIn is currently a term which is not a free variable.  \( \text{term_typing:nonvar/1} \)
  ParsedOut is a free variable.  \( \text{term_typing:var/1} \)

stripExposed/2:
No further documentation available for this predicate.

stripRestricted/2:
No further documentation available for this predicate.

stripInterface/2:
Usage: strip_interface(Interface, StrippedInterface)
– Description: The predicate will remove comments from the interface that we read for the PROTOtype. This will help us when setting the properties.
– Call and exit should be compatible with:
  Interface is a list of terms.  \( \text{basic_props:list/2} \)
  StrippedInterface is a list of terms.  \( \text{basic_props:list/2} \)
The following properties should hold at call time:

- **Interface** is currently a term which is not a free variable. (term_typing:nonvar/1)
- **StrippedInterface** is a free variable. (term_typing:var/1)

**set_parsed/3:**

Usage: `set_parsed(ParseIn, NewParseList, ParseOut)`

- **Description:** The predicate will create a new parse structure from the first parse structure with the parse list from the second argument.

- **Call and exit should be compatible with:**
  - **ParseIn** is a parse structure. (internal_types:parse/1)
  - **NewParseList** is a list of terms. (basic_props:list/2)
  - **ParseOut** is a parse structure. (internal_types:parse/1)

- **The following properties should hold at call time:**
  - **ParseIn** is currently a term which is not a free variable. (term_typing:nonvar/1)
  - **NewParseList** is currently a term which is not a free variable. (term_typing:nonvar/1)
  - **ParseOut** is a free variable. (term_typing:var/1)

**set_environment/3:**

Usage: `set_environment(Environment, ParseIn, ParseOut)`

- **Description:** The modifier will return a parse structure with the environment given with the other properties from the first parse structure.

- **Call and exit should be compatible with:**
  - **Environment** is an environment structure. (internal_types:environment/1)
  - **ParseIn** is a parse structure. (internal_types:parse/1)
  - **ParseOut** is a parse structure. (internal_types:parse/1)

- **The following properties should hold at call time:**
  - **Environment** is currently a term which is not a free variable. (term_typing:nonvar/1)
  - **ParseIn** is currently a term which is not a free variable. (term_typing:nonvar/1)
  - **ParseOut** is a free variable. (term_typing:var/1)

**insert_parsed/3:**

No further documentation available for this predicate.

**reverse_parsed/2:**

No further documentation available for this predicate.
**stop_parse/2:**

**Usage:** stop_parse(TermIn, TermOut)

- **Description:** The predicate will bind the invalue to the outvalue, used to terminate a parsing.

- **Call and exit should be compatible with:**
  - TermIn is any term. (basic_props:term/1)
  - TermOut is any term. (basic_props:term/1)

- **The following properties should hold at call time:**
  - TermIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - TermOut is a free variable. (term_typing:var/1)

**look_first_parsed/2:**

**Usage:** look_first_parsed(Parse, First)

- **Description:** Look at the first item in the parse structure.

- **Call and exit should be compatible with:**
  - Parse is a parse structure. (internal_types:parse/1)
  - First is any term. (basic_props:term/1)

- **The following properties should hold at call time:**
  - Parse is currently a term which is not a free variable. (term_typing:nonvar/1)
  - First is a free variable. (term_typing:var/1)

**get_first_parsed/3:**

**Usage:** get_first_parsed(ParseIn, ParseOut, First)

- **Description:** Get the first item in the parse structure and return the parse structure with the item removed.

- **Call and exit should be compatible with:**
  - ParseIn is a parse structure. (internal_types:parse/1)
  - ParseOut is a parse structure. (internal_types:parse/1)
  - First is any term. (basic_props:term/1)

- **The following properties should hold at call time:**
  - ParseIn is currently a term which is not a free variable. (term_typing:nonvar/1)
  - ParseOut is a free variable. (term_typing:var/1)
  - First is a free variable. (term_typing:var/1)

**remove_code/3:**

No further documentation available for this predicate.

**look_ahead/3:**

**Usage:** look_ahead(Name, Parsed, Parsed)

- **Description:** This predicate is used normally by the CDG and the two last arguments will therefore be the same because we don’t remove the parsed. The name is the name inside a term, the first argument.
Call and exit should be compatible with:

- Name is an atom.  
  - (basic_props:atm/1)
- Parsed is a list of terms.  
  - (basic_props:list/2)
- Parsed is a list of terms.  
  - (basic_props:list/2)

The following properties should hold at call time:

- Name is currently a term which is not a free variable.  
  - (term_typing:nonvar/1)
- Parsed is currently a term which is not a free variable.  
  - (term_typing:nonvar/1)
- Parsed is a free variable.  
  - (term_typing:var/1)
213 possible (library)

Author(s): Göran Smedbäck.
Version: 0.1 (1999/2/19, 6:32:46 MET)

213.1 Usage and interface (possible)

- Library usage:
  :- use_module(library(possible)).
- Exports:
  - Predicates:
    continue/3.
- Other modules used:
  - System library modules:
    lists.

213.2 Documentation on exports (possible)

continue/3:  
No further documentation available for this predicate.
214 tokeniser (library)

Author(s): Göran Smedbäck.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#178 (2003/12/4, 17:49:33 CET)

214.1 Usage and interface (tokeniser)

- Library usage:
  
  ```prolog
  :- use_module(library(tokeniser)).
  ```

- Exports:

  - Predicates:
    
    - `tokeniser/2`, `token_read/3`.

- Other modules used:

  - System library modules:
    
    iso_byte_char, lists, write, provrml/error.

214.2 Documentation on exports (tokeniser)

tokeniser/2:

**PREDICATE**

**Usage:** `tokeniser(VRML, Tokens)`

- **Description:** This predicate will perform the parsing of the VRML code. The result will be tokens that will be the source for producing the Prolog terms of the VRML code. This is done in the parser module. From these terms analysis, changing, and any thing that you want to do with VRML code from Prolog programming language. We perform the predicate with a catch call to be able to output error messages if encountered.

- *Call and exit should be compatible with:*
  
  VRML is a list of `atm`s.  
  Tokens is a list of `term`s.  

- *The following properties should hold at call time:*
  
  VRML is currently a term which is not a free variable.  
  Tokens is a free variable.


token_read/3:

**PREDICATE**

No further documentation available for this predicate.
Chapter 215: Double linked list

215 Double linked list

Author(s): David Trallero Mena.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.9#116 (2003/12/1, 22:4:57 CET)

This library allows the user to work with double linked lists. An index is used for referencing the current element in the list. Such index can be modified by next and prev predicates. The value of the current index can be obtained with top predicate.

215.1 Usage and interface (ddlist)

- Library usage:
  ```prolog```
  ```prolog
  :- use_module(library(ddlist)).
  ```

- Exports:
  ```prolog```
  ```prolog
  Predicates:
  null_list/1, next/2, prev/2, insert/3, insert_top/3, insert_after/3, delete/2,
  delete_top/2, delete_after/2, top/2, rewind/2, forward/2, length/2, length_next/2, length_prev/2.
  Regular Types:
  ddlist/1.
  ```

- Other modules used:
  ```prolog```
  ```prolog
  System library modules:
  lists.
  ```

215.2 Documentation on exports (ddlist)

null_list/1:  
```
Usage: null_list(NullList)

Description: NullList is an empty list
```

next/2:  
```
Usage: next(ddlist(OldList), ddlist(NewList))

Description: NewList is OldList but index is set to next element of current element of OldList. It satisfies next(A,B),prev(B,A)
```

prev/2:  
```
Usage: prev(ddlist(OldList), ddlist(NewList))

Description: NewList is OldList but index is set to previous element of current element of OldList top) of OldList
```
insert/3:
Usage: insert(ddlist(List), Element, ddlist(NewList))
- Description: NewList is like List but with Element inserted BEFORE the current index. It satisfies insert( X, A, Xp ), delete( Xp, X ).

insert_top/3:
Usage: insert_top(ddlist(List), Element, ddlist(NewList))
- Description: Put Element as new top of NewList and push the rest of elements after it. It satisfies top( NewList, element ).

insert_after/3:
Usage: insert_after(ddlist(List), Element, ddlist(NewList))
- Description: NewList is like List but with Element inserted AFTER the current index. It satisfies insert_after( X, A, Xp ), delete_after( Xp, X ).

delete/2:
Usage: delete(ddlist(OldList), ddlist(NewList))
- Description: NewList does not have the previous element (top(prev)) of OldList.

delete_top/2:
Usage: delete_top(ddlist(OldList), ddlist(NewList))
- Description: NewList does not have the current element (top) of OldList.

delete_after/2:
Usage: delete_after(ddlist(OldList), ddlist(NewList))
- Description: NewList does not have next element to current element (top) of OldList.

top/2:
Usage: top(ddlist(List), Element)
- Description: Element is the element pointed by index.

rewind/2:
Usage: rewind(ddlist(OldList), ddlist(NewList))
- Description: NewList is the OldList but index is set to 0.

forward/2:
Usage: forward(ddlist(OldList), ddlist(NewList))
- Description: NewList is the OldList but index is set to length of NewList.
Chapter 215: Double linked list

215.3 Other information (ddlist)

Two simple examples of the use of the ddlist library package follow.

215.3.1 Using insert_after

:- module( ddl1 , _ , [] ).

:- use_module( library(ddlist) ) .

main(A,B):-
  % L = []
  null_list( L ),
  % L = [1]
  insert_after( L , 1 , L1 ),
  % L = [1,2]
  insert_after( L1 , 2 , L2 ),
  % L = [1,3,2]
  insert_after( L2 , 3 , L3 ),
  % L = [1,3,2] => A = [1]
  top( L3 , A ),
  % L = [3,2]
  next( L3 , PL3 ),
  top( PL3 , B ) .
215.3.2 More Complex example

:- module( ddl2 , _ , [ ] ).

:- use_module( library(ddlist) ).

main(A,B):-
  % L = []
  null_list( L ),
  % L = [1]
  insert_after( L , 1 , L1 ),
  % L = [1,2]
  insert_after( L1 , 2 , L2 ),
  % L = [1,2]
  insert( L2 , 3 , L3 ),
  % L = [3,1,2]
  prev( L3 , PL3 ),
  % L = []
  forward( PL3 , FOR ),
  % L = [2]
  prev( FOR , FOR1 ),
  % L = [2] => A = 2
  top( FOR1 , A ),
  % L = [1,2]
  prev( FOR1 , FOR2 ),
  % L = [2]
  delete_after( FOR2 , FOR3 ),
  % L = [3,2]
  prev( FOR3 , FOR4 ),
  % L = [3,2] => B = 3
  top( FOR4 , B ).
216 Measuring features from predicates (time cost or memory used...)

Author(s): David Trallero Mena.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.0#2 (2003/12/3, 0:10:32 CET)

This library has been done for measuring or compare execution features (currently only time) of predicates. This module relies on gnuplot, an auxiliary module which use the tool gnuplot, for representing results graphically.

216.1 Usage and interface (time_analyzer)

- Library usage:
  :- use_module(library(time_analyzer)).
- Exports:
  - Predicates:
    performance/3, benchmark/6, compare_benchmark/7, benchmark2/6, compare_benchmark2/7, sub_times/3, div_times/2, cost/3.
- Other modules used:
  - System library modules:
    gnuplot/gnuplot, prolog_sys, lists, write, system, hiordlib.

216.2 Documentation on exports (time_analyzer)

performance/3: Meta-predicate with arguments: performance(goal,?,?).

- Description: performance accepts a goal, P, as a first argument. The aim of this predicate is to call P several times and measure some feature (in this version, only time, that is because no extra parameter has been added). M defines how many times P should be called. Usually, calling the predicate in some succession (10,100,1000) and dividing by the number of times it is executed we can obtain the "execution time" of the predicate (if we are measuring time).

The result of executions are returned in the list Times

The different modes are:
- graph( Start, End, Increment ). It defines arithmetic succession starting in Start and ending in End, by increment of Increment. So P is called Start times on the first time, Start+Increment on the second, etc.
- graph The same as graph/3 but with default options
- graph_exp( Start, End, Exp ). It defines geometric succession. Start is multiplied by Exp till it gets End. So P is called Start times on the first time, Start*Exp on the second, etc.
- graph_exp The same as graph_exp/3 but with default options
The Ciao Prolog System

The following properties should hold at call time:
- \( P \) is a term which represents a goal, i.e., an atom or a structure. (\( \text{basic_props:callable/1} \))
- \( M \) is any term. (\( \text{basic_props:term/1} \))
- \( \text{Times} \) is a free variable. (\( \text{term_typing:var/1} \))

The following properties should hold upon exit:
- \( P \) is a term which represents a goal, i.e., an atom or a structure. (\( \text{basic_props:callable/1} \))
- \( M \) is any term. (\( \text{basic_props:term/1} \))
- \( \text{Times} \) is a list of num s. (\( \text{basic_props:list/2} \))

**PREDICATE**

**benchmark/6:**

Usage: \( \text{benchmark}(P, \text{BenchList}, \text{NumTimes}, \text{Method}, \text{Reserved}, \text{OutList}) \)

**Description:** The predicate \( P \), which accepts ONE argument, is called with the first member of each pair of the BenchList list NumTimes. The entry list have pairs because the second member of the pair express the meaning of the first one in the X-Axis. For example, if we are doing a benchmark of qsort function, the first member will be a list for being ordered and the second one will be the length of the unordered list. The output is a list of (X,Y) points where Y means the time needed for its entry of "cost" X. \( \text{OutList} \) can be used as TimeList in predicate generate_plot. \( \text{Reserved} \) is reserved for future implementations (it will take the value of runtime, memory_used...)

The following properties should hold at call time:
- \( P \) is a term which represents a goal, i.e., an atom or a structure. (\( \text{basic_props:callable/1} \))
- BenchList is a list of pairs. (\( \text{basic_props:list/2} \))
- NumTimes is an integer. (\( \text{basic_props:int/1} \))
- \( \text{time_analyzer:average_mode}(\text{Method}) \) (\( \text{time_analyzer:average_mode/1} \))
- Reserved is any term. (\( \text{basic_props:term/1} \))
- OutList is a free variable. (\( \text{term_typing:var/1} \))

The following properties should hold upon exit:
- \( P \) is a term which represents a goal, i.e., an atom or a structure. (\( \text{basic_props:callable/1} \))
- BenchList is a list of pairs. (\( \text{basic_props:list/2} \))
- NumTimes is an integer. (\( \text{basic_props:int/1} \))
- \( \text{time_analyzer:average_mode}(\text{Method}) \) (\( \text{time_analyzer:average_mode/1} \))
- Reserved is any term. (\( \text{basic_props:term/1} \))
- OutList is a list of pairs. (\( \text{basic_props:list/2} \))

**compare_benchmark/7:**

Usage: \( \text{compare_benchmark}(\text{ListPred}, \text{BenchList}, \text{Method}, \text{NumTimes}, \text{BaseName}, \text{Reserved}, \text{GeneralOptions}) \)

**Description:** It is the generalization of execute predicate \( \text{benchmark/6} \) with several predicates. \( \text{benchmark/6} \) predicate is called with each predicate in ListPred, and \( \text{BaseName} \) is used for the temporaries basename file. \( \text{GeneralOptions} \) are aplied to the plot
The following properties should hold at call time:

- \( \text{ListPred} \) is a list of \( \text{preds} \).  
- \( \text{BenchList} \) is a list.  
- \( \text{time\_analyzer:average\_mode}(\text{Method}) \)  
- \( \text{NumTimes} \) is an integer.  
- \( \text{BaseName} \) is currently instantiated to an atom.  
- \( \text{Reserved} \) is any term.  
- \( \text{GeneralOptions} \) is a free variable.  

The following properties should hold upon exit:

- \( \text{ListPred} \) is a list of \( \text{preds} \).  
- \( \text{BenchList} \) is a list.  
- \( \text{time\_analyzer:average\_mode}(\text{Method}) \)  
- \( \text{NumTimes} \) is an integer.  
- \( \text{BaseName} \) is currently instantiated to an atom.  
- \( \text{Reserved} \) is any term.  
- \( \text{GeneralOptions} \) is a list.  

**benchmark2/6:**

Usage: \( \text{benchmark2}(P, \text{BenchList}, \text{Method}, \text{NumTimes}, \text{What}, \text{OutList}) \)

- **Description:** The predicate \( P \), which accepts TWO arguments, is called \( \text{NumTimes} \) with the first member of each pair of the \( \text{BenchList} \) list and a free variable as the second. The time of execution (in the future, the desired featured for be measured) is expected to be the second argument, that is because it is a variable. The entry list, \( \text{BenchList} \) have pairs because the second member of the pair express the cost of the first (in X-Axis). For example, if we are doing a benchmark of qsort function, the first member will be a list for being ordered and the second one will represent the length of the unordered list. The output is a list of \( (X,Y) \) points where \( Y \) express the time needed for they entry of "cost" \( X \). \( \text{OutList} \) can be used as \( \text{TimeList} \) in predicate \( \text{generate\_plot} \). \( \text{What} \) is reserved for future use.

- **The following properties should hold at call time:**

- \( P \) is a term which represents a goal, i.e., an atom or a structure.  
- \( \text{BenchList} \) is a list of pairs.  
- \( \text{time\_analyzer:average\_mode}(\text{Method}) \)  
- \( \text{NumTimes} \) is an integer.  
- \( \text{What} \) is currently instantiated to an atom.  
- \( \text{OutList} \) is a free variable.  

- **The following properties should hold upon exit:**

- \( P \) is a term which represents a goal, i.e., an atom or a structure.  
- \( \text{BenchList} \) is a list of pairs.  
- \( \text{time\_analyzer:average\_mode}(\text{Method}) \)  
- \( \text{NumTimes} \) is an integer.  
- \( \text{What} \) is currently instantiated to an atom.  
- \( \text{OutList} \) is a list of pairs.
**compare_benchmark2/7:**

**Usage:** compare_benchmark2(ListPred, BenchList, Method, NumTimes, BaseName, Reserved, GeneralOptions)

- **Description:** It is the generalization of execute predicate `benchmark2/6` with several predicates. `benchmark2/6` is called with each predicate in `ListPred` and `BaseName` is used for the temporary basename file. `GeneralOptions` are applied to the plot ('default' can be used for default General options)

- **The following properties should hold at call time:**
  - `ListPred` is a list of `preds`. (basic_props:list/2)
  - `BenchList` is a list. (basic_props:list/1)
  - `time_analyzer:average_mode(Method)` (time_analyzer:average_mode/1)
  - `NumTimes` is an integer. (basic_props:int/1)
  - `BaseName` is currently instantiated to an atom. (term_typing:atom/1)
  - `Reserved` is currently instantiated to an atom. (term_typing:atom/1)
  - `GeneralOptions` is a list. (basic_props:list/1)

- **The following properties should hold upon exit:**
  - `ListPred` is a list of `preds`. (basic_props:list/2)
  - `BenchList` is a list. (basic_props:list/1)
  - `time_analyzer:average_mode(Method)` (time_analyzer:average_mode/1)
  - `NumTimes` is an integer. (basic_props:int/1)
  - `BaseName` is currently instantiated to an atom. (term_typing:atom/1)
  - `Reserved` is currently instantiated to an atom. (term_typing:atom/1)
  - `GeneralOptions` is a list. (basic_props:list/1)

**sub_times/3:**

**Usage:** sub_times(A, B, C)

- **Description:** `C` is the result of doing `A - B`, where `A`, `B`, `C` are a list of pairs as `(Time,_)`

- **Call and exit should be compatible with:**
  - `A` is a list of `pairs`. (basic_props:list/2)
  - `B` is a list of `pairs`. (basic_props:list/2)
  - `C` is a list of `pairs`. (basic_props:list/2)

**div_times/2:**

**Usage:** div_times(A, B)

- **Description:** `A` is a list of pairs `(P1,P2)`. `B` is a list of pairs with the form `(P1,P2/P1)` for each `(P1,P2)` that belongs to `A`

- **Call and exit should be compatible with:**
  - `A` is a list of `pairs`. (basic_props:list/2)
  - `B` is a list of `pairs`. (basic_props:list/2)
**cost/3:**

*Meta-predicate* with arguments: `cost(goal,?,?,?)`.

**Usage:** `cost(A, T, What)`

- **Description:** This pred is thought for measuring constant complexity predicates. `T` is the expected measured feature. `What` is reserved for future implementations, just put 'runtime'.
- **Call and exit should be compatible with:**
  - `A` is a term which represents a goal, i.e., an atom or a structure.  
    
  - `T` is an integer.  
    
  - `What` is any term.  

---

**generate_plot/3:**

Imported from *gnuplot* (see the corresponding documentation for details).

**generate_plot/2:**

Imported from *gnuplot* (see the corresponding documentation for details).

**set_general_options/1:**

Imported from *gnuplot* (see the corresponding documentation for details).

**get_general_options/1:**

Imported from *gnuplot* (see the corresponding documentation for details).
217 Printing graph using gnuplot as auxiliary tool.

**Author(s):** David Trallero Mena.

This library uses gnuplot for printing graphs.

Two predicates are provided:

- `generate_plot(BaseName, DataList)`
- `generate_plot(BaseName, DataList, GlobalOptions)`

Several files can be generated as temporary files. A BaseName is required for generating the temporary files. Data files name will be created from BaseName + number + .dat. The BaseName + ".plot" will be the name used for gnuplot tool.

A list of pairs of list of pairs of the from (X,Y) and Local Option value is provided to the main predicate as data. In other words `DataList = [(CurveDataList,LocalOptions), (CurveDataList1,LocalOptions1) ...]`. Additionally `(function( String ), LocalOptions)` can be used for adding a curve to the plot (imagine you want to compare your result with x=y).

LocalOptions of the DataList are options that are applied to the curve, as for example, if we print the curve with lines, or the title in the legend, etc. GlobalOptions are referred to the plot options, like title in x or y axis, etc.

217.1 Usage and interface (gnuplot)

- **Library usage:**
  :- use_module(library(gnuplot)).

- **Exports:**
  - Predicates:
    - `get_general_options/1`, `set_general_options/1`, `generate_plot/2`, `generate_plot/3`.

- **Other modules used:**
  - System library modules:
    - lists, write, system.

217.2 Documentation on exports (gnuplot)

**get_general_options/1:**

**Usage:** `get_general_options(X)`

- **Description:** Get the general options of the graphic that will be plotted
- **The following properties should hold at call time:**
  - X is a free variable. (term_typing:var/1)
- **The following properties should hold upon exit:**
  - X is a list. (basic_props:list/1)

**set_general_options/1:**

**Usage:** `set_general_options(list(X))`
- **Description**: Get the general options of the graphic that will be plotted. Possible options are:
  - `format(A)` Specify the format of points
  - `nokey` Legend is not represented
  - `nogrid` No grid
  - `grid` An smooth grid is shown
  - `label(L, (X,Y))` Put Label L at point (X,Y)
  - `xlabel(A)` Label of X-Axis
  - `ylabel(A)` Label of Y-Axis
  - `xrange(A,B)` Define the X range representation
  - `yrange(A,B)` Define the Y range representation
  - `title(A)` Title of the plot
  - `key(A)` Define the key (for example `[left,box]`, left is the position, box indicates that a box should be around)
  - `term_post(A)` Define the postscript terminal. A is a list of atoms.
  - `size(A,B)` Specify the size of the plot (A,B float numbers)
  - `autoscale` Autoscale the size of the plot
  - `autoscale(A)` Autoscale the argument (for example: `autoscale(x)`)

**generate_plot/2:**

**Usage:** `generate_plot(BaseName, TimeList)`

- **Description**: This predicate generates a `BaseName + .ps` postscript file using each element of `TimeList` as a pair of list of pairs and local options, i.e., `(list((X,Y)), LocalOptions)`, in which X is the position in X-Axis and Y is the position in Y-Axis. Nevertheless, each element of `TimeList` can be a list of pairs instead of a pair for comodity. `gnuplot` is used as auxiliary tool. Temporary files `BaseName + N.dat` are generated for every list of pairs, and `BaseName + .plot` is the file used by `gnuplot`. The local options can be:
  - `with(Option)` Tells how the curve will be represented. Option can be line, dots, boxes, impulses, linespoints. This option HAVE TO BE the last one
  - `title(T)` Put the name of the curve in the legend to T
  - The following properties should hold at call time:
    - `BaseName` is currently instantiated to an atom. (`term_typing:atom/1`)
    - `TimeList` is a list of pairs. (`basic_props:list/2`)

- The following properties should hold upon exit:
  - `BaseName` is currently instantiated to an atom. (`term_typing:atom/1`)
  - `TimeList` is a list of pairs. (`basic_props:list/2`)

**generate_plot/3:**

**Usage:** `generate_plot(BaseName, TimeList, GeneralOptions)`

- **Description**: It is the same as `generate_plot/2` but `GeneralOptions` are used as the general options of the plot. Look at predicate `set_general_options` for detailed description of possible options
The following properties should hold at call time:

- **BaseName** is currently instantiated to an atom. (term_typing:atom/1)
- **TimeList** is a list of pairs. (basic_props:list/2)
- **GeneralOptions** is a list. (basic_props:list/1)

The following properties should hold upon exit:

- **BaseName** is currently instantiated to an atom. (term_typing:atom/1)
- **TimeList** is a list of pairs. (basic_props:list/2)
- **GeneralOptions** is a list. (basic_props:list/1)
218 Automatic modules caller tester

Author(s): David Trallero Mena.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.0#6 (2004/2/24, 17:12:13 CET)

This module is intended to agilize the work of calling several modules as prove cases of some program. Usually when you are developing a program you have several auto-test program cases that you would like to execute whenever you do some modification in your program/system. The predicate mod_tester/2 was created with the propouse of execute this test an report to you which of them were correctly executed and which one were not.

218.1 Usage and interface (modtester)

- Library usage:
  
  ```prolog
  :- use_module(library(modtester)).
  ```

- Exports:
  - Predicates:
    
    ```prolog
    tester_func/1, modules_tester/2, pred_tester/2.
    ```

- Other modules used:
  - System library modules:
    
    ```prolog
    tester/tester, lists, write, filenames, compiler/compiler, terms_check, conc_aggregates, system.
    ```

218.2 Documentation on exports (modtester)

tester_func/1:

No further documentation available for this predicate.

modules_tester/2:

Usage 1: modules_tester(BaseName, ModulesList)

- Description: modules_tester accepts an atom as basename of the two generated files. For each module in ModulesList an output and report is saved in 'basename_test_output.log' and 'basename_test_summary.log' respectively
- The following properties should hold at call time:
  
  BaseName is currently instantiated to an atom. (term_typing:atom/1)
  ModulesList is a list. (basic_props:list/1)

Usage 2: modules_tester(BaseName, PredList)

- Description: pred_tester accepts an atom as basename of the two generated files. For each element with the pattern (FindPatter, precidate, [results]), module in PredList an output and report is saved in 'basename_test_output.log' and 'basename_test_summary.log' respectively. For example, you can call this predicate as: pred_tester( test , [(X,mypred(X),[1,2,3]),( (X,Y),mypred2( X, aa , Y ), [(1,2),(2,3)])] ).
The following properties should hold at call time:

- BaseName is currently instantiated to an atom.
- PredList is a list.

pred_tester/2:
No further documentation available for this predicate.
219  Automatic tester

Author(s): David Trallero Mena.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.0 (2003/10/16, 11:52:43 CEST)

This module have been created to automatize the test that a predicate should pass hopefully. With that intention we have to provide a set of test and its correct answers. The predicate `run_tester/10` will execute every test and compare it with its answer, generating two traces, one with detailed information, and another with the summary of executions of the tests.

219.1 Usage and interface (tester)

- Library usage:
  ```prolog
  :- use_module(library(tester)).
  ```

- Exports:
  - Predicates:
    ```prolog
    run_tester/10.
    ```

- Other modules used:
  - System library modules:
    ```prolog
    lists, write, io_alias_redirection.
    ```

219.2 Documentation on exports (tester)

`run_tester/10`:

*Meta-predicate* with arguments:

```prolog
run_tester(? ,?,pred(0),pred(1),? ,?,pred(1),? ,pred(0),? ,?).
```

**Usage:**

`run_tester(LogFile, ResultFile, Begin, Test, TestList, Check, CheckList, End, GoodExamples, Slider)`

- **Description:** `run_tester` is a predicate for automatize testers. It get 2 file names as entry (LogFile and ResultFile) for saving the trace and the short result scheme respectively. **Begin** and **End** are called at the beginning and at the end of the test. **Test** is called which each element of **TestList** and after, **Check** is called with the corresponding element in **CheckList** for checking the results of **Test** predicate. **GoodExample** is ground(int) at the exit and tells the number of examples that passed the test correctly. **Slider** can take the values slider(no) or slider(Title) and slider will be shown everytime a new test is called.

- **The following properties should hold at call time:**
  - **LogFile** is a string (a list of character codes).
  - **ResultFile** is a string (a list of character codes).
  - **Begin** is a term which represents a goal, i.e., an atom or a structure.
  - **Test** is a term which represents a goal, i.e., an atom or a structure.
  - **TestList** is a list.
Check is a term which represents a goal, i.e., an atom or a structure.  (basic_props:callable/1)
CheckList is a list.  (basic_props:list/1)
End is a term which represents a goal, i.e., an atom or a structure.  (basic_props:callable/1)
GoodExamples is a free variable.  (term_typing:var/1)
Slider is any term.  (basic_props:term/1)

219.3 Other information (tester)

Two simple examples of the use of the run_tester are provided.

219.3.1 Understanding run_test predicate

```prolog
:- module( tester_test2 , _ , _ ).

:- use_module( '../tester' ).
%:- use_module( library(tester) ).
:- use_module( library(lists) ).
:- use_module( library(write) ).

init_func :-
    write( 'Starting the test\n' ).

tester_func( (X,X,_) ) :-
    write( 'The argument is correct ' ),
    write( X ) , nl.

checker_func( (_,X,X) ) :-
    write( 'check is fine\n
' ).

end_func :-
    write( 'Test ended\n' ).

main :-
    L = [ (1,1,1), % CORRECT
         (2,2,1), % Test CORRECT , CHECK FALSE
         (1,2,2) % Test FALSE
    ],

run_tester(
    'test.log',
    'result.log',
    init_func ,
    tester_func ,
    L,
    checker_func,
```
219.3.2 More complex example

In this example we just want to test if the output of Ciaopp is readable by CIAO. Tester function succeeds if it is able to write the output file. Checker function succeeds if it is able to load the written file.

```prolog
:- module( tester_test1 , _ , [] ).
%:- use_module( library(tester) , [run_tester/10] ).
:- use_module( '../tester' , [run_tester/10] ).
:- use_module( library(ciaopp) ).
:- use_module( library(compiler) ).
:- use_module(library(filenames)).
:- use_module( library( write ) ).
:- use_module( library( lists ) ).
init_func.
test_files( '/home/dtm/Ciaopp/Benchmarks/ciaopp/modes/' ).
tester_func( FileArg ) :-
    test_files( Path ),
    atom_concat( Path , FileArg , File0 ),
    message( note , 
        [ '+++++++++++++++++++++++++++++++++++++++++++++++
          
        (unload( File0 )->true;true),
        module( File0 ),
        atom_concat( TFile , '.pl', File0 ),
        atom_concat( TFile , '_test.pl' , TestFile ),
        output( TestFile ).
```
get_module( Path , Module ) :-
   no_path_file_name( Path , File ),
   (atom_concat( Module , '.pl' , File )
     -> true ; Module = File ).

checker_func( FileArg ) :-
   get_module( FileArg , Module ),
   (unload( Module )->true;true),
   atom_concat(RawFile, '.pl' , FileArg ),
   atom_concat(RawFile, '_test.pl' , OptFile ),
   test_files( Path ),
   atom_concat( Path , OptFile , OptFilePath ),
   message( note , [ 'Cargando ' , OptFilePath ] ),
   use_module( OptFilePath ).

end_func.

main :-
   L = [
      'aiakl.pl',
      'query.pl',
      'mmatrix.pl',
      'ann.pl',
      'bid.pl',
      'rdtok.pl',
      'myread.pl',
      'boyer.pl',
      'read.pl',
      'occur.pl',
      'serialize.pl',
      'browse.pl',
      'peephole.pl',
      'tak.pl',
      'deriv.pl',
      'progeom.pl',
      'warplan.pl',
      'fib.pl',
      'qplan.pl',
      'witt.pl',
      'grammar.pl',
      'zebra.pl',
      'qsortapp.pl',
      'hanoiapp.pl'
   ],
run_tester(
    'test.log',
    'result.log',
    init_func,
    tester_func,
    L,
    checker_func,
    L,
    end_func,
    Res,
    slider( 'Tester1: ' )
  ),
length( L , LL ),
Op is (Res / LL) * 100,
PART XII - Appendices

**Author(s):** The CLIP Group.

These appendices describe the installation of the Ciao environment on different systems and some other issues such as reporting bugs, signing up on the Ciao user’s mailing list, downloading new versions, limitations, etc.
220 Installing Ciao from the source distribution

Author(s): Manuel Carro, Daniel Cabeza, Manuel Hermenegildo.

Version: 1.11#222 (2004/5/24, 13:8:7 CEST)

Version of last change: 1.7#130 (2001/10/28, 17:6:47 CET)

This describes the installation procedure for the Ciao Prolog system, including libraries and manuals, from a source distribution. This applies primarily to Unix-type systems (Linux, Mac OS X, Solaris, SunOS, etc.). However, the sources can also be compiled if so desired on Windows systems – see Section 220.6 [Installation and compilation under Windows], page 902 for details.

If you find any problems during installation, please refer to Section 220.8 [Troubleshooting (nasty messages and nifty workarounds)], page 904. See also Section 222.3 [Downloading new versions], page 911 and Section 222.4 [Reporting bugs], page 912.

220.1 Un*x installation summary

Note: it is recommended that you read the full installation instructions (specially if the installation will be shared by different architectures). However, in many cases it suffices to follow this summary:

1. Uncompress and unpackage (using gunzip and tar -xpf) the distribution. This will put everything in a new directory whose name reflects the Ciao version.

2. Enter the newly created directory (SRC). Edit SETTINGS and check/set the variables SRC, BINROOT (where the executables will go), LIBROOT (where the libraries will go), and DOCROOT (where the documentation will go, preferably a directory accessible via WWW).

3. Type gmake install. This will build executables, compile libraries, and install everything in a directory LIBROOT/ciao and in BINROOT.

Note that gmake refers to the GNU implementation of the make Un*x command, which is available in many systems (including all Linux systems and Mac OS X) simply as make. I.e., you can try simply typing make install if gmake install does not work. If typing make stops right away with error messages it is probably an older version and you need to install gmake.

4. Make the following modifications in your startup scripts. This will make the documentation accessible, set the correct mode when opening Ciao source files in emacs, etc. Note that <LIBROOT> must be replaced with the appropriate value:

   • For users a csh-compatible shell (csh, tcsh, ...), add to ~/.cshrc:

     ```
     if ( -e <LIBROOT>/ciao/DOTcshrc ) then
       source <LIBROOT>/ciao/DOTcshrc
     endif
     ```

     Mac OS X users should add (or modify) the path file in the directory ~/Library/init/tcsh, adding the lines shown above. Note: while this is recognized by the terminal shell, and therefore by the text-mode Emacs which comes with Mac OS X, the Aqua native Emacs 21 does not recognize that initialization. It is thus necessary, at this moment, to set manually the Ciao shell (ciaosh) and Ciao library location by hand. This can be done from the Ciao menu within Emacs after a Ciao Prolog file has been loaded. We suppose that the reason is that Mac OS X does not actually consult the per-user initialization files on startup. It should also be possible to put the right initializations in the .emacs file using the setenv function of Emacs-lisp, as in

     ```
     (setenv "CIAOLIB" "<LIBROOT>/ciao")
     ```

     The same can be done for the rest of the variables initialized in <LIBROOT>/ciao/DOTcshrc

   • For users of an sh-compatible shell (sh, bash, ...), add to ~/.profile:
The Ciao Prolog System

```bash
if [ -f <LIBROOT>/ciao/DOTprofile ]; then
  . <LIBROOT>/ciao/DOTprofile
fi
```

This will set up things so that the Ciao executables are found and you can access the Ciao system manuals using the `info` command. Note that, depending on your shell, you may have to log out and back in for the changes to take effect.

- Also, if you use `emacs` (highly recommended) add this line to your `~/.emacs` file:
  ```lisp
  (load-file "<LIBROOT>/ciao/DOTemacs.el")
  ```

If you are installing Ciao globally in a multi-user machine, make sure that you instruct all users to do the same. If you are the system administrator, the previous steps can be done once and for all, and globally for all users by including the lines above in the central startup scripts (e.g., in Linux `/etc/bashrc`, `/etc/csh.login`, `/etc/csh.cshrc`, `/etc/skel`, `/usr/share/emacs/.../lisp/site-init.pl`, etc.).

5. Finally, if the (freely available) `emacs` editor/environment is not installed in your system, we highly recommend that you also install it at this point (see Section 220.2 [Un*x full installation instructions], page 898 for instructions). While it is easy to use Ciao with any editor of your choice, the Ciao distribution includes a very powerful application development environment which is based on `emacs` and which enables, e.g., source-level debugging, syntax coloring, context-sensitive on-line help, etc.

6. You may want now to check your installation (see Section 220.3 [Checking for correct installation on Un*x], page 901) and read the documentation, which is stored in `DOCROOT` (copied from `SRC/doc/reference`) and can be easily accessed as explained in that same section. There are special “getting started” sections at the beginning of the manual.

7. If you have any problems you may want to check the rest of the instructions. The system can be `uninstalled` by typing `gmake uninstall`.

### 220.2 Un*x full installation instructions

1. **Uncompress and unpack** (using `gunzip` and `tar -xpf`) the distribution in a suitable directory. This will create a new directory called `ciao-X.Y`, where `X.Y` is the version number of the distribution. The `-p` option in the `tar` command ensures that the relative dates of the files in the package are preserved, which is needed for correct operation of the Makefiles.

2. **Select installation options**: Edit the file `SETTINGS` and set the following variables:
   - **SRC**: directory where the sources are stored.
   - **BINROOT**: directory where the Ciao executables will go. For example, if `BINROOT=/usr/local/bin`, then the Ciao compiler (`ciaoc`) will be stored at `/usr/local/bin/ciaoc`. Actually, it will be a link to `ciaoc-VersionNumber`. This applies also to other executables below and is done so that several versions of Ciao can coexist on the same machine. Note that the `version installed latest` will be the one started by default when typing `ciao`, `ciaoc`, etc.
   - **LIBROOT**: directory where the run-time libraries will be installed. The Ciao installation procedure will create a new subdirectory `ciao` below `LIBROOT` and a subdirectory below this one for each Ciao version installed. For example, if `LIBROOT=/usr/local/lib` and you have Ciao version `x.y`, then the libraries will be installed under `/usr/local/lib/ciao/ciao-x.y`. This allows you to install site-specific programs under `/usr/local/lib/ciao` and they will not be overwritten if a new version of Ciao is installed. It also again allows having several Ciao versions installed simultaneously.
• **DOCROOT**: directory where the manuals will be installed. It is often convenient if this directory is accessible via WWW (DOCROOT=/home/httpd/html/ciao, or something like that).

For network-based installations, it is of **utmost importance** that the paths given be reachable in all the networked machines. Different machines with different architectures can share the same physical SRC directory during installation, since compilations for different architectures take place in dedicated subdirectories. Also, different machines/architectures can share the same LIBROOT directory. This saves space since the architecture-independent libraries will be shared. See Section 220.5 [Multiarchitecture support], page 902 below.

3. **Compile Ciao**: At the ciao top level directory type *gmake all*. **Important**: use GNU make (gmake), not the standard UNIX make, as the latter does not support some features used during the compilation. It does not matter if the name of the executable is *make* or *gmake*: only make sure that it is GNU make.

This will:
- Build an engine in $(SRC)/bin/$(CIAOARCH), where $(CIAOARCH) depends on the architecture. The engine is the actual interpreter of the low level code into which Ciao Prolog programs are compiled.
- Build a new Ciao standalone compiler (ciaoc), with the default paths set for your local configuration (nonetheless, these can be overridden by environment variables, as described below).
- Precompile all the libraries under $(SRC)/lib and $(SRC)/library using this compiler.
- Compile a toplevel Prolog shell and a shell for Prolog scripts, under the $(SRC)/shell directory.
- Compile some small, auxiliary applications (contained in the etc directory, and documented in the part of the manual on 'Miscellaneous Standalone Utilities').

This step can be repeated successively for several architectures in the same source directory. Only the engine and some small parts of the libraries (those written in C) differ from one architecture to the other. Standard Ciao Prolog code compiles into bytecode object files (.po) and/or executables which are portable among machines of different architecture, provided there is an executable engine accessible in every such machine. See more details below under Section 220.5 [Multiarchitecture support], page 902.

4. **Check compilation**: If the above steps have been satisfactorily finished, the compiler has compiled itself and all the distribution modules, and very probably everything is fine.

5. **Install Ciao**: To install Ciao in the directories selected in the file SETTINGS during step 2 above, type *gmake justinstall*. This will:
- Install the executables of the Ciao program development tools (i.e., the general driver/top-level ciao, the standalone compiler ciaoc, the script interpreter ciao-shell, miscellaneous utilities, etc.) in BINROOT (see below). In order to use these tools, the PATH environment variable of users needs to contain the path BINROOT.
- Install the Ciao libraries under LIBROOT/ciao (these will be automatically found).
- Install under DOCROOT the Ciao manuals in several formats (such as GNU info, html, postscript, etc.), depending on the distribution. In order for these manuals to be found when typing M-x info within emacs, or by the standalone info and man commands, the MANPATH and INFOPATH environment variables of users both need to contain the path DOCROOT.
- Install under LIBROOT/ciao the Ciao GNU emacs interface (ciao.el, which provides an interactive interface to the Ciao program development tools, as well as some other auxiliary files) and a file DOTemacs containing the emacs initialization commands which are needed in order to use the Ciao emacs interface.
6. **Set up user environments:** In order to automate the process of setting the variables above, the installation process leaves the files `LIBROOT/ciao/DOTcshrc` (for `csh`-like shells), `LIBROOT/ciao/DOTprofile` (for `sh`-like shells), and `LIBROOT/ciao/DOTemacs` (for `emacs`) with appropriate definitions which will take care of all needed environment variable definitions and `emacs` mode setup. Make the following modifications in your startup scripts, so that these files are used (`<LIBROOT>` must be replaced with the appropriate value):

- For users a *csh-compatible shell* (`csh`, `tcsh`, ...), add to `~/.cshrc`:
  ```bash
  if ( -e <LIBROOT>/ciao/DOTcshrc ) then
    source <LIBROOT>/ciao/DOTcshrc
  endif
  ```
  Mac OS X users should add (or modify) the `path` file in the directory `~/Library/init/tcsh`, adding the lines shown above. **Note:** while this is recognized by the terminal shell, and therefore by the text-mode `Emacs` which comes with Mac OS X, the Aqua native `Emacs` 21 does not recognize that initialization. It is thus necessary, at this moment, to set manually the Ciao shell (`ciaosh`) and Ciao library location by hand. This can be done from the Ciao menu within `Emacs` after a Ciao Prolog file has been loaded. We suppose that the reason is that Mac OS X does not actually consult the per-user initialization files on startup. It should also be possible to put the right initializations in the `.emacs` file using the `setenv` function of Emacs-lisp, as in
  ```lisp
  (setenv "CIAOLIB" "<LIBROOT>/ciao")
  ```
  This can be done for the rest of the variables initialized in `<LIBROOT>/ciao/DOTcshrc`

- For users of an *sh-compatible shell* (`sh`, `bash`, ...), add to `~/.profile`:
  ```bash
  if [ -f <LIBROOT>/ciao/DOTprofile ]; then
    . <LIBROOT>/ciao/DOTprofile
  fi
  ```
  This will set up things so that the Ciao executables are found and you can access the Ciao system manuals using the `info` command. Note that, depending on your shell, you may have to log out and back in for the changes to take effect.

- Also, if you use `emacs` (highly recommended) add this line to your `~/.emacs` file:
  ```lisp
  (load-file "<LIBROOT>/ciao/DOTemacs.el")
  ```

If you are installing Ciao globally in a multi-user machine, make sure that you instruct all users to do the same. If you are the system administrator, the previous steps can be done once and for all, and globally for all users by including the lines above in the central startup scripts (e.g., in Linux `/etc/bashrc`, `/etc/csh.login`, `/etc/csh.cshrc`, `/etc/skel`, `/usr/share/emacs/.../lisp/site-init.pl`, etc.).

7. **Download and install Emacs (highly recommended):** If the (freely available) `emacs` editor is not installed in your system, its installation is *highly recommended* (if you are installing in a multi-user machine, you may want to do it in a general area so that it is available for other users, even if you do not use it yourself). While it is easy to use Ciao with any editor of your choice, the Ciao distribution includes a very powerful *application development environment* which is based on `emacs` and which enables, e.g., source-level debugging, syntax coloring, context-sensitive on-line help, etc.

The `emacs` editor (in all its versions: Un*x, Windows, etc.) can be downloaded from, for example, [http://www.emacs.org/](http://www.emacs.org/), and also from the many GNU mirror sites worldwide (See [http://www.gnu.org/](http://www.gnu.org/) for a list), in the `gnu/emacs` and `gnu/windows/emacs` directories. You can find answers to frequently asked questions (FAQ) about `emacs` in general at [http://www.gnu.org/software/emacs/emacs-faq.text](http://www.gnu.org/software/emacs/emacs-faq.text) and about the Windows version at [http://www.gnu.org/software/emacs/windows/ntemacs.html](http://www.gnu.org/software/emacs/windows/ntemacs.html) (despite the `ntemacs` name it runs fine also as is on Win9X and Win2000 machines).
Chapter 220: Installing Ciao from the source distribution

8. **Check installation / read documentation:** You may now want to check your installation (see Section 220.3 [Checking for correct installation on Un*x], page 901) and read the documentation, which is stored in DOCROOT (copied from SRC/doc/reference) and can be easily accessed as explained that same section. There are special “getting started” sections at the beginning of the manual.

   Other useful make targets are listed at the beginning of $(SRC)/Makefile.

   If you have any problems you may want to check Section 220.8 [Troubleshooting (nasty messages and nifty workarounds)], page 904.

   The system can be **uninstalled** by typing `gmake uninstall` in the top directory (the variables in SETTINGS should have the same value as when the install was performed, so that the same directories are cleaned).

220.3 Checking for correct installation on Un*x

   If everything has gone well, several applications and tools should be available to a normal user. Try the following while logged in as a *normal user* (important in order to check that permissions are set up correctly):
   
   - Typing `ciao` (or `ciaosh`) should start the typical Prolog top-level shell.
   - In the top-level shell, Prolog library modules should load correctly. Type for example `use_module(library(dec10_io))` – you should get back a prompt with no errors reported.
   - To exit the top level shell, type `halt`. as usual, or `CTRL-D`.
   - Typing `ciaoc` should produce the help message from the Ciao standalone compiler.
   - Typing `ciao-shell` should produce a message saying that no code was found. This is a Ciao application which can be used to write scripts written in Prolog, i.e., files which do not need any explicit compilation to be run.

   Also, the following documentation-related actions should work:
   
   - If the `info` program is installed, typing `info` should produce a list of manuals which **should include Ciao manual(s) in a separate area** (you may need to log out and back in so that your shell variables are reinitialized for this to work).
   - Opening with a WWW browser (e.g., `netscape`) the directory or URL corresponding to the DOCROOT setting should show a series of Ciao-related manuals. Note that style sheets should be activated for correct formatting of the manual.
   - Typing `man ciao` should produce a man page with some very basic general information on Ciao (and pointing to the on-line manuals).
   - The DOCROOT directory should contain the manual also in the other formats such as `postscript` or `pdf` which specially useful for printing. See Section 2.3.7 [Printing manuals (Un*x)], page 20 for instructions.

   Finally, if `emacs` is installed, after starting it (typing `emacs`) the following should work:
   
   - Typing `CTRL-0` (or in the menus Help->Manuals->Browse Manuals with Info) should open a list of manuals in info format in which the Ciao manual(s) should appear.
   - When opening a Prolog file, i.e., a file with `.pl` or `.pls` ending, using `CTRL-X CTRL-F` `filename` (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and Ciao/Prolog menus should appear in the menu bar on top of the `emacs` window.
   - Loading the file using the Ciao/Prolog menu (or typing `CTRL-0`) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the toplevel shell and make queries from within `emacs`.

   **Note:** when using `emacs` it is **very convenient** to swap the locations of the (normally not very useful) `Caps Lock` key and the (very useful in `emacs`) `CTRL` key on the keyboard. How to do this is explained in the `emacs` frequently asked questions FAQs (see the `emacs` download instructions for their location).
220.4 Cleaning up the source directory

After installation, the source directory can be cleaned up in several ways:

- `gmake uninstall` removes the installation but does not touch the source directories.
- `gmake totalclean` leaves the distribution in its original form, throwing away any intermediate files (as well as any unneeded files left behind by the Ciao developers), while still allowing recompilation.

Other useful `make` targets are listed at the beginning of `$SRC/Makefile`.

220.5 Multiarchitecture support

As mentioned before, Ciao applications (including the compiler and the top level) can run on several machines with different architectures without any need for recompiling, provided there is one Ciao engine (compiled for the corresponding architecture) accessible in each machine. Also, the Ciao libraries (installed in LIBROOT, which contain also the engines) and the actual binaries (installed in BINROOT) can themselves be shared on several machines with different architectures, saving disk space.

For example, assume that the compiler is installed as:

```
/usr/local/share/bin/ciaoc
```

and the libraries are installed under

```
/usr/local/share/lib
```

Assume also that the `/usr/local/share` directory is mounted on, say, a number of Linux and a number of Solaris boxes. In order for `ciaoc` to run correctly on both types of machines, the following is needed:

1. Make sure you that have done `gmake install` on one machine of each architecture (once for Linux and once for Solaris in our example). This recompiles and installs a new engine and any architecture-dependent parts of the libraries for each architecture. The engines will have names such as `ciaoengine.LINUXi86`, `ciaoengine.SolarisSparc`, and so on.

2. In multi-architecture environments it is even more important to make sure that users make the modifications to their startup scripts using `<LIBROOT>/ciao/DOTcshrc` etc. The selection of the engine (and architecture-dependent parts of libraries) is done in these scripts by setting the environment variable `CIAOARCH`, using the `ciao_get_arch` command, which is installed automatically when installing Ciao. This will set `CIAOARCH` to, say, `LINUXi86`, `SolarisSparc`, respectively, and `CIAOENGINE` will be set to `ciaoengine.CIAOARCH`.

However, note that this is not strictly necessary if running on only one architecture: if `CIAOARCH` is not set (i.e., undefined), the Ciao executables will look simply for `ciaoengine`, which is always a link to the latest engine installed in the libraries. But including the initialization files provided has the advantage of setting also paths for the manuals, etc.

220.6 Installation and compilation under Windows

There are two possibilities in order to install Ciao Prolog on Windows machines:

- Installing from the Windows `precompiled` distribution. This is the easiest since it requires no compilation and is highly recommended. This is described in Chapter 221 [Installing Ciao from a Win32 binary distribution], page 907.

- Installing the standard Ciao Prolog (`Un*X`) system source distribution and compiling it under Windows. This is somewhat more complex and currently requires the (freely available) Cygnus Win32 development libraries — described below.
In order to compile Ciao Prolog for Win32 environments you need to have the (public domain) 
_Cygnus Win32_ and development libraries installed in your system. Compilation should be 
performed preferably under Windows NT-type systems.

- Thus, the first step, if Cygnus Win32 is not installed in your system, is to download it 
  (from, e.g., _http://www.cygnus.com/misc/gnu-win32_) and install it. The compilation 
  process also requires that the executables `rm.exe`, `sh.exe`, and `uname.exe` from the Cygnus 
  distribution be copied under `/bin` prior to starting the process (if these executables are not 
  available under `/bin` the compilation process will produce a number of errors and eventually 
  stop prematurely).

- Assuming all of the above is installed, type `make allwin32`. This will compile both the 
  engine and the Prolog libraries. In this process, system libraries that are normally linked 
  dynamically under Un*x (i.e., those for which `.so` dynamically loadable files are generated) 
  are linked statically into the engine (this is done instead of generating `.dll`s because of a 
  limitation in the current version of the Cygnus Win32 environment). No actual installation 
  is made at this point, i.e., this process leaves things in a similar state as if you had just 
  downloaded and uncompressed the precompiled distribution. Thus, in order to complete 
  the installation you should now:

- Follow now the instructions in Chapter 221 [Installing Ciao from a Win32 binary distribu- 
  tion], page 907.

A further note regarding the executables generated by the Ciao compiler and top-level: the 
same considerations given in Chapter 221 [Installing Ciao from a Win32 binary distribu- 
]on, page 907 apply regarding `.bat` files, etc. However, in a system in which Cygnus Win32 is 
installed these executables can also be used in a very simple way. In fact, the executables 
can be run as in Un*x by simply typing their name at the `bash` shell command line without any 
associated `.bat` files. This only requires that the `bash` shell which comes with Cygnus Win32 
be installed and accessible: simply, make sure that `/bin/sh.exe` exists.

### 220.7 Porting to currently unsupported operating systems

If you would like to port Ciao to a currently unsupported platform, there are several issues 
to take into account. The main one is to get the `engine` to compile in that platform, i.e., the 
C code under the `engine` directory. The procedure currently followed by Ciao to decide the 
various flags needed to compile is as follows:

- The shell script `$(SRC)/etc/ciao_get_arch` is executed; it returns a string describing the 
  operating system and the processor architecture (e.g., LINUXi86, SolarisSparc, SolarisAl- 
  pha, etc.). You should make sure it returns a correct (and meaningful) string for your setup. 
  This string is used throughout the compilation to create several architecture-dependant 
  flags.

- In the directory `$(SRC)/makefile-sysdep` there are files called `mkf-<OS>-<ARCH>` for ev- 
  ery combination of operating system and architecture in which Ciao is know to (and how 
  to) compile. They set several flags regarding, for example, whether to use or not threads, 
  which threads library to use, the optimization flags to use, the compiler, linker, and it also 
  sets separately the architecture name (`ARCHNAME` variable) and the operating system 
  (`OSNAME`). You should create a new `mkf` file for your machine, starting from the one 
  which is closest to you.

- Most times the porting problems happen in the use of locks and threads. You 
can either disable them, or have a look at the files `$(SRC)/engine/locks.h` and 
`$(SRC)/engine/threads.h`. If you know how to implement native (assembler) locks for 
your architecture, enable `HAVE_NATIVE_SLOCKS` for your architecture and add the def- 
initions. Otherwise, if you have library-based locks, enable them. The mechanism in 
`threads.h` is similar.
Once a working engine is achieved, it should be possible to continue with the standard installation procedure, which will try to use a completely static version of the standalone compiler (ciaoc.sta in the ciaoc directory) to compile the interactive top-level (ciaosh) and a new version of the standalone compiler (ciaoc). These in turn should be able to compile the Prolog libraries. You may also need to look at some libraries (such as, for example, sockets) which contain C code. If you do succeed in porting to a platform that is currently unsupported please send the mkf-CIAOARCH and any patches to ciao@clip.dia.fi.upm.es, and we will include them (with due credit, of course) in the next distribution.

220.8 Troubleshooting (nasty messages and nifty workarounds)

The following is a list of common installation problems reported by users:

- **Problem:** Compilation errors appear when trying a new installation/compilation after the previous one was aborted (e.g., because of errors).
  **Possible reason and solution:** It is a good idea to clean up any leftovers from the previous compilation using `make engclean` before restarting the installation or compilation process.

- **Problem:** During engine compilation, messages such as the following appear: `tasks.c:102:PTHREAD_CANCELASYNCHRONOUS undeclared (first use of this function)`.  
  **Possible reason and solution:** Your (Linux?) system does not have (yet) the Posix threads library installed. You can upgrade to one which does have it, or download the library from [http://pauillac.inria.fr/~xleroy/linuxthreads/index.html](http://pauillac.inria.fr/~xleroy/linuxthreads/index.html) and install it, or disable the use of threads in Linux: for this, edit the SETTINGS file and specify `USE_THREADS=no`, which will avoid linking against thread libraries (it will disable the use of thread-related primitives as well). Clean the engine with `make engclean` and restart compilation.

If you have any alternative threads library available, you can tinker with `engine/threads.h` and the files under `makefile-sysdep` in order to get the task managing macros right for your system. Be sure to link the right library. If you succeed, we (ciao@clip.dia.fi.upm.es) will be happy of knowing about what you have done.

- **Problem:** `-lpthread: library not found` (or similar)  
  **Possible reason and solution:** Your (Linux?) system seems to have Posix threads installed, but there is no threads library in the system. In newer releases (e.g., RedHat 5.0), the Posix threads system calls have been included in `glibc.so`, so specifying `-lpthread` in `makefile-sysdep/mkf-LINUX` is not needed; remove it. `make engclean` and restart installation.

Alternatively, you may have made a custom installation of Posix threads in a non-standard location: be sure to include the flag `-L/this/is/where/the/posix/libraries/are before -lpthread`, and to update `/etc/ld.so.conf` (see `man ldconfig`).

- **Problem:** Segmentation Violation (when starting the first executable)  
  **Possible reason and solution:** This has been observed with certain older versions of gcc which generated erroneous code under full optimization. The best solution is to upgrade to a newer version of gcc. Alternatively, lowering the level of optimization (by editing the SETTINGS file in the main directory of the distribution) normally solves the problem, at the cost of reduced execution speed.
Problem: ciaoc: /home/clip/lib/ciao/ciao-X.Y/engine/ciaoengine: not found
Possible reason and solution:
- The system was not fully installed and the variable CIAOENGINE was not set.
- The system was installed, the variable CIAOENGINE is set, but it is does not point to a valid ciaoengine.
See the file LIBROOT/ciao/DOTcshrc for user settings for environment variables.

Problem:
ERROR: File library(compiler) not found - aborting... (or any other library is not found)
Possible reason and solution:
- The system was not installed and the variable CIAOLIB was not set.
- The system is installed and the variable CIAOLIB is wrong.
See the file LIBROOT/ciao/DOTcshrc for user settings for environment variables.

Problem:
ERROR: File <some_directory>/<some_file>.itf not found - aborting...
Possible reason and solution:
Can appear when compiling .pl files. The file to compile (<some_file>.pl) is not in the directory <some_directory>. You gave a wrong file name or you are in the wrong directory.

Problem:
*ERROR*: /(write_option,1) is not a regular type (and similar ones)
Possible reason and solution:
This is not a problem, but rather the type checker catching some minor inconsistencies which may appear while compiling the libraries. Bug us to remove it, but ignore it for now.

Problem:
WARNING: Predicate <some_predicate>/<N> undefined in module <some_module>
Possible reason and solution:
It can appear when the compiler is compiling Ciao library modules. If so, ignore it (we will fix it). If it appears when compiling user programs or modules, you may want to check your program for those undefined predicates.

Problem:
gmake[1]: execve: /home/clip/mcarro/ciao-0.7p2/etc/collect_modules: No such file or directory
Possible reason and solution:
Check if collect_modules is in $(SRC)/etc and is executable. If it is not here, your distribution is incorrect: please let us know.

Problem:
make: Fatal error in reader: SHARED, line 12: Unexpected end of line seen
Possible reason and solution:
You are using standard Un*x make, not GNU’s make implementation (gmake).

Problem:
WARNINGs or ERRORs while compiling the Ciao libraries during installation.
Possible reason and solution:
It is possible that you will see some such errors while compiling the Ciao libraries during installation. This is specially the case if you are installing a Beta or Alpha release of Ciao. These releases (which have “odd” version numbers such as 1.5 or 2.1) are typically snapshots.
of the development directories, on which many developers are working simultaneously, which may include libraries which have typically not been tested yet as much as the "official" distributions (those with "even" version numbers such as 1.6 or 2.8). Thus, minor warnings may not have been eliminated yet or even errors can sneak in. These warnings and errors should not affect the overall operation of the system (e.g., if you do not use the affected library).
221 Installing Ciao from a Win32 binary distribution

Author(s): Daniel Cabeza, Manuel Carro, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.5#92 (2000/3/28, 17:41:25 CEST)

This describes the installation of Ciao after downloading the Windows binary (i.e., precompiled) distribution. It includes the installation of libraries and manuals and applies to Windows 95/98/NT/2000/XP systems. This is the simplest Windows installation, since it requires no compilation and is highly recommended. However, it is also possible to compile Ciao from the source distribution on these systems (please refer to Chapter 220 [Installing Ciao from the source distribution], page 897 for details).

If you find any problems during installation, please refer to Section 220.8 [Troubleshooting (nasty messages and nifty workarounds)], page 904. See also Section 222.3 [Downloading new versions], page 911 and Section 222.4 [Reporting bugs], page 912.

221.1 Win32 binary installation summary

Please follow these steps (below we use the terms folder and directory interchangeably):

1. Download the precompiled distribution and unpack it into any suitable folder, such as, e.g., C:\Program Files.
   This will create there a folder whose name reflects the Ciao version. Due to limitations of Windows related to file associations, do not put Ciao too deep in the folder hierarchy. For unpacking you will need a recent version of a zip archive manager – there are many freely available such as WinZip, unzip, pkunzip, etc. (see for example www.winzip.com). Some users have reported some problems with version 6.2 of WinZip, but no problems with, e.g., version 7. With WinZip, simply click on “Extract” and select the extraction folder as indicated above.

2. Stop any Ciao-related applications.
   If you have a previous version of Ciao installed, make sure you do not have any Ciao applications (including, e.g., a toplevel shell) running, or the extraction process may not be able to complete. You may also want to delete the entire folder of the previous installation to save space.

3. Open the Ciao source directory created during extraction and run (e.g. by double-clicking on it) the install(.bat) script. Answer “yes” to the dialog that pops up and type any character in the installation window to finish the process. You may need to reboot for the changes in the registry to take effect.
   This will update the windows registry (the file ciao(.reg) lists the additions) and also create some .bat files which may be useful for running Ciao executables from the command line. It also creates initialization scripts for the emacs editor. The actions performed by the installation script are reported in the installation window.

4. You may want to add a windows shortcut in a convenient place, such as the desktop, to\ciaosh.cpx, the standard interactive toplevel shell. It is located inside the shell folder (e.g., click on the file ciaosh.cpx with the right mouse button and select the appropriate option, Send to->Desktop as shortcut).

5. You may also want to add another shortcut to the file ciao(.html) located inside doc\reference\ciao_html so that you can open the Ciao manual by simply double-clicking on this shortcut.

6. Finally, if the (freely available) emacs editor/environment is not installed in your system, we highly recommend that you also install it at this point. While it is easy to use Ciao with any editor of your choice, the Ciao distribution includes a very powerful application...
development environment which is based on emacs and which enables, e.g., source-level debugging, syntax coloring, context-sensitive on-line help, etc. If you are not convinced, consider that many programmers inside Microsoft use emacs for developing their programs. The emacs editor (in all its versions: Unix, Windows, etc.) can be downloaded from, for example, http://www.emacs.org/, and also from the many GNU mirror sites worldwide (See http://www.gnu.org/ for a list), in the gnu/emacs and gnu/windows/emacs directories. You can find answers to frequently asked questions (FAQ) about emacs in general at http://www.gnu.org/software/emacs/emacs-faq.text and about the Windows version at http://www.gnu.org/software/emacs/windows/ntemacs.html (despite the ntemacs name it runs fine also as is on Win9X and Win2000 machines).

You need to tell emacs how to load the Ciao mode automatically when editing and how to access the on-line documentation:

- Start emacs (double click on the icon or from the Start menu). Open (menu Files->Open File or simply \(\texttt{Alt} \texttt{X} \texttt{ih} \texttt{F} \texttt{i}\)) the file ForEmacs.txt that the installation script has created in directory where you installed the Ciao distribution.
- Copy the lines in the file (select with the mouse and then menu Edit->Copy). Open/Create using emacs (menu Files->Open File or simply \(\texttt{Alt} \texttt{X} \texttt{ih} \texttt{F} \texttt{i}\)) the file \(\sim/.emacs\) (or, if this fails, c:/.emacs).
- Paste the two lines (menu Edit->Paste or simply \(\texttt{Alt} \texttt{Y} \texttt{i}\)) into the file and save (menu Files->Save Buffer or simply \(\texttt{Alt} \texttt{X} \texttt{ih} \texttt{S}\)).
- Exit emacs and start it again.

emacs should not report any errors (at least related to Ciao) on startup. At this point the emacs checks in the following section should work.

221.2 Checking for correct installation on Win32

After the actions and registry changes performed by the installation procedure, you should check that the following should work correctly:

- Ciao-related file types (.pl source files, .cpx executables, .itf,.po,.asr interface files, .pls scripts, etc.) should have specific icons associated with them (you can look at the files in the folders in the Ciao distribution to check).
- Double-clicking on the shortcut to ciaosh(.cpx) on the desktop should start the typical Prolog top-level shell in a window. If this shortcut has not been created on the desktop, then double-clicking on the ciaosh(.cpx) icon inside the shell folder within the Ciao source folder should have the same effect.
- In the top-level shell, Prolog library modules should load correctly. Type for example use_module(library(dec10_io)). at the Ciao top-level prompt –you should get back a prompt with no errors reported.
- To exit the top level shell, type halt. as usual, or \(\texttt{Alt} \texttt{D}\).

Also, the following documentation-related actions should work:

- Double-clicking on the shortcut to ciao(.html) which appears on the desktop should show the Ciao manual in your default WWW browser. If this shortcut has not been created you can double-click on the ciao(.html) file in the doc\reference\ciao_html folder inside the Ciao source folder. Make sure you configure your browser to use style sheets for correct formatting of the manual (note, however, that some older versions of Explorer did not support style sheets well and will give better results turning them off).
- The doc\reference folder contains the manual also in the other formats present in the distribution, such as info (very convenient for users of the emacs editor/program development system) and postscript or pdf, which are specially useful for printing. See Section 3.2.7 [Printing manuals (Win32)], page 25 for instructions.
Finally, if **emacs** is installed, after starting it (double-clicking on the **emacs** icon or from the **Start** menu) the following should work:

- Typing `h^Hhi` (or in the menus **Help->Manuals->Browse Manuals with Info**) should open a list of manuals in info format in which the Ciao manual(s) should appear.
- When opening a Prolog file, i.e., a file with `.pl` or `.pls` ending, using `hXhi` filename (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and Ciao/Prolog menus should appear in the menu bar on top of the **emacs** window.
- Loading the file using the Ciao/Prolog menu (or typing `hC`i) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the toplevel shell and make queries from within **emacs**.

**Note:** when using **emacs** it is very convenient to swap the locations of the (normally not very useful) `Caps Lock` key and the (very useful in **emacs**) `Ctrl` key on the keyboard. How to do this is explained in the **emacs** frequently asked questions FAQs (see the **emacs** download instructions for their location).

If you find that everything works but **emacs** cannot start the Ciao toplevel you may want to check if you can open a normal Windows shell within **emacs** (just do `M-x shell`). If you cannot, it is possible that you are using some anti-virus software which is causing problems. See [http://www.gnu.org/software/emac/windows/faq3.html#anti-virus](http://www.gnu.org/software/emac/windows/faq3.html#anti-virus) for a workaround.

In some Windows versions it is possible that you had to change the first backslashes in the DOTemacs.el file in the Ciao Directory. E.g., assuming you have installed in drive c:, instances of `c:\` need to be changed to `c:/`. For example: `c:\prolog/ciao-1.7p30Win32/shell/ciaosh.bat` should be changed to `c:/prolog/ciao-1.7p30Win32/shell/ciaosh.bat`.

### 221.3 Compiling the miscellaneous utilities under Windows

The **etc** folder contains a number of utilities, documented in the manual in **PART V - Miscellaneous Standalone Utilities**. In the Win32 distribution these utilities are not compiled by the installation process. You can create the executable for each of them when needed by compiling the corresponding .pl file.

### 221.4 Server installation under Windows

If you would like to install Ciao on a server machine, used by several clients, the following steps are recommended:

- Follow the standard installation procedure on the server. When selecting the folder in which Ciao is installed make sure you select a folder that is visible by the client machines. Also make sure that the functionality specified in the previous sections is now available on the server.
- Perform a client installation on each client, by running (e.g., double-click on it) the `client.bat` script. This should update the registry of each client. At this point all the functionality should also be available on the clients.

### 221.5 CGI execution under IIS

The standard installation procedure updates the windows registry so that Ciao executables (ending in `.cpx`) are directly executable as CGIs under Microsoft’s IIS, i.e., so that you make applications written in Ciao available on the WWW (see the pillow library for specific support for this task). In the event you re-install IIS, you probably would lose the entries in the registry which allow this. In that case, processing the file `ciao.reg` produced during the installation (or simply reinstalling Ciao) will add those entries again.
221.6 Uninstallation under Windows

To uninstall Ciao under Windows, simply delete the directory in which you put the Ciao distribution. If you also want to delete the registry entries created by the Ciao installation (not strictly needed) this must currently be done by hand. The installation leaves a list of these entries in the file \texttt{ciao.reg} to aid in this task. Also, all the register entries contain the word \texttt{ciao}. Thus, to delete all Ciao entries, run the application \texttt{regedit} (for example, by selecting \texttt{Run} from the Windows \texttt{Start} menu), search (\texttt{^{F}}) for \texttt{ciao} in all registry entries (i.e., select all of \texttt{Keys}, \texttt{Values}, and \texttt{Data} in the \texttt{Edit->Find} dialog), and delete each matching key (click on the left window to find the matching key for each entry found).

\texttt{%% Local Variables:} \texttt{%% mode: CIAO} \texttt{%% update-version-comments: "off"} \texttt{%% End:}
222 Beyond installation

Author(s): Manuel Carro, Daniel Cabeza, Manuel Hermenegildo.
Version: 1.11#222 (2004/5/24, 13:8:7 CEST)
Version of last change: 1.7#55 (2001/1/26, 17:36:30 CET)

222.1 Architecture-specific notes and limitations

Ciao makes use of advanced characteristics of modern architectures and operating systems such as multithreading, shared memory, sockets, locks, dynamic load libraries, etc., some of which are sometimes not present in a given system and others may be implemented in very different ways across the different systems. As a result, currently not all Ciao features are available in all supported operating systems. Sometimes this is because not all the required features are present in all the OS flavors supported and sometimes because we simply have not had the time to port them yet.

The current state of matters is as follows:

LINUX: multithreading, shared DB access, and locking working.
Solaris: multithreading, shared DB access, and locking working.
IRIX: multithreading, shared DB access, and locking working.
SunOS 4: multithreading, shared DB access, and locking NOT working.
Win 95/98/NT/2000/XP: multithreading, shared DB access, and locking working. Dynamic linking of object code (C) libraries NOT working.
Mac OS X (Darwin): multithreading, shared DB access, and locking working.

The features that do not work are disabled at compile time.

222.2 Keeping up to date with the Ciao users mailing list

We recommend that you join the Ciao users mailing list (ciao-users@clip.dia.fi.upm.es), in order to receive information on new versions and solutions to problems. Simply send a message to ciao-users-request@clip.dia.fi.upm.es, containing in the body only the word:

subscribe

alone in one line. Messages in the list are strictly limited to issues directly related to Ciao Prolog and your email address will of course be kept strictly confidential. You mail also want to subscribe to the comp.lang.prolog newsgroup.

There is additional info available on the Ciao system, other CLIP group software, publications on the technology underlying these systems, etc. in the CLIP group's WWW site http://clip.dia.fi.upm.es.

222.3 Downloading new versions

Ciao and its related libraries and utilities are under constant improvement, so you should make sure that you have the latest versions of the different components, which can be downloaded from:

http://clip.dia.fi.upm.es/Software
222.4 Reporting bugs

If you still have problems after downloading the latest version and reading the installation instructions you can send a message to ciao-bug@clip.dia.fi.upm.es. Please be as informative as possible in your messages, so that we can reproduce the bug.

- For installation problems we typically need to have the version and patch number of the Ciao package (e.g., the name of the file downloaded), the output produced by the installation process (you can capture it by redirecting the output into a file or cutting and pasting with the mouse), and the exact version of the Operating System you are using (as well as the C compiler, if you took a source distribution).

- For problems during use we also need the Ciao and OS versions and a small example of code which we can run to reproduce the bug.
References


persdb\_sql: SQL Persistent Database Interface. 

Some Paradigms for Visualizing Parallel Execution of Logic Programs. 

[CH95] D. Cabeza and M. Hermenegildo. 
Distributed Concurrent Constraint Execution in the CIAO System. 
In *Proc. of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation* 
Available from http://www.clip.dia.fi.upm.es/

[CH97] D. Cabeza and M. Hermenegildo. 
WWW Programming using Computational Logic Systems (and the PiLLoW/Ciao Library). 

[CH99] D. Cabeza and M. Hermenegildo. 
The Ciao Modular Compiler and Its Generic Program Processing Library. 

[CH00a] D. Cabeza and M. Hermenegildo. 
A New Module System for Prolog. 
In *International Conference on Computational Logic, CL2000*, number 1861 in 

[CH00b] D. Cabeza and M. Hermenegildo. 
The Ciao Modular, Standalone Compiler and Its Generic Program Processing Library. 
In *Special Issue on Parallelism and Implementation of (C)LP Systems*, volume 30(3) 
of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, 
March 2000.

[CH00c] M. Carro and M. Hermenegildo. 
Tools for Constraint Visualization: The VIFID/TRIFID Tool. 
Springer-Verlag, September 2000.

[CH00d] M. Carro and M. Hermenegildo. 
Tools for Search Tree Visualization: The APT Tool. 
Springer-Verlag, September 2000.

Design of a Generic, Homogeneous Interface to Relational Databases. 
The PiLLoW/Ciao Library for INTERNET/WWW Programming using Computational Logic Systems.

The PiLLoW/Ciao Library for INTERNET/WWW Programming using Computational Logic Systems.
Available from http://clement.info.umoncton.ca/~lpnet/

[CLI95] The CLIP Group.
CIAO Compiler: Distributed Execution and Low Level Support Subsystem.

*Programming in Prolog.*
Springer-Verlag, 1981.

[Col78] A. Colmerauer.
Metamorphosis grammars.

[Col82] A. Colmerauer et al.

*Prolog: The Standard.*
Springer-Verlag, 1996.

Co-operating sequential processes.

Cost analysis of logic programs.

Non-Failure Analysis for Logic Programs.

Lower Bound Cost Estimation for Logic Programs.

WebDB: A Database WWW Interface.
Fast and precise regular approximations of logic programs.

The CIAO Multi-Dialect Compiler and System: A Demo and Status Report.
In *Proceedings of the JICSLP’96 Workshop on Parallelism and Implementation Technology*, Computer Science Department, Technical University of Madrid, September 1996.
Available from \htmladdnormallink\tt http://www.clip.dia.fi.upm.es/Projects/COMPULOG/meeting96/papers/PS/clip.ps.gz

The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems.

The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems.
Available from \htmladdnormallink\tt http://www.clip.dia.fi.upm.es/
http://www.clip.dia.fi.upm.es/.

Program Analysis, Debugging and Optimization Using the Ciao System Preprocessor.

Towards CIAO-Prolog – A Parallel Concurrent Constraint System.


An Automatic Documentation Generator for (C)LP – Reference Manual.
Online at \tt http://clip.dia.fi.upm.es/Software/Ciao/.

[HCC95] M. Hermenegildo, D. Cabeza, and M. Carro.
Using Attributed Variables in the Implementation of Concurrent and Parallel Logic
References

Programming Systems.

[Her86] M. Hermenegildo.
An Abstract Machine for Restricted AND-parallel Execution of Logic Programs.

[Her96] M. Hermenegildo.
Writing “Shell Scripts” in SICStus Prolog, April 1996.

[Her99] M. Hermenegildo.
A Documentation Generator for Logic Programming Systems.

[Her00] M. Hermenegildo.
A Documentation Generator for (C)LP Systems.

\&-Prolog and its Performance: Exploiting Independent And-Parallelism.

The \&-Prolog System: Exploiting Independent And-Parallelism.

[Hog84] C.~J. Hogger.
Introduction to Logic Programming.

[Hol90] C. Holzbaur.
Specification of Constraint Based Inference Mechanisms through Extended Unification.

Metastructures vs. Attributed Variables in the Context of Extensible Unification.

[Hol94] C. Holzbaur.
University of Vienna, 1994.

Compilation of Logic Programs for Restricted And-Parallelism.

[Knu84] D. Knuth.
Literate programming.

[Kor85] R. Korf.
Depth-first iterative deepening: an optimal admissible tree search.
A Methodology for Granularity Based Control of Parallelism in Logic Programs.

Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.

*The MU-Prolog 3.2 Reference Manual.*

Adding equations to NU-Prolog.

[Par97] The RADIOWEB Project Partners.

An Assertion Language for Debugging of Constraint Logic Programs.

An Assertion Language for Constraint Logic Programs.

Some Issues in Analysis and Specialization of Modular Ciao-Prolog Programs.

Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks.

*The Art of Prolog.*

[Swe95] Swedish Institute of Computer Science, P.O. Box 1263, S-16313 Spanga, Sweden.

The Andorra Model.
# Library/Module Definition Index

## A
- actmods ........................................... 421
- aggregates ..................................... 177
- andorra .......................................... 407
- andprolog ....................................... 405
- arc_class ........................................ 551
- argnames ................................-------- 397
- arithmetic ...................................... 105
- arrays ............................................ 659
- assertions ....................................... 269
- assertions_props ................................. 277
- assert_write ..................................... 345
- atom2term ....................................... 319
- atomic_basic ..................................... 99
- attributes ...................................... 151

## B
- basic_props ....................................... 79
- basiccontrol ..................................... 73
- basicmodes ...................................... 297
- between .......................................... 235
- bf .................................................. 425
- bltclass ......................................... 749
- boundary ......................................... 817
- build_foreign_interface ......................... 493
- builtin_directives ................................ 77
- button_class ..................................... 535

## C
- callgraph ......................................... 703
- canvas_class ..................................... 529
- chartlib ......................................... 723
- chartlib_errhandle ............................... 751
- checkbutton_class ............................... 537
- CiaoMode ........................................ 63
- ciaosh ............................................ 37
- class .............................................. 447
- clpq ............................................... 429
- clpr ............................................... 431
- color_pattern .................................... 753
- compiler ......................................... 233
- conc_aggregates .................................. 361
- concurrency ...................................... 355
- counters ......................................... 661
- ctricclean ....................................... 321

## D
- data_facts ........................................ 137
- davinci .......................................... 497
- db_client_types .................................. 625
- dcg ............................................... 211
- dcg_expansion ................................... 215
- ddlist ............................................ 873
- debugger ......................................... 53
- dec10_io .......................................... 259
- default_predicates ............................... 159
- det_hook_rt ...................................... 411
- dict .............................................. 305
- dictionary ....................................... 819
- dictionary_tree ................................... 821
- dynamic .......................................... 181

## E
- emacs .............................................. 651
- entry_class ...................................... 541
- errhandle ........................................ 323
- error ............................................. 825
- exceptions ....................................... 129
- expansion_tools .................................. 351

## F
- factsdb_rt ....................................... 599
- fastrw ............................................ 325
- fd .................................................. 733
- field_type ...................................... 827
- field_value ...................................... 829
- field_value_check ............................... 831
- file_locks ....................................... 333
- file_utils ....................................... 331
- filenames ........................................ 327
- foreign_compilation ............................. 491
- foreign_interface ............................... 475
- foreign_interface_properties ................... 485
- format ........................................... 217
- freeze ........................................... 415
- functions ........................................ 401
- fuzzy ............................................. 435
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G</strong></td>
<td></td>
</tr>
<tr>
<td>genbar1</td>
<td>757</td>
</tr>
<tr>
<td>genbar2</td>
<td>763</td>
</tr>
<tr>
<td>genbar3</td>
<td>767</td>
</tr>
<tr>
<td>genbar4</td>
<td>771</td>
</tr>
<tr>
<td>generator</td>
<td>833</td>
</tr>
<tr>
<td>generator_util</td>
<td>835</td>
</tr>
<tr>
<td>gengraph1</td>
<td>775</td>
</tr>
<tr>
<td>gengraph2</td>
<td>785</td>
</tr>
<tr>
<td>genmultibar</td>
<td>793</td>
</tr>
<tr>
<td>global</td>
<td>403</td>
</tr>
<tr>
<td>gnuplot</td>
<td>883</td>
</tr>
<tr>
<td>graphs</td>
<td>669</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>H</strong></td>
<td></td>
</tr>
<tr>
<td>hiord_rt</td>
<td>393</td>
</tr>
<tr>
<td>hiordlib</td>
<td>395</td>
</tr>
<tr>
<td>html</td>
<td>569</td>
</tr>
<tr>
<td>http</td>
<td>579</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>I</strong></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>427</td>
</tr>
<tr>
<td>idlists</td>
<td>663</td>
</tr>
<tr>
<td>indexer</td>
<td>391</td>
</tr>
<tr>
<td>internal_types</td>
<td>839</td>
</tr>
<tr>
<td>io</td>
<td>843</td>
</tr>
<tr>
<td>io_alias_redirection</td>
<td>317</td>
</tr>
<tr>
<td>io_aux</td>
<td>147</td>
</tr>
<tr>
<td>io_basic</td>
<td>121</td>
</tr>
<tr>
<td>iso</td>
<td>175</td>
</tr>
<tr>
<td>iso_byte_char</td>
<td>199</td>
</tr>
<tr>
<td>iso_incomplete</td>
<td>207</td>
</tr>
<tr>
<td>iso_misc</td>
<td>205</td>
</tr>
<tr>
<td>isomodes</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>J</strong></td>
<td></td>
</tr>
<tr>
<td>javart</td>
<td>635</td>
</tr>
<tr>
<td>javasock</td>
<td>647</td>
</tr>
<tr>
<td>jtopl</td>
<td>643</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>L</strong></td>
<td></td>
</tr>
<tr>
<td>label_class</td>
<td>543</td>
</tr>
<tr>
<td>lggraphs</td>
<td>679</td>
</tr>
<tr>
<td>libpaths</td>
<td>61</td>
</tr>
<tr>
<td>libbrowser</td>
<td>347</td>
</tr>
<tr>
<td>linda</td>
<td>655</td>
</tr>
<tr>
<td>line_class</td>
<td>561</td>
</tr>
<tr>
<td>lists</td>
<td>223</td>
</tr>
<tr>
<td>loading_code</td>
<td>71</td>
</tr>
<tr>
<td>lookup</td>
<td>847</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>M</strong></td>
<td></td>
</tr>
<tr>
<td>make</td>
<td>369</td>
</tr>
<tr>
<td>make_rt</td>
<td>375</td>
</tr>
<tr>
<td>menu_class</td>
<td>525</td>
</tr>
<tr>
<td>menu_entry_class</td>
<td>547</td>
</tr>
<tr>
<td>menubutton_class</td>
<td>545</td>
</tr>
<tr>
<td>messages</td>
<td>311</td>
</tr>
<tr>
<td>modtester</td>
<td>887</td>
</tr>
<tr>
<td>modules</td>
<td>67</td>
</tr>
<tr>
<td>mycin</td>
<td>721</td>
</tr>
<tr>
<td>mysql_client</td>
<td>623</td>
</tr>
<tr>
<td>native_props</td>
<td>289</td>
</tr>
<tr>
<td>numlists</td>
<td>665</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>O</strong></td>
<td></td>
</tr>
<tr>
<td>objects</td>
<td>459</td>
</tr>
<tr>
<td>objects_rt</td>
<td>465</td>
</tr>
<tr>
<td>odd</td>
<td>413</td>
</tr>
<tr>
<td>old_database</td>
<td>261</td>
</tr>
<tr>
<td>operators</td>
<td>197</td>
</tr>
<tr>
<td>oval_class</td>
<td>555</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P</strong></td>
<td></td>
</tr>
<tr>
<td>parser</td>
<td>853</td>
</tr>
<tr>
<td>parser_util</td>
<td>855</td>
</tr>
<tr>
<td>patterns</td>
<td>667</td>
</tr>
<tr>
<td>persdbrt</td>
<td>589</td>
</tr>
<tr>
<td>persdbrt_mysql</td>
<td>603</td>
</tr>
<tr>
<td>persdbtr_sql</td>
<td>631</td>
</tr>
<tr>
<td>pillow</td>
<td>567</td>
</tr>
<tr>
<td>pillow_types</td>
<td>581</td>
</tr>
<tr>
<td>pl2sql</td>
<td>615</td>
</tr>
<tr>
<td>pl2sqlinsert</td>
<td>633</td>
</tr>
<tr>
<td>poly_class</td>
<td>559</td>
</tr>
<tr>
<td>possible</td>
<td>869</td>
</tr>
<tr>
<td>pretty_print</td>
<td>341</td>
</tr>
<tr>
<td>prolog_flags</td>
<td>131</td>
</tr>
<tr>
<td>prolog_sys</td>
<td>253</td>
</tr>
<tr>
<td>provrml</td>
<td>811</td>
</tr>
<tr>
<td>Package</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>pure</td>
<td>389</td>
</tr>
<tr>
<td>Q</td>
<td>681</td>
</tr>
<tr>
<td>queues</td>
<td>683</td>
</tr>
<tr>
<td>R</td>
<td>283</td>
</tr>
<tr>
<td>radiobutton_class</td>
<td>539</td>
</tr>
<tr>
<td>random</td>
<td>683</td>
</tr>
<tr>
<td>regtypes</td>
<td>187</td>
</tr>
<tr>
<td>remote</td>
<td>471</td>
</tr>
<tr>
<td>runtime_ops</td>
<td>265</td>
</tr>
<tr>
<td>S</td>
<td>685</td>
</tr>
<tr>
<td>sets</td>
<td>549</td>
</tr>
<tr>
<td>shape_class</td>
<td>363</td>
</tr>
<tr>
<td>sockets</td>
<td>367</td>
</tr>
<tr>
<td>sockets_io</td>
<td>231</td>
</tr>
<tr>
<td>sort</td>
<td>627</td>
</tr>
<tr>
<td>sqltypes</td>
<td>303</td>
</tr>
<tr>
<td>streams</td>
<td>111</td>
</tr>
<tr>
<td>streams_basic</td>
<td>307</td>
</tr>
<tr>
<td>strings</td>
<td>329</td>
</tr>
<tr>
<td>symfnames</td>
<td>143</td>
</tr>
<tr>
<td>syntax_extensions</td>
<td>237</td>
</tr>
<tr>
<td>system</td>
<td>377</td>
</tr>
<tr>
<td>system_extra</td>
<td>155</td>
</tr>
<tr>
<td>system_info</td>
<td>801</td>
</tr>
<tr>
<td>table_widget1</td>
<td>797</td>
</tr>
<tr>
<td>table_widget2</td>
<td>803</td>
</tr>
<tr>
<td>table_widget3</td>
<td>805</td>
</tr>
<tr>
<td>table_widget4</td>
<td>501</td>
</tr>
<tr>
<td>tcltk</td>
<td>509</td>
</tr>
<tr>
<td>tcltk_low_level</td>
<td>91</td>
</tr>
<tr>
<td>term_basic</td>
<td>335</td>
</tr>
<tr>
<td>term_compare</td>
<td>337</td>
</tr>
<tr>
<td>term_typing</td>
<td>87</td>
</tr>
<tr>
<td>terms</td>
<td>339</td>
</tr>
<tr>
<td>terms_check</td>
<td>807</td>
</tr>
<tr>
<td>terms_vars</td>
<td>889</td>
</tr>
<tr>
<td>test_format</td>
<td>563</td>
</tr>
<tr>
<td>tester</td>
<td>877</td>
</tr>
<tr>
<td>text_class</td>
<td>871</td>
</tr>
<tr>
<td>time_analyzer</td>
<td>515</td>
</tr>
<tr>
<td>tokenizer</td>
<td>263</td>
</tr>
<tr>
<td>ttyout</td>
<td>673</td>
</tr>
<tr>
<td>U</td>
<td>689</td>
</tr>
<tr>
<td>ugraphs</td>
<td>677</td>
</tr>
<tr>
<td>V</td>
<td>513</td>
</tr>
<tr>
<td>vndict</td>
<td>417</td>
</tr>
<tr>
<td>W</td>
<td>739</td>
</tr>
<tr>
<td>wgraphs</td>
<td>515</td>
</tr>
<tr>
<td>when</td>
<td>743</td>
</tr>
<tr>
<td>widget_class</td>
<td>191</td>
</tr>
<tr>
<td>window_class</td>
<td>739</td>
</tr>
<tr>
<td>write</td>
<td>677</td>
</tr>
<tr>
<td>X</td>
<td>743</td>
</tr>
<tr>
<td>xdr_handle</td>
<td>513</td>
</tr>
<tr>
<td>xml_path</td>
<td>191</td>
</tr>
</tbody>
</table>
## Predicate/Method Definition Index

<table>
<thead>
<tr>
<th>Predicate/Method</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$check_attr/3</td>
<td>416, 419</td>
</tr>
<tr>
<td>$class$/1</td>
<td>513, 533, 549, 559, 561</td>
</tr>
<tr>
<td>$combine_attr/4</td>
<td>416, 419</td>
</tr>
<tr>
<td>$factsdb$cached_goal/3</td>
<td>601</td>
</tr>
<tr>
<td>$is_persistent/2</td>
<td>595</td>
</tr>
<tr>
<td>$meta_call/1</td>
<td>394</td>
</tr>
<tr>
<td>$metachoice/1</td>
<td>75</td>
</tr>
<tr>
<td>$metacut/1</td>
<td>75</td>
</tr>
<tr>
<td>$nodebug_call/1</td>
<td>394</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>411</td>
</tr>
<tr>
<td>!/0</td>
<td>74</td>
</tr>
<tr>
<td>@</td>
<td>96</td>
</tr>
<tr>
<td>@&lt;&lt;/2</td>
<td>96</td>
</tr>
<tr>
<td>@&gt;/2</td>
<td>96</td>
</tr>
<tr>
<td>@&gt;=/2</td>
<td>96</td>
</tr>
<tr>
<td>@&lt;/2</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>76</td>
</tr>
<tr>
<td>1/2</td>
<td>76</td>
</tr>
<tr>
<td>&gt;</td>
<td>106</td>
</tr>
<tr>
<td>&gt;/2</td>
<td>106</td>
</tr>
<tr>
<td>&gt;=/2</td>
<td>107</td>
</tr>
<tr>
<td>&lt;</td>
<td>106</td>
</tr>
<tr>
<td>&lt;/2</td>
<td>106</td>
</tr>
</tbody>
</table>

- **A**
  - abolish/1: 183
  - abort/0: 130
  - absolute_file_name/2: 116
  - absolute_file_name/7: 117
  - accepted_type/2: 627
  - action_widget/1: 522
  - action_widget/3: 522
  - active_agents/1: 405
  - add_after/4: 225, 663
  - add_before/4: 226, 664
  - add_edges/3: 674
  - add_environment_whitespace/3: 860
  - add_indentation/3: 861
  - add_prefix/3: 381
  - add_suffix/3: 381
  - add_vertices/3: 674
  - adjust_debugger_state/2: 56
  - aggregate_function/3: 621
  - aggregate_functor/2: 622
class$default_cons/1       514, 534, 549, 559, 561
class$destructor/3         514, 534, 550, 560, 562
class$implements/2         514, 534, 550, 560, 562
class$initial_state/3      513, 533, 549, 559, 561
class$super/2              513, 533, 549, 559, 561
class$virtual/6            513, 533, 549, 559, 561
close/2                    207
close/1                    113
close/2                    303
close_client/0             655
close_DEF/5                837
close_EXTENDEDPROTO/6      836
close_file/1               259
close_input/1              303
close_node/5               836
close_nodeGut/4            836
close_output/1             303
close_predicate/1          140
close_PROTO/6              836
close_script/5             837
code_class/2               125
color/2                    754
column_value/1             521
columnspan_value/1         521
combine_attributes/2       152
command_button/1           535
copy/3                     97
copy/benchmark2            878
comparison/2               880
copy_compile/1             41
compiler_and_opts/2        491
complete_dict/3            689
complete_vars_dict/3       690
compound/1                 205
concurrent/1               359
connect_to_socket/2        363
connect_to_socket_type/4   366
copy_constructor/0          451
consult/1                  41
contains_ro/2              226
contains1/2                227
continue/3                 869
convert_atoms_to_string/2  843
convert_permutations/4     378
coord/2                    563
coord/4                    551, 555
copy_args/3                335
copy_file/2                240
copy_files/2               378
copy_files_rec/3           382
copy_stdout/1              332
copy_term/2                92
core/1                     512
correct_commenting/4       857
cost/3                     880
create_dict/2              689
create_dictionaries/1      821
create_directed_field/5    857
create_environment/4       859
create_field/3             856
create_field/4             856
create_field/5             857
create_node/3              855
create_parse_structure/1   858
create_parse_structure/2   858
create_parse_structure/3   858
creation_bind/1            523
creation_menu_name/1       527
creation_options/1         523, 526
creation_options_entry/1   527
creation_position/1        523
creation_position_grid/1   523
cross_product/2            229
crtrl_clean/1              321
crtrlclean/0               321
current_atom/1             255
current_env/2              240
current_executable/1       242
current_fact/1             138, 600
current_fact/2             139
current_fact_nb/1          140
current_host/1             241
current_infixop/4           198
current_input/1            113
current_key/2              262
current_module/1           156
current_op/3               198
current_output/1           114
current_postfixop/3        198
current_predicate/1        184
current_predicate/2        184
current_prefixop/3         198
current_prolog_flag/2      132
current_stream/3           115
cyg2win/3                  250
D

data/1 ................................................. 185
datetime/1 ............................................. 238
datetime/9 ............................................. 238
datetime_atom/1 ..................................... 380
datetime_atom/2 ..................................... 380
datetime_string/1 .................................... 380
datetime_string/2 .................................... 380
davinci/0 ............................................. 497
davinci_get/1 ......................................... 497
davinci_get_all/1 .................................... 497
davinci_lgraph/1 ..................................... 498
davinci_put/1 ......................................... 498
davinci_quit/0 ........................................ 498
davinci_ugraph/1 .................................... 498
db_query/4 ........................................... 612
db_query_one_tuple/4 ................................. 613
dbassertz_fact/1 ..................................... 607
dbcall/2 ............................................... 608
dbcurrent_fact/1 ...................................... 607
dbfindall/4 ........................................... 608
dbId/2 ................................................ 631
dbretract_fact/1 ..................................... 607
dbretractall_fact/1 ................................ 607
dcg_translation/2 .................................... 215
debug/0 ............................................... 54
debug/1 ............................................... 148
debug_goal/1 ......................................... 76
debug_goal/2 ......................................... 314
debug_goal/3 ......................................... 315
debug_message/1 .................................... 314
debug_message/2 .................................... 314
debug_module/1 ...................................... 53
debug_module_source/1 ............................... 53
debugging/0 ......................................... 56
dec_indentation/2 .................................. 861
decompose_field/3 ................................... 837
define_flag/3 ....................................... 134, 188, 196, 251, 576
del_dir_if_empty/1 .................................. 377
del_endings_nofail/2 ................................ 379
del_env/1 ............................................. 240
del_file_nofail/1 .................................... 378
del_file_nofail/2 .................................... 379
del_global/1 ......................................... 403
del_vertices/3 ...................................... 674
delete/1 .............................................. 512
delete/2 .............................................. 874
delete/3 .............................................. 224, 664
delete_after/2 ...................................... 874
delete_directory/1 .................................. 250
delete_file/1 ........................................ 250
delete_files/1 ....................................... 378
delete_files_nofail/1 ......................... 378
delete_files/1 ....................................... 381
delete_non_ground/3 ................................ 224
delete_on_ctrlc/2 .................................. 321
delete_top/2 ........................................ 874
derived_from/2 ..................................... 467
describe/1 ........................................... 349
destroy/1 ............................................ 468
destructor/0 ......................................... 452, 530
det__try/3 ........................................... 411
detach_attribute/1 .................................. 152
digraph_to_ugraph/2 ................................. 669
dic_get/3 ............................................. 306
dic_lookup/3 ......................................... 305
dic_lookup/4 ......................................... 306
dic_node/2 ............................................ 305
dic_replace/4 ....................................... 306
dict2varnames1/2 ................................... 690
dictionary/5 ......................................... 305
dictionary/6 ......................................... 819
dictionary_insert/5 ................................ 822
dictionary_lookup/5 ................................ 822
difference/3 ......................................... 228
directory_files/2 ................................... 246
display/1 ............................................. 126
display/2 ............................................. 126
display_list/1 ....................................... 149
display_string/1 .................................... 149
display_term/1 ....................................... 149
displayq/1 ........................................... 127
displayq/2 ........................................... 127
div_times/2 .......................................... 880
digraph_to_lgraph/2 ................................. 670
dlist/3 ............................................... 226
do/2 .................................................. 380
do_interface/1 ...................................... 495
do_on_abolish/1 ..................................... 185
dyn_load_cfg_module_into_make/1 ............... 376
dynamic/1 ........................................... 185
dynamic_search_path/1 ............................... 43

e

edges/2 ............................................. 674
dges_to_lgraph/2 ................................... 670
dges_to_ugraph/2 ................................... 670
eemacs_edit/1 ....................................... 652
eemacs_edit_nowait/1 ................................ 652
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_byte/2</td>
<td>201</td>
</tr>
<tr>
<td>get_char/1</td>
<td>202</td>
</tr>
<tr>
<td>get_char/2</td>
<td>202</td>
</tr>
<tr>
<td>get_code/1</td>
<td>121</td>
</tr>
<tr>
<td>get_code/2</td>
<td>121</td>
</tr>
<tr>
<td>get_cookies/1</td>
<td>573</td>
</tr>
<tr>
<td>get_definition_dictionary/2</td>
<td>821</td>
</tr>
<tr>
<td>get_dictionaries/2</td>
<td>863</td>
</tr>
<tr>
<td>get_dirs/2</td>
<td>381</td>
</tr>
<tr>
<td>get_environment/2</td>
<td>863</td>
</tr>
<tr>
<td>get_environment_name/2</td>
<td>859</td>
</tr>
<tr>
<td>get_environment_type/2</td>
<td>860</td>
</tr>
<tr>
<td>get_first_parsed/3</td>
<td>866</td>
</tr>
<tr>
<td>get_form_input/1</td>
<td>572</td>
</tr>
<tr>
<td>get_form_value/3</td>
<td>572</td>
</tr>
<tr>
<td>get_general_options/1</td>
<td>883</td>
</tr>
<tr>
<td>get_global/2</td>
<td>403</td>
</tr>
<tr>
<td>get_indentation/2</td>
<td>861</td>
</tr>
<tr>
<td>get_line/1</td>
<td>307</td>
</tr>
<tr>
<td>get_line/2</td>
<td>307</td>
</tr>
<tr>
<td>get_os/1</td>
<td>155</td>
</tr>
<tr>
<td>get_parsed/2</td>
<td>862</td>
</tr>
<tr>
<td>get_pid/1</td>
<td>241</td>
</tr>
<tr>
<td>get_primes/2</td>
<td>665</td>
</tr>
<tr>
<td>get_prototype_definition/2</td>
<td>848</td>
</tr>
<tr>
<td>get_prototype_dictionary/2</td>
<td>822</td>
</tr>
<tr>
<td>get_prototype_interface/2</td>
<td>847</td>
</tr>
<tr>
<td>get_rec_dirs/2</td>
<td>381</td>
</tr>
<tr>
<td>get_rec_dirs/3</td>
<td>382</td>
</tr>
<tr>
<td>get_rec_dirs/4</td>
<td>382</td>
</tr>
<tr>
<td>get_row_number/2</td>
<td>860</td>
</tr>
<tr>
<td>get_stream/2</td>
<td>317</td>
</tr>
<tr>
<td>get_type/2</td>
<td>627</td>
</tr>
<tr>
<td>get1_code/1</td>
<td>122</td>
</tr>
<tr>
<td>get1_code/2</td>
<td>122</td>
</tr>
<tr>
<td>getcounter/2</td>
<td>661</td>
</tr>
<tr>
<td>getc/2</td>
<td>125</td>
</tr>
<tr>
<td>getc1/2</td>
<td>126</td>
</tr>
<tr>
<td>getenviron/2</td>
<td>239</td>
</tr>
<tr>
<td>glb/2</td>
<td>737</td>
</tr>
<tr>
<td>goal_id/1</td>
<td>358</td>
</tr>
<tr>
<td>graph_b1/13</td>
<td>777</td>
</tr>
<tr>
<td>graph_b1/9</td>
<td>776</td>
</tr>
<tr>
<td>graph_b2/13</td>
<td>786</td>
</tr>
<tr>
<td>graph_b2/9</td>
<td>786</td>
</tr>
<tr>
<td>graph_w1/13</td>
<td>778</td>
</tr>
<tr>
<td>graph_w1/9</td>
<td>777</td>
</tr>
<tr>
<td>graph_w2/13</td>
<td>788</td>
</tr>
<tr>
<td>graph_w2/9</td>
<td>787</td>
</tr>
</tbody>
</table>

**H**

- halt/0 ........................................... 130
- halt/1 ........................................... 130
- halt_server/0 .................................... 656
- handle_error/2 .................................... 323
- hash_term/2 ....................................... 392
- height/1 ......................................... 552, 556
- height_value/1 .................................... 531
- hide_/0 ........................................... 530
- hide_shape/1 ...................................... 530
- highlight_color/1 ................................ 517
- highlightbackground_color/1 ................. 517
- hostname_address/2 .............................. 364
- html_expansion/2 .................................. 576
- html_protect/1 .................................... 576
- html_report_error/1 ............................. 572
- html_template/3 ................................... 570
- html2terms/2 ...................................... 569
- http_lines/3 .................................... 576

**I**

- icon_address/2 ..................................... 576
- if/3 ................................................ 74
- imports_meta_pred/3 ............................ 351
- in/1 ............................................... 655
- in/2 ............................................... 655
- in/noblock/1 ..................................... 655
- in_stream/2 ........................................ 656
- inc_indentation/2 .................................. 861
- inccounter/2 ...................................... 661
- include/1 .......................................... 41
- indentation_list/2 ............................... 837
- indep/1 ........................................... 406
- indep/2 ........................................... 405
- inform_user/1 ..................................... 149
- inherited/1 ........................................ 451
- init_sql_persdb/0 ................................ 607
- initialize_db/0 ................................... 594
- insert/3 .......................................... 685, 874
- insert_after/3 .................................... 874
- insert_comments_in_beginning/3 ............ 859
- insert_last/3 ..................................... 226
- insert_parsed/3 ................................... 865
- insert_top/3 ...................................... 874
- inside_proto/1 .................................... 863
- instance_codes/2 .................................. 468
- instance_of/2 .................................... 466
- intercept/3 ....................................... 129
- interface/2 ....................................... 467
<table>
<thead>
<tr>
<th>Predicate/Method De®nition Index</th>
<th>929</th>
</tr>
</thead>
<tbody>
<tr>
<td>interp_file/2</td>
<td>750</td>
</tr>
<tr>
<td>interpret_compiled_goal/2</td>
<td>76</td>
</tr>
<tr>
<td>interpret_goal/2</td>
<td>76</td>
</tr>
<tr>
<td>intersection/3</td>
<td>228</td>
</tr>
<tr>
<td>intset_delete/3</td>
<td>227</td>
</tr>
<tr>
<td>intset_in/2</td>
<td>227</td>
</tr>
<tr>
<td>intset_insert/3</td>
<td>227</td>
</tr>
<tr>
<td>intset_sequence/3</td>
<td>227</td>
</tr>
<tr>
<td>is/2</td>
<td>105</td>
</tr>
<tr>
<td>is_array/1</td>
<td>659</td>
</tr>
<tr>
<td>is_connected_to_java/0</td>
<td>649</td>
</tr>
<tr>
<td>is_dictionaries/1</td>
<td>821</td>
</tr>
<tr>
<td>issue_debug_messages/1</td>
<td>315</td>
</tr>
<tr>
<td>J</td>
<td></td>
</tr>
<tr>
<td>java_add_listener/3</td>
<td>641</td>
</tr>
<tr>
<td>java_connect/2</td>
<td>638</td>
</tr>
<tr>
<td>java_create_object/2</td>
<td>639</td>
</tr>
<tr>
<td>java_debug/1</td>
<td>649</td>
</tr>
<tr>
<td>java_debug_redo/1</td>
<td>649</td>
</tr>
<tr>
<td>java_delete_object/1</td>
<td>640</td>
</tr>
<tr>
<td>java_disconnect/0</td>
<td>638</td>
</tr>
<tr>
<td>java_get_value/2</td>
<td>641</td>
</tr>
<tr>
<td>java_invoke_method/2</td>
<td>640</td>
</tr>
<tr>
<td>java_query/2</td>
<td>648</td>
</tr>
<tr>
<td>java_remove_listener/3</td>
<td>642</td>
</tr>
<tr>
<td>java_response/2</td>
<td>648</td>
</tr>
<tr>
<td>java_set_value/2</td>
<td>641</td>
</tr>
<tr>
<td>java_start/0</td>
<td>637</td>
</tr>
<tr>
<td>java_start/1</td>
<td>637</td>
</tr>
<tr>
<td>java_start/2</td>
<td>638</td>
</tr>
<tr>
<td>java_stop/0</td>
<td>638</td>
</tr>
<tr>
<td>java_use_module/1</td>
<td>639</td>
</tr>
<tr>
<td>join_socket_interface/0</td>
<td>648</td>
</tr>
<tr>
<td>justify_entry/1</td>
<td>542</td>
</tr>
<tr>
<td>justify_text/1</td>
<td>565</td>
</tr>
<tr>
<td>K</td>
<td></td>
</tr>
<tr>
<td>keysort/2</td>
<td>231</td>
</tr>
<tr>
<td>keyword/1</td>
<td>596, 602</td>
</tr>
<tr>
<td>L</td>
<td></td>
</tr>
<tr>
<td>label_value/1</td>
<td>526, 547</td>
</tr>
<tr>
<td>labeling/1</td>
<td>735</td>
</tr>
<tr>
<td>last/2</td>
<td>227</td>
</tr>
<tr>
<td>leash/1</td>
<td>56</td>
</tr>
<tr>
<td>length/2</td>
<td>224, 785</td>
</tr>
<tr>
<td>length_next/2</td>
<td>875</td>
</tr>
<tr>
<td>length_prev/2</td>
<td>875</td>
</tr>
<tr>
<td>letter_match/2</td>
<td>668</td>
</tr>
<tr>
<td>library_directory/1</td>
<td>62, 119</td>
</tr>
<tr>
<td>linda_client/1</td>
<td>655</td>
</tr>
<tr>
<td>linda_timeout/2</td>
<td>656</td>
</tr>
<tr>
<td>line_count/2</td>
<td>114</td>
</tr>
<tr>
<td>line_position/2</td>
<td>114</td>
</tr>
<tr>
<td>linker_and_opts/2</td>
<td>491</td>
</tr>
<tr>
<td>list_breakpt/0</td>
<td>56</td>
</tr>
<tr>
<td>list_concat/2</td>
<td>226</td>
</tr>
<tr>
<td>list_insert/2</td>
<td>227</td>
</tr>
<tr>
<td>list_lookup/3</td>
<td>227</td>
</tr>
<tr>
<td>list_lookup/4</td>
<td>227</td>
</tr>
<tr>
<td>list_to_list_of_lists/2</td>
<td>229</td>
</tr>
<tr>
<td>look_atom/1</td>
<td>358</td>
</tr>
<tr>
<td>look_field/3</td>
<td>333</td>
</tr>
<tr>
<td>lookahead/3</td>
<td>866</td>
</tr>
<tr>
<td>look_first_parsed/2</td>
<td>866</td>
</tr>
<tr>
<td>lookup_check_field/6</td>
<td>848</td>
</tr>
<tr>
<td>lookup_check_interface_fieldValue/8</td>
<td>848</td>
</tr>
<tr>
<td>lookup_check_node/4</td>
<td>848</td>
</tr>
<tr>
<td>lookup_field/4</td>
<td>849</td>
</tr>
<tr>
<td>lookup_field_access/4</td>
<td>850</td>
</tr>
<tr>
<td>lookup_fieldTypeId/1</td>
<td>849</td>
</tr>
<tr>
<td>lookup_get_fieldType/4</td>
<td>850</td>
</tr>
<tr>
<td>lookup_route/9</td>
<td>849</td>
</tr>
<tr>
<td>lookup_set_def/3</td>
<td>850</td>
</tr>
<tr>
<td>lookup_set_extern_prototype/4</td>
<td>851</td>
</tr>
<tr>
<td>lookup_set_interface_fieldValue/8</td>
<td>851</td>
</tr>
<tr>
<td>lookup_set_prototype/4</td>
<td>851</td>
</tr>
<tr>
<td>ls/2</td>
<td>379</td>
</tr>
<tr>
<td>ls/3</td>
<td>379</td>
</tr>
<tr>
<td>lss/3</td>
<td>382</td>
</tr>
<tr>
<td>lss/4</td>
<td>382</td>
</tr>
<tr>
<td>lss/6</td>
<td>382</td>
</tr>
<tr>
<td>lub/2</td>
<td>737</td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>make/1</td>
<td>375</td>
</tr>
<tr>
<td>make_actmod/2</td>
<td>42</td>
</tr>
<tr>
<td>make_directory/1</td>
<td>243</td>
</tr>
<tr>
<td>make_directory/2</td>
<td>242</td>
</tr>
<tr>
<td>make_dirpath/1</td>
<td>243</td>
</tr>
<tr>
<td>make_dirpath/2</td>
<td>243</td>
</tr>
<tr>
<td>make_exec/2</td>
<td>41</td>
</tr>
<tr>
<td>make_option/1</td>
<td>375</td>
</tr>
<tr>
<td>make_persistent/2</td>
<td>594</td>
</tr>
<tr>
<td>make_po/1</td>
<td>42, 233</td>
</tr>
<tr>
<td>make_sql_persistent/3</td>
<td>608</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>map/3</td>
<td>395</td>
</tr>
<tr>
<td>match_pattern/2</td>
<td>667</td>
</tr>
<tr>
<td>match_pattern/3</td>
<td>667</td>
</tr>
<tr>
<td>match_pattern_pred/2</td>
<td>668</td>
</tr>
<tr>
<td>maxdepth/1</td>
<td>56</td>
</tr>
<tr>
<td>member_0/2</td>
<td>663</td>
</tr>
<tr>
<td>memberchk/2</td>
<td>663</td>
</tr>
<tr>
<td>menu_data/1</td>
<td>525</td>
</tr>
<tr>
<td>menu_name/1</td>
<td>545, 548</td>
</tr>
<tr>
<td>merge/3</td>
<td>688</td>
</tr>
<tr>
<td>merge_tree/2</td>
<td>823</td>
</tr>
<tr>
<td>message/1</td>
<td>148</td>
</tr>
<tr>
<td>message/2</td>
<td>147</td>
</tr>
<tr>
<td>message_lns/4</td>
<td>147</td>
</tr>
<tr>
<td>mstringValue/5</td>
<td>829</td>
</tr>
<tr>
<td>mstringValue/7</td>
<td>831</td>
</tr>
<tr>
<td>minimum/3</td>
<td>395</td>
</tr>
<tr>
<td>mktemp/2</td>
<td>246</td>
</tr>
<tr>
<td>mode_of_module/2</td>
<td>234</td>
</tr>
<tr>
<td>modif_time/2</td>
<td>248</td>
</tr>
<tr>
<td>modif_time0/2</td>
<td>248</td>
</tr>
<tr>
<td>module_of/2</td>
<td>234</td>
</tr>
<tr>
<td>modules_tester/2</td>
<td>887</td>
</tr>
<tr>
<td>most_general_instance/3</td>
<td>337</td>
</tr>
<tr>
<td>most_specific_generalization/3</td>
<td>337</td>
</tr>
<tr>
<td>move_file/2</td>
<td>378</td>
</tr>
<tr>
<td>move_files/2</td>
<td>377</td>
</tr>
<tr>
<td>multibarchart/10</td>
<td>794</td>
</tr>
<tr>
<td>multibarchart/8</td>
<td>794</td>
</tr>
<tr>
<td>multifile/1</td>
<td>43</td>
</tr>
<tr>
<td>my_url/1</td>
<td>574</td>
</tr>
<tr>
<td>mysql_connect/5</td>
<td>623</td>
</tr>
<tr>
<td>mysql_disconnect/1</td>
<td>624</td>
</tr>
<tr>
<td>mysql_fetch/2</td>
<td>624</td>
</tr>
<tr>
<td>mysql_free_query_connection/1</td>
<td>624</td>
</tr>
<tr>
<td>mysql_get_tables/2</td>
<td>624</td>
</tr>
<tr>
<td>mysql_query/3</td>
<td>623</td>
</tr>
<tr>
<td>mysql_query_one_tuple/3</td>
<td>623</td>
</tr>
<tr>
<td>mysql_table_types/3</td>
<td>624</td>
</tr>
</tbody>
</table>

**N**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>name/2</td>
<td>99</td>
<td>930</td>
</tr>
<tr>
<td>name_menu/1</td>
<td>525</td>
<td>930</td>
</tr>
<tr>
<td>negated_comparison/2</td>
<td>621</td>
<td>930</td>
</tr>
<tr>
<td>neighbors/3</td>
<td>673</td>
<td>930</td>
</tr>
<tr>
<td>new/2</td>
<td>465</td>
<td>930</td>
</tr>
<tr>
<td>new_array/1</td>
<td>659</td>
<td>930</td>
</tr>
<tr>
<td>new_atom/1</td>
<td>255</td>
<td>930</td>
</tr>
<tr>
<td>new_interp/1</td>
<td>509, 749</td>
<td>930</td>
</tr>
</tbody>
</table>

**O**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>once/1</td>
<td>205</td>
<td>930</td>
</tr>
<tr>
<td>op/3</td>
<td>197</td>
<td>930</td>
</tr>
<tr>
<td>open/3</td>
<td>111, 329</td>
<td>930</td>
</tr>
<tr>
<td>open/4</td>
<td>112</td>
<td>930</td>
</tr>
<tr>
<td>open_client/2</td>
<td>656</td>
<td>930</td>
</tr>
<tr>
<td>open_DEF/5</td>
<td>836</td>
<td>930</td>
</tr>
<tr>
<td>open_EXTERNALPROTO/5</td>
<td>836</td>
<td>930</td>
</tr>
<tr>
<td>open_input/2</td>
<td>303</td>
<td>930</td>
</tr>
<tr>
<td>open_node/6</td>
<td>836</td>
<td>930</td>
</tr>
<tr>
<td>open_null_stream/1</td>
<td>303</td>
<td>930</td>
</tr>
<tr>
<td>open_output/2</td>
<td>303</td>
<td>930</td>
</tr>
<tr>
<td>open_predicate/1</td>
<td>140</td>
<td>930</td>
</tr>
<tr>
<td>open_PROTO/4</td>
<td>836</td>
<td>930</td>
</tr>
<tr>
<td>open_SCRIPT/5</td>
<td>837</td>
<td>930</td>
</tr>
<tr>
<td>optional_message/2</td>
<td>314</td>
<td>930</td>
</tr>
<tr>
<td>optional_message/3</td>
<td>314</td>
<td>930</td>
</tr>
</tbody>
</table>
ord_delete/3 .................................. 685
ord_disjoint/2 .................................. 688
ord_intersect/2 .................................. 686
ord_intersection/3 .................................. 686
ord_intersection_diff/4 .................................. 686
ord_member/2 .................................. 685
ord_subset/2 .................................. 687
ord_subset_diff/3 .................................. 687
ord_subtract/3 .................................. 686
ord_test_member/3 .................................. 686
ord_union/3 .................................. 687
ord_union_change/3 .................................. 688
ord_union_diff/4 .................................. 687
ord_union_symdiff/4 .................................. 688
otherwise/0 ........................................ 75
out/1 ........................................ 655, 843
out/3 ........................................ 843
out_stream/2 ........................................ 656
outline_color/1 ........................................ 553, 556
output_error/1 ........................................ 825
output_html/1 ........................................ 569

P
padx_value/1 ........................................ 519, 532
pady_value/1 ........................................ 520, 533
parse_term/3 ........................................ 319
parser/2 ........................................ 853
passerta_fact/1 ........................................ 592
passertz_fact/1 ........................................ 592
pattern/2 ........................................ 755
pause/1 ........................................ 237
peek_byte/1 ........................................ 201
peek_byte/2 ........................................ 201
peek_char/1 ........................................ 202
peek_char/2 ........................................ 203
peek_code/1 ........................................ 123
peek_code/2 ........................................ 122
percentbarchart1/7 ........................................ 759
percentbarchart2/7 ........................................ 765
percentbarchart3/7 ........................................ 768
percentbarchart4/7 ........................................ 772
performance/3 ........................................ 877
persistent_dir/2 ........................................ 595, 601
phrase/2 ........................................ 215
phrase/3 ........................................ 215
pit/2 ........................................ 735
pl2sqlInsert/2 ........................................ 633
pl2sqlstring/3 ........................................ 616
pl2sqlterm/3 ........................................ 618
point/2 ........................................ 563
point_to/3 ........................................ 675
pop_global/2 ........................................ 403
pop_prolog_flag/1 ........................................ 133
popen/3 ........................................ 245
portray/1 ........................................ 196
portray_attribute/2 ........................................ 196
portray_clause/1 ........................................ 194
portray_clause/2 ........................................ 194
postgres2sqltype/2 ........................................ 629
postgres2sqltypes_list/2 ........................................ 629
powerset/2 ........................................ 229
pred_test/2 ........................................ 888
predicate_property/2 ........................................ 255
pretract_fact/1 ........................................ 593
pretractall_fact/1 ........................................ 593
pretty_print/2 ........................................ 341
pretty_print/3 ........................................ 341
prettyvars/1 ........................................ 195
prev/2 ........................................ 873
print/1 ........................................ 194
print/2 ........................................ 194
printable_char/1 ........................................ 195
prolog_flag/3 ........................................ 132
prolog_query/2 ........................................ 648
prolog_response/2 ........................................ 649
prolog_server/0 ........................................ 643
prolog_server/1 ........................................ 644
prolog_server/2 ........................................ 644
prompt/2 ........................................ 133
prune_dict/3 ........................................ 690
push_dictionaries/3 ........................................ 862
push_global/2 ........................................ 403
push_prolog_flag/2 ........................................ 133
push_whitespace/3 ........................................ 862
put_byte/1 ........................................ 201
put_byte/2 ........................................ 202
put_code/1 ........................................ 203
put_char/1 ........................................ 203
put_char/2 ........................................ 203
put_code/1 ........................................ 124
put_code/2 ........................................ 123

Q
q_delete/3 ........................................ 681
q_empty/1 ........................................ 681
q_insert/3 ........................................ 681
q_member/2 ........................................ 681
query_generation/3 ........................................ 619
query_requests/2 ........................................ 645
<table>
<thead>
<tr>
<th>query_solutions/2</th>
<th>644</th>
</tr>
</thead>
<tbody>
<tr>
<td>random/1</td>
<td>683</td>
</tr>
<tr>
<td>random/3</td>
<td>683</td>
</tr>
<tr>
<td>random_color/1</td>
<td>755</td>
</tr>
<tr>
<td>random_darkcolor/1</td>
<td>756</td>
</tr>
<tr>
<td>random_lightcolor/1</td>
<td>755</td>
</tr>
<tr>
<td>random_pattern/1</td>
<td>756</td>
</tr>
<tr>
<td>rd/1</td>
<td>656</td>
</tr>
<tr>
<td>rd/2</td>
<td>656</td>
</tr>
<tr>
<td>rd_findall/3</td>
<td>656</td>
</tr>
<tr>
<td>rd_noblock/1</td>
<td>656</td>
</tr>
<tr>
<td>reachability/4</td>
<td>703</td>
</tr>
<tr>
<td>read/1</td>
<td>187</td>
</tr>
<tr>
<td>read/2</td>
<td>187</td>
</tr>
<tr>
<td>read_page/2</td>
<td>814</td>
</tr>
<tr>
<td>read_term/2</td>
<td>188</td>
</tr>
<tr>
<td>read_term/3</td>
<td>188</td>
</tr>
<tr>
<td>read_terms_file/2</td>
<td>844</td>
</tr>
<tr>
<td>read_top_level/3</td>
<td>188</td>
</tr>
<tr>
<td>read_vrm_file/2</td>
<td>844</td>
</tr>
<tr>
<td>readf/2</td>
<td>380</td>
</tr>
<tr>
<td>reading/4</td>
<td>835</td>
</tr>
<tr>
<td>reading/5</td>
<td>836</td>
</tr>
<tr>
<td>reading/6</td>
<td>836</td>
</tr>
<tr>
<td>rebuild_foreign_interface/1</td>
<td>493</td>
</tr>
<tr>
<td>rebuild_foreign_interface_explicit_decls/2</td>
<td>494</td>
</tr>
<tr>
<td>receive_confirm/2</td>
<td>512</td>
</tr>
<tr>
<td>receive_event/2</td>
<td>511</td>
</tr>
<tr>
<td>receive_list/2</td>
<td>511</td>
</tr>
<tr>
<td>receive_result/2</td>
<td>510</td>
</tr>
<tr>
<td>recorda/3</td>
<td>261</td>
</tr>
<tr>
<td>recorded/3</td>
<td>262</td>
</tr>
<tr>
<td>recorded/3</td>
<td>261</td>
</tr>
<tr>
<td>reduce_indentation/3</td>
<td>862</td>
</tr>
<tr>
<td>relief_type/1</td>
<td>518</td>
</tr>
<tr>
<td>remove_code/3</td>
<td>866</td>
</tr>
<tr>
<td>remove_comments/4</td>
<td>837</td>
</tr>
<tr>
<td>rename/2</td>
<td>691</td>
</tr>
<tr>
<td>rename_file/2</td>
<td>250</td>
</tr>
<tr>
<td>repeat/0</td>
<td>75</td>
</tr>
<tr>
<td>replace_strings_in_file/3</td>
<td>380</td>
</tr>
<tr>
<td>reserved_words/1</td>
<td>818</td>
</tr>
<tr>
<td>retract/1</td>
<td>183</td>
</tr>
<tr>
<td>retract_fact/1</td>
<td>139, 593, 600</td>
</tr>
<tr>
<td>retract_fact_nb/1</td>
<td>140</td>
</tr>
<tr>
<td>retractall/1</td>
<td>183</td>
</tr>
<tr>
<td>retractall_fact/1</td>
<td>139, 594</td>
</tr>
<tr>
<td>retrieve_list_of_values/2</td>
<td>738</td>
</tr>
<tr>
<td>retrieve_range/2</td>
<td>736</td>
</tr>
<tr>
<td>retrieve_store/2</td>
<td>736</td>
</tr>
<tr>
<td>reverse/2</td>
<td>223</td>
</tr>
<tr>
<td>reverse/3</td>
<td>224</td>
</tr>
<tr>
<td>reverse_parsed/2</td>
<td>865</td>
</tr>
<tr>
<td>rewind/2</td>
<td>874</td>
</tr>
<tr>
<td>rooted_subgraph/3</td>
<td>675</td>
</tr>
<tr>
<td>row_value/1</td>
<td>520</td>
</tr>
<tr>
<td>rowspan_value/1</td>
<td>520</td>
</tr>
<tr>
<td>run_tester/10</td>
<td>889</td>
</tr>
<tr>
<td>running_queries/2</td>
<td>645</td>
</tr>
</tbody>
</table>

| safe_write/2       | 367 |
| scattergraph_b1/12 | 780 |
| scattergraph_b1/8  | 779 |
| scattergraph_b2/12 | 789 |
| scattergraph_b2/8  | 789 |
| scattergraph_w1/12 | 781 |
| scattergraph_w1/8  | 780 |
| scattergraph_w2/12 | 791 |
| scattergraph_w2/8  | 790 |
| second_prompt/2    | 188 |
| see/1              | 259 |
| seeing/1           | 259 |
| seen/0             | 259 |
| select/3           | 224 |
| select_socket/5    | 365 |
| self/1             | 451 |
| send_term/2        | 511 |
| serve_socket/3     | 367 |
| set_action/1       | 547 |
| set_cookie/2       | 573 |
| set_debug_mode/1   | 42, 234 |
| set_debug_module/1 | 234 |
| set_debug_module_source/1 | 234 |
| set_env/2          | 240 |
| set_environment/3  | 865 |
| set_fact/1         | 140 |
| set_general_options/1 | 883 |
| set_global/2       | 403 |
| set_input/1        | 113 |
| set_name/1         | 547 |
| set_nodebug_module/1 | 42, 234 |
| set_nodebug_module/1 | 234 |
| set_output/1       | 113 |
set_parsed/3 ........................................ 865
set_perms/2 ........................................ 380
set_prolog_flag/2 .................................. 132
set_stream/3 ........................................ 317
setarg/3 ............................................ 413
setcounter/2 ....................................... 661
setof/3 .............................................. 240
shell/1 .............................................. 344
shell/2 .............................................. 365
show/0 ............................................... 530
show_shape/1 ....................................... 530
side_type/1 ......................................... 518, 531
simple_message/2 ................................ 313
skip_code/1 .......................................... 123
skip_code/2 .......................................... 123
skip_line/0 .......................................... 123
skip_line/1 .......................................... 123
socket_accept/2 .................................... 366
socket_recv_code/3 ................................ 365
socket_send/2 ....................................... 364
sort/2 ............................................... 231
sort_dict/2 ......................................... 690
spy/1 .................................................. 54
sql_attribute/4 ..................................... 619, 633
sql_relation/3 ....................................... 618, 633
sql_get_tables/2 .................................... 609
sql_goal_tr/2 ........................................ 631
sql_persistent_location/2 ....................... 611
sql_persistent_tr/2 ................................ 631
sql_query/3 .......................................... 609
sql_query_one_tuple/3 .............................. 613
sql_table_types/3 .................................. 610
sqlterm2string/2 ................................... 618
srandom/1 .......................................... 684
start_socket_interface/2 ......................... 647
start_threads/0 ..................................... 649
start_vrmlScene/4 .................................. 837
statistics/0 ......................................... 253
statistics/2 ......................................... 253
stop_parse/2 ........................................ 865
stop_socket_interface/0 ............................ 648
stream_code/2 ...................................... 116
stream_property/2 .................................. 207
stream_to_string/2 ................................ 332
string/3 ............................................. 309
string2term/2 ....................................... 319
strip_clean/2 ....................................... 864
strip_exposed/2 .................................... 864
strip_from_list/2 ................................... 863
strip_from_term/2 ................................... 864
strip_interface/2 ................................... 864
strip_restricted/2 .................................. 864
sum_list/2 .......................................... 665
sum_list/3 .......................................... 666
sum_list_of_lists/2 ................................ 666
sum_list_of_lists/3 ................................ 666
sybase2sqltype/2 ................................... 629
sybase2sqltypes_list/2 ............................ 629
symbolic_link/2 ............................ 378
symbolic_link/3 ..................................... 378
SYSCALL/1 .......................................... 394
system/1 ............................................ 245
system/2 ............................................ 245
system_lib/1 ........................................ 349
T

tab/1 .................................................. 125
tab/2 .................................................. 124
tablewidget1/4 ...................................... 797
tablewidget1/5 ...................................... 797
tablewidget2/4 ...................................... 801
tablewidget2/5 ...................................... 802
tablewidget3/4 ...................................... 803
tablewidget3/5 ...................................... 803
tablewidget4/4 ...................................... 805
tablewidget4/5 ...................................... 806
tcl_delete/1 ........................................ 505
tcl_eval/3 ........................................... 504
tcl_event/3 ......................................... 505
tcl_name/1 ........................................... 526
tcl_new/1 ........................................... 504
tcltk/2 .............................................. 510
tcltk_raw_code/2 ................................... 510, 749
tearoff_value/1 ..................................... 526
tell/1 ................................................. 259
telling/1 ............................................ 259
terms_file_to_vrml/2 ............................... 812
# Property Definition Index

<table>
<thead>
<tr>
<th>Character</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>&lt;=/2</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>atom/1</td>
</tr>
<tr>
<td></td>
<td>atomic/1</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>class_name/1</td>
</tr>
<tr>
<td></td>
<td>class_source/1</td>
</tr>
<tr>
<td></td>
<td>compat/2</td>
</tr>
<tr>
<td></td>
<td>constructor/1</td>
</tr>
<tr>
<td></td>
<td>covered/1</td>
</tr>
<tr>
<td></td>
<td>covered/2</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>davinci_command/1</td>
</tr>
<tr>
<td></td>
<td>dictionary/1</td>
</tr>
<tr>
<td></td>
<td>do_not_free/2</td>
</tr>
<tr>
<td></td>
<td>docstring/1</td>
</tr>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>eval/1</td>
</tr>
<tr>
<td></td>
<td>expander_pred/1</td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>fails/1</td>
</tr>
<tr>
<td></td>
<td>field_Id/1</td>
</tr>
<tr>
<td></td>
<td>finite_solutions/1</td>
</tr>
<tr>
<td></td>
<td>float/1</td>
</tr>
<tr>
<td></td>
<td>foreign/1</td>
</tr>
<tr>
<td></td>
<td>foreign/2</td>
</tr>
<tr>
<td></td>
<td>fuzzybody/1</td>
</tr>
<tr>
<td></td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>ground/1</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>head_pattern/1</td>
</tr>
<tr>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>indep/1</td>
</tr>
<tr>
<td></td>
<td>indep/2</td>
</tr>
<tr>
<td></td>
<td>instance/2</td>
</tr>
<tr>
<td></td>
<td>instance_id/1</td>
</tr>
<tr>
<td></td>
<td>integer/1</td>
</tr>
<tr>
<td></td>
<td>interface_name/1</td>
</tr>
<tr>
<td></td>
<td>interface_source/1</td>
</tr>
<tr>
<td></td>
<td>internal_module_id/1</td>
</tr>
<tr>
<td></td>
<td>is_det/1</td>
</tr>
<tr>
<td></td>
<td>iso/1</td>
</tr>
<tr>
<td></td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>lgraph/1</td>
</tr>
<tr>
<td></td>
<td>line/1</td>
</tr>
<tr>
<td></td>
<td>linear/1</td>
</tr>
<tr>
<td></td>
<td>list1/2</td>
</tr>
<tr>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>member/2</td>
</tr>
<tr>
<td></td>
<td>method_spec/1</td>
</tr>
<tr>
<td></td>
<td>mshare/1</td>
</tr>
<tr>
<td></td>
<td>multpredspec/1</td>
</tr>
<tr>
<td></td>
<td>mut_exclusive/1</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>nabody/1</td>
</tr>
<tr>
<td></td>
<td>native/1</td>
</tr>
<tr>
<td></td>
<td>native/2</td>
</tr>
<tr>
<td></td>
<td>non_det/1</td>
</tr>
<tr>
<td></td>
<td>nonground/1</td>
</tr>
<tr>
<td></td>
<td>nonvar/1</td>
</tr>
<tr>
<td></td>
<td>not_covered/1</td>
</tr>
<tr>
<td></td>
<td>not_fails/1</td>
</tr>
<tr>
<td></td>
<td>not_further_inst/2</td>
</tr>
<tr>
<td></td>
<td>not_mut_exclusive/1</td>
</tr>
<tr>
<td></td>
<td>number/1</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>parse/1</td>
</tr>
<tr>
<td></td>
<td>possibly_fails/1</td>
</tr>
<tr>
<td></td>
<td>possibly_nondet/1</td>
</tr>
</tbody>
</table>
R
regtype/1......................... 85
returns/2......................... 486

S
sideff/2......................... 85
size_lb/2......................... 292
size_of/3......................... 486
size_ub/2......................... 292
sourcenames/1.................... 43
steps/2........................... 292
steps_lb/2....................... 292
steps_ub/2....................... 292
sublist/2......................... 228
subordlist/2..................... 228

T
terminates/1..................... 293
type/2............................ 89

U
ugraph/1.......................... 499

V
var/1.............................. 87
virtual_method_spec/1........... 469

W
write_option/1.................... 192
## Regular Type Definition Index

### A
- address/1 ............................................. 485
- answertableterm/1 .............................. 626
- answertupleterm/1 .............................. 626
- any_term/1 ........................................... 486
- apropos_spec/1 .................................... 350
- argspec/1 ........................................... 392
- arithexpression/1 ............................... 108
- assert_body/1 ....................................... 277
- assert_status/1 .................................... 282
- assert_type/1 ........................................ 282
- atm/1 ................................................. 80
- atm_or_atm_list/1 ............................... 84
- attributes/1 ......................................... 782
- axis_limit/1 ......................................... 761

### B
- bltwish_interp/1 ................................. 750
- body/1 ............................................... 342
- bound/1 .............................................. 839
- bound_double/1 .................................... 839
- byte/1 ............................................... 485
- byte_list/1 .......................................... 485

### C
- c_assert_body/1 ..................................... 280
- callable/1 .......................................... 81
- canonic_html_term/1 ............................ 581
- canonic_xml_item/1 ............................... 747
- canonic_xml_query/1 ............................ 747
- canonic_xml_subquery/1 ........................ 747
- canonic_xml_term/1 ............................... 582, 747
- cell_value/1 ......................................... 799
- character_code/1 .................................. 83
- clause/1 .............................................. 342
- clauses/1 ........................................... 342
- click_result/1 ...................................... 256
- clockfreq_result/1 ............................... 257
- cterm/1 .............................................. 342
- color/1 .............................................. 753
- complex_arg_property/1 ....................... 278
- complex_goal_property/1 ...................... 279
- constant/1 .......................................... 81

### D
- datetime_struct/1 ............................... 239
- dbconnection/1 ..................................... 612, 623
- dbname/1 ............................................. 610, 625
- dbqueryconnection/1 ......................... 624
- dlist/1 .............................................. 875
- detcond/1 .......................................... 408
- dgraph/1 ............................................ 669
- dictionary/1 ....................................... 280, 839
- directoryname/1 .................................. 596
- digraph/1 ........................................... 669

### E
- elisp_string/1 ....................................... 653
- environment/1 ...................................... 840

### F
- fagggregator/1 ..................................... 438
- fd_item/1 ........................................... 734
- fd_range/1 .......................................... 734
- fd_store/1 .......................................... 735
- fd_store_entity/1 ............................... 735
- fd_subrange/1 ..................................... 734
- flag/1 ............................................... 342
- filt/1 ................................................. 80
- footer/1 ............................................. 761
- form_assignment/1 .............................. 585
- form_dict/1 ......................................... 585
- form_value/1 ....................................... 586
- format_control/1 ................................. 218

### G
- g_assert_body/1 ..................................... 281
- garbage_collection_option/1 ............. 256
- gc_result/1 ......................................... 257
- gnd/1 ............................................... 81

### H
- handler_type/1 ....................................... 752
- header/1 ............................................. 761
- hms_time/1 .......................................... 587
- html_term/1 ......................................... 583
- http_date/1 ......................................... 587
- http_request_param/1 ...................... 586
- http_response_param/1 .................... 586
The Ciao Prolog System

I
image/1 ........................................... 798
indexspecs/1 .................................... 392
int/1 .............................................. 79
int_list/1 ........................................ 485
intlist/1 .......................................... 665
io_mode/1 ......................................... 118

J
java_constructor/1 ............................. 639
java_event/1 .................................... 639
java_field/1 .................................... 639
java_method/1 .................................. 640
java_object/1 ................................... 639

K
dklist/1 ............................................ 232
dkeypair/1 ......................................... 232

L
lgraph/2 ........................................... 679
list/1 ............................................. 82
list/2 ............................................. 82
location/1 ....................................... 315

M
machine_name/1 .................................. 638
memory_option/1 .................................. 256
memory_result/1 .................................. 257
metaspec/1 ....................................... 70
modulename/1 ..................................... 70
month/1 .......................................... 587
multibar_attribute/1 ............................. 795

N
nneutralint/1 ..................................... 80
null/1 ............................................. 485
null_dict/1 ....................................... 689
num/1 ............................................ 80
numlist/1 ......................................... 665

O
open_option_list/1 .............................. 112
operator_specifier/1 ............................ 82
option/1 ......................................... 529

P
pair/1 ............................................. 671
parse/1 .......................................... 840
passwd/1 ......................................... 611, 625
path/1 ............................................ 409
pattern/1 ........................................ 668, 755
popen_mode/1 .................................... 245
postgresqltype/1 ................................ 629
predfunctor/1 .................................... 282
 predname/1 ....................................... 84
projterm/1 ....................................... 611, 617
prolog_goal/1 .................................... 639
property_conjunction/1 ........................ 279
property_startterm/1 ............................ 279
propfunctor/1 ..................................... 282

Q
querybody/1 ....................................... 611, 616

R
read_option/1 ..................................... 189
reference/1 ....................................... 141
row/1 .............................................. 798, 799

S
s ASSERT_body/1 .................................. 280
sequence/2 ....................................... 83
sequence_or_list/2 ............................. 83
shape/1 .......................................... 529
shutdown_type/1 ................................ 364
size/1 ............................................ 783
smooth/1 ......................................... 782
socket_type/1 ..................................... 364
socketname/1 ..................................... 610, 625
 sourcename/1 .................................... 117
sqlstring/1 ....................................... 617, 626
sqlite/1 .......................................... 627
stream/1 ......................................... 118
stream_alias/1 .................................... 118
string/1 .......................................... 83
struct/1 .......................................... 81
sybasetype/1....................... 628
symbol/1.......................... 783
symbol_option/1.................... 256
symbol_result/1.................... 257

table/1............................. 798
tag_attrib/1........................ 747
target/1............................. 375
tclCommand/1........................ 505
tclInterpreter/1..................... 505
term/1................................. 79
time_option/1....................... 256
time_result/1....................... 256
title/1................................. 761
translation_predname/1............. 145
tree/1................................ 840
triple/1.............................. 671
tuple/1................................ 612, 626

U
ughraph/1............................ 675
url_term/1............................ 586
user/1................................. 610, 625

V
value_dict/1.......................... 586
varname/1.............................. 691
varnamedict/1........................ 691
varnamesl/1............................ 691
vector/1................................ 782

W
wakeup_exp/1.......................... 418
weekday/1.............................. 587
whitespace/1........................... 840

X
xbarelement1/1....................... 761
xbarelement2/1....................... 765
xbarelement3/1 ....................... 769
xbarelement4/1 ....................... 773
xdr_node/1............................... 740
xelement/1.............................. 796

Y
yelement/1.............................. 760
Declaration Definition Index

A
add_clause_trans/1 ........................................ 145
add_goal_trans/1 ........................................ 145
add_sentence_trans/1 .................................... 144
add_term_trans/1 ......................................... 144
aggr/1 ...................................................... 436
argnames/1 .................................................. 397

C
calls/1 ...................................................... 271
calls/2 ...................................................... 271
comment/2 .................................................. 282
comp/1 ....................................................... 272
comp/2 ....................................................... 272
concurrent/1 ............................................... 141, 449

D
data/1 ......................................................... 141, 448
decl/1 ........................................................ 274
decl/2 ........................................................ 274
determinate/2 ................................................ 407
discontiguous/1 ............................................. 77
dynamic/1 .................................................... 449

E
ensure_loaded/1 ............................................. 71
entry/1 ........................................................ 273
export/1 ...................................................... 68, 448, 721
extra_compiler_opts/1 ....................................... 487
extra_compiler_opts/2 ....................................... 487
extra_linker_opts/1 .......................................... 488
extra_linker_opts/2 .......................................... 488

F
facts/2 ........................................................ 601
foreign_inline/2 ........................................... 489

I
impl_defined/1 .............................................. 77
implements/1 .............................................. 450
import/2 ..................................................... 69
include/1 ................................................... 71
index/1 ....................................................... 391
inherit_class/1 ............................................. 449
inheritable/1 ............................................... 448

initialization/1 ........................................... 78
instance_of/2 ............................................... 459

L
load_compilation_module/1 .............................. 144

M
meta_predicate/1 ........................................... 69
modedef/1 .................................................. 273
module/2 ................................................... 68
module/3 ................................................... 67
multifile/1 .................................................. 77

N
new/2 .......................................................... 460
new_declaration/1 ........................................ 143
new_declaration/2 .......................................... 143

O
on_abort/1 ................................................... 78
op/3 .......................................................... 143

P
persistent/2 ............................................... 595
pred/1 ....................................................... 270
pred/2 ....................................................... 271
prop/1 ........................................................ 272
prop/2 ........................................................ 273
public/1 .................................................... 448

R
redefining/1 ............................................... 78
reexport/1 .................................................. 69
reexport/2 .................................................. 69
regtype/1 ................................................... 286
regtype/2 ................................................... 287

S
sql_persistent/3 ........................................... 612
success/1 ................................................... 271
success/2 ................................................... 271
### U

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_active_module/2</td>
<td>424</td>
</tr>
<tr>
<td>use_class/1</td>
<td>459</td>
</tr>
<tr>
<td>use_compiler/1</td>
<td>488</td>
</tr>
<tr>
<td>use_compiler/2</td>
<td>488</td>
</tr>
<tr>
<td>use_foreign_library/1</td>
<td>487</td>
</tr>
<tr>
<td>use_foreign_library/2</td>
<td>487</td>
</tr>
<tr>
<td>use_foreign_source/1</td>
<td>487</td>
</tr>
<tr>
<td>use_foreign_source/2</td>
<td>487</td>
</tr>
<tr>
<td>use_linker/1</td>
<td>488</td>
</tr>
<tr>
<td>use_linker/2</td>
<td>489</td>
</tr>
<tr>
<td>use_module/1</td>
<td>69</td>
</tr>
<tr>
<td>use_module/2</td>
<td>68</td>
</tr>
<tr>
<td>use_package/1</td>
<td>71</td>
</tr>
</tbody>
</table>

### V

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual/1</td>
<td>450</td>
</tr>
</tbody>
</table>
Concept Definition Index

&
&-Prolog ........................................... 10
.
.ciaorc ........................................... 20, 25

A
abort ................................................. 51
abstract methods .................................. 450
acceptable modes .................................. 278
acknowledgments .................................. 9
active module .................................... 33, 421
active object .................................... 33, 421
addmodule and pred(N) meta-arguments .... 447
ancestors .......................................... 51
active object .................................... 51
answer variable .................................. 38
assertion body syntax ........................... 277, 280, 281
attribute .......................................... 448
attributed variables ................................ 151
Austrian Research Institute for AI ............ 10

B
binary directory .................................. 898
box-type debugger ................................ 45
breakpoint ....................................... 55
Bristol University ................................ 10
bugs, reporting .................................... 912

C
calls assertion .................................... 271
certainty factor .................................. 721
CGI ................................................. 567
CGI executables .................................. 59
check assertion .................................. 274
Christian Holzbauer .............................. 10
Ciao basic builtins ................................. 8, 65
Ciao engine ....................................... 10
Ciao preprocessor ................................ 10
ciao, global description ......................... 3
Ciao, why this name .............................. 4
ciao-users ......................................... 911
cltool installation ................................ 909
CLIP group ....................................... 9
closed ............................................. 140
command .......................................... 52

D
D.H.D. Warren .................................... 10
D.L. Bowen ....................................... 10
Daniel Cabeza .................................. 9
data declaration .................................. 137
data predicate .................................... 137
database initialization ........................... 594
debug options .................................... 50
debugger .......................................... 45
debugging ......................................... 51
dcl assertion ...................................... 274
declarations, user defined ......................... 77
DECsystem-10 Prolog User's Manual .......... 10
depth first iterative deepening ................ 427
depth limit ....................................... 427
destructor ......................................... 452
determinate goal .................................. 407
development environment ....................... 20, 25, 898, 900, 907
display ............................................ 51
downloading emacs ............................... 900, 908
downloading, latest versions ................... 911

comment assertion ................................ 274
comments, machine readable .................. 269
comp assertion .................................... 272
compatibility properties ......................... 283
compiler, standalone ............................. 29
compiling programs .............................. 18, 19, 24, 25
compiling, from command line ................ 29
compiling, Win32 ................................ 903
concurrency ...................................... 355
concurrent attribute .............................. 449
concurrent predicate .............................. 137
concurrent predicates ............................. 137
configuration file ................................ 709
constructor ....................................... 451
contributed libraries ............................. 9, 719
creep .............................................. 50
csh-compatible shell .............................. 18, 897, 900
current input stream ............................. 113
current output stream ............................ 113
Cygnus Win32 .................................... 903
L
L. Byrd ........................................ 10
L.M. Pereira ................................. 10
Leap .......................................... 50
Lib library .................................... 8, 65
Library directory ......................... 898
Limitations, architecture-specific .... 911
Linkoping U. ................................ 10
Loading mode ................................ 46
Loading programs ......................... 18, 24
LogIn ......................................... 38
LPdoc ......................................... 3
Lpmake ....................................... 709
Lpmake autodocumentation ............. 709

M
mailing list ................................... 911
Make .......................................... 709
MANPATH ......................................
Manual, printing ............................ 20, 23, 25, 908
Manual, tour .................................. 7
Manuals, printing ......................... 20, 25
Manuel Carro ................................ 9
Manuel Hermenegildo ...................... 9, 10
Maria Jose Garcia de la Banda ........ 9
Mats Carlsson ................................ 10
Maurice Bruynooghe ...................... 10
MCC ............................................ 10
Melbourne U. ................................. 10
Modular interface ......................... 33
Module qualification ...................... 67
Modules, active ............................. 33
Monash U. .................................... 10
Multi-evaluated .............................. 448
Multiarchitecture support .............. 902

N
Naming term arguments .................. 397
New Mexico State University .......... 10
Nodebug ....................................... 51
Nospy ......................................... 51
Notation ...................................... 5

O
Overriden ..................................... 449

P
P. Lincoln .................................... 38
Parallel Prolog .............................. 10
Parallelizing compiler ................... 10
Parametric type functor ................. 286
PATH .......................................... 899
Path alias .................................... 117
Patterns ...................................... 667
Paulo Moura ................................ 10
Pawel Pietrzak ............................... 10
Pedro Lopez .................................. 9
Peter Stuckey ................................ 10
Pierre Deransart ............................. 10
PiLoW on-line tutorial ..................... 567
Polymorphism ................................ 441
Pred assertion ............................... 270, 271
Print .......................................... 51
Printdepth ..................................... 52
Printing, manual ........................... 20, 23, 25, 908
Programming environment .............. 7, 27
Prolog flag ................................... 131, 147
Prolog server ............................... 644
Prolog shell scripts ....................... 59
Prolog to Java Interface Structure ...... 635
Prolog to Java Interface Structure, Java side ... 635
Prolog to Java Interface Structure, Prolog side ... 635
Prolog-emacs interface ................... 651
Prop assertion ............................... 272, 273
Properties of computations ............. 283
Properties of execution states ......... 283
Properties, basic ........................... 79
Properties, native ......................... 289
Protected .................................... 448
Public domain .............................. 1
Public interface ............................ 448
Pure Prolog .................................. 389

Q
Query ......................................... 37
| R | records | 8, 387, 397 |
|   | recursive level | 38 |
|   | references, to Ciao | 5 |
|   | referring to Ciao | 5 |
|   | regtype assertion | 286, 287 |
|   | regular expressions | 667 |
|   | regular type expression | 286 |
|   | reporting bugs | 912 |
|   | retry | 51 |
|   | Roger Nasr | 10, 38 |
|   | running programs | 18, 19, 24, 25 |

| S | Saumya Debray | 10 |
|   | scripts | 19, 25, 899 |
|   | Seif Haridi | 10 |
|   | sh-compatible shell | 18, 898, 900 |
|   | sharing sets | 290 |
|   | shortcut, windows | 907 |
|   | SICS | 10 |
|   | SICStus Prolog | 10 |
|   | skip | 50 |
|   | Socket implementation | 647 |
|   | source directory | 898 |
|   | spy | 51 |
|   | standard total ordering | 95 |
|   | status, this manual | 3 |
|   | style sheets | 17, 23, 901, 908 |
|   | subterm | 52 |
|   | success assertion | 271, 272 |
|   | super class | 449 |
|   | Swedish Institute of Computer Science | 10 |

| T | tar | 898 |
|   | Technical University of Madrid | 9 |
|   | top-level shell, starting, unix | 18 |
|   | top-level shell, starting, windows | 23 |
|   | tour, of the manual | 7 |
|   | troubleshooting | 897, 904, 907 |
|   | true assertion | 275 |
|   | trust assertion | 275 |

| U | U. of Arizona | 10 |
|   | uninstalling | 898, 901 |
|   | UPM | 10 |
|   | user module | 67 |
|   | user modules, debugging | 45 |
|   | user setup | 17 |
|   | users mailing list | 911 |

| V | Veroniek Dumortier | 10 |
|   | virtual | 450 |

| W | WAM | 10 |
|   | why the name Ciao | 4 |
|   | windows shortcut | 907 |
|   | Wlodek Drabent | 10 |
|   | write | 51 |
|   | WWW, interfacing with | 567 |

| X | XML | 567 |
# Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, etc., are referred to in the text of the document. Note that due to limitations of the info format unfortunately only the first reference will appear in online versions of the document.

<table>
<thead>
<tr>
<th>Concept</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>411</td>
<td>73</td>
<td>74</td>
<td>73</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>108</td>
<td>108</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>567</td>
<td>397</td>
<td>567</td>
<td>415</td>
<td>416</td>
<td>418</td>
<td>419</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>10</td>
<td>405</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;.Prolog</td>
<td>734</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`</td>
<td>745</td>
<td>83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`&lt;'/1</td>
<td>425</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`&lt;'/2</td>
<td>425</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`persdb/11'</td>
<td>592, 593</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>108</td>
<td>619</td>
<td>108</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*/1</td>
<td>295</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*/2</td>
<td>279, 295, 296</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, etc., are referred to in the text of the document. Note that due to limitations of the info format unfortunately only the first reference will appear in online versions of the document.
c:/.emacs

bytecode object files

c_itf

.call/2

.call/N

.call_in_module/2

.call_unknown/1

Calling emacs

callable/1

calls/1

calls/2

.cache

canonic_xml_subquery/1

canonic_xml_term/1

.canvas_class/1

.chartlib_text_error_protect/1

.chartlib_visual_error_protect/1

.chartlib/table_widget4

canonic_html_term/1

.character_count/2

.chartlib/table_widget3

canonic_xml_item/1

.chartlib/table_widget2

canonic_xml接纳_object/1

.chartlib/chartlib_errhandle

canvas_class/0

canvas_class/1

.chartlib/chartlib_genbar1

.chartlib/chartlib_genbar2

.chartlib/chartlib_genbar3

.chartlib/chartlib_genbar4

.chartlib/chartlib_gengraph1

.chartlib/chartlib_gengraph2

.chartlib/install_util

.chartlib/chartlib_genmultibar

.chartlib/chartlib_test_format

.chartlib/chartlib_errhandle

.chartlib/chartlib_text_error_protect/1

.chartlib/chartlib_visual_error_protect/1

.check assertion

.check/1

.check_sublist/4

.children_nodes/1

.chmod/2

.chmod/3

.choose_free_var/2

.choose_value/2

.choose_var/3

.choose_var/nd/2

.christian holzbauer

ciao

ciao basic builtins

.ciao engine

.ciao engine builtins

.ciao preprocessor

ciao, global description

.ciao, why this name

ciao-shell

ciao-users

ciao.el
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>del_endings_nofail/2</td>
<td>377, 379</td>
</tr>
<tr>
<td>del_env/1</td>
<td>169, 237, 240, 385</td>
</tr>
<tr>
<td>del_file_nofail/1</td>
<td>377, 378</td>
</tr>
<tr>
<td>del_file_nofail/2</td>
<td>377, 379</td>
</tr>
<tr>
<td>del_global/1</td>
<td>403</td>
</tr>
<tr>
<td>del_vertices/3</td>
<td>673, 674</td>
</tr>
<tr>
<td>delaying predicate execution</td>
<td>8, 387</td>
</tr>
<tr>
<td>delete/1</td>
<td>509, 512</td>
</tr>
<tr>
<td>delete/2</td>
<td>873, 874</td>
</tr>
<tr>
<td>delete/3</td>
<td>159, 223, 224, 663, 664</td>
</tr>
<tr>
<td>delete_after/2</td>
<td>873, 874</td>
</tr>
<tr>
<td>delete_directory/1</td>
<td>166, 237, 250, 382</td>
</tr>
<tr>
<td>delete_files/1</td>
<td>873, 874</td>
</tr>
<tr>
<td>delete_files_nofail/1</td>
<td>377, 378</td>
</tr>
<tr>
<td>delete_files/1</td>
<td>159, 223, 224, 663, 664</td>
</tr>
<tr>
<td>delete_non_ground/3</td>
<td>223, 224</td>
</tr>
<tr>
<td>delete_on_ctrlc/2</td>
<td>321</td>
</tr>
<tr>
<td>depends on top/2</td>
<td>159, 223, 224, 663, 664</td>
</tr>
<tr>
<td>dependent files</td>
<td>705</td>
</tr>
<tr>
<td>depth first iterative deepening</td>
<td>427</td>
</tr>
<tr>
<td>depth limit</td>
<td>427</td>
</tr>
<tr>
<td>derived_from/2</td>
<td>465, 467</td>
</tr>
<tr>
<td>describe/1</td>
<td>347, 349</td>
</tr>
<tr>
<td>destroy/1</td>
<td>465, 468</td>
</tr>
<tr>
<td>destructor</td>
<td>452</td>
</tr>
<tr>
<td>destructor/0</td>
<td>447, 452, 529, 530</td>
</tr>
<tr>
<td>det_hook/det_hook_rt</td>
<td>606</td>
</tr>
<tr>
<td>det_try/3</td>
<td>411</td>
</tr>
<tr>
<td>detach_attribute/1</td>
<td>151, 152</td>
</tr>
<tr>
<td>detcond/1</td>
<td>407, 408</td>
</tr>
<tr>
<td>determine goal</td>
<td>407</td>
</tr>
<tr>
<td>determine/2</td>
<td>407</td>
</tr>
<tr>
<td>development environment</td>
<td>20, 25, 898, 900, 907</td>
</tr>
<tr>
<td>dgraph/1</td>
<td>669</td>
</tr>
<tr>
<td>dgraph_to_ugraph/2</td>
<td>669</td>
</tr>
<tr>
<td>dic_get/3</td>
<td>305, 306</td>
</tr>
<tr>
<td>dic_lookup/3</td>
<td>305</td>
</tr>
<tr>
<td>dic_lookup/4</td>
<td>305, 306</td>
</tr>
<tr>
<td>dic_node/2</td>
<td>305, 306</td>
</tr>
<tr>
<td>dic_replace/4</td>
<td>305, 306</td>
</tr>
<tr>
<td>dict</td>
<td>325, 673</td>
</tr>
<tr>
<td>dict2varnames1/2</td>
<td>689, 690</td>
</tr>
<tr>
<td>dictionary/1</td>
<td>277, 280, 305, 839</td>
</tr>
<tr>
<td>dictionary/5</td>
<td>305</td>
</tr>
<tr>
<td>dictionary/6</td>
<td>819</td>
</tr>
<tr>
<td>dictionary_insert/5</td>
<td>821, 822</td>
</tr>
<tr>
<td>difference/3</td>
<td>223, 228</td>
</tr>
<tr>
<td>directives</td>
<td>77</td>
</tr>
<tr>
<td>directory_files/2</td>
<td>167, 237, 246, 383</td>
</tr>
<tr>
<td>directoryname/1</td>
<td>505</td>
</tr>
<tr>
<td>display_list/1</td>
<td>377, 379</td>
</tr>
<tr>
<td>display/1</td>
<td>121, 126, 147, 191</td>
</tr>
<tr>
<td>display/2</td>
<td>121, 126, 127, 191</td>
</tr>
<tr>
<td>display_list/1</td>
<td>147, 149</td>
</tr>
<tr>
<td>display_string/1</td>
<td>147, 149</td>
</tr>
<tr>
<td>display/1</td>
<td>147, 149</td>
</tr>
<tr>
<td>do_on_abolish/1</td>
<td>93, 495</td>
</tr>
<tr>
<td>do_not_free/2</td>
<td>476, 485, 486</td>
</tr>
<tr>
<td>do_on_abolish/1</td>
<td>181, 185</td>
</tr>
<tr>
<td>docstring/1</td>
<td>269, 277, 278, 280, 281, 282</td>
</tr>
<tr>
<td>documentation generator</td>
<td>7, 27</td>
</tr>
<tr>
<td>downloading emacs</td>
<td>900, 908</td>
</tr>
<tr>
<td>downloading new versions</td>
<td>9, 895</td>
</tr>
<tr>
<td>downloading, latest versions</td>
<td>911</td>
</tr>
<tr>
<td>dvips</td>
<td>709</td>
</tr>
<tr>
<td>dyn_load_cfg_module_into_make/1</td>
<td>375, 376</td>
</tr>
<tr>
<td>dynamic</td>
<td>159, 161, 162, 175, 185, 606, 631, 643, 647, 817, 819, 821, 833, 843, 847, 853, 855</td>
</tr>
<tr>
<td>dynamic predicate</td>
<td>137</td>
</tr>
<tr>
<td>dynamic/1</td>
<td>161, 181, 185, 447, 449</td>
</tr>
<tr>
<td>dynamic_search_path/1</td>
<td>40, 43</td>
</tr>
</tbody>
</table>

**E**

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges/2</td>
<td>673, 674</td>
</tr>
<tr>
<td>edges_to_lgraph/2</td>
<td>669, 670</td>
</tr>
<tr>
<td>edges_to_ugraph/2</td>
<td>669, 670</td>
</tr>
<tr>
<td>EDIPIA</td>
<td>10</td>
</tr>
</tbody>
</table>
form_default/3 | 569, 573
form_dict/1 | 581, 585
form_empty_value/1 | 569, 572
form_request_method/1 | 569, 575
form_value/1 | 581, 586
format | 53, 159, 166, 311, 345, 375, 493, 497, 509, 637, 643, 647, 749, 843
format/2 | 166, 217, 218
format/3 | 166, 217, 218
formatting commands | 269
formatting/2 | 497, 498
forward/2 | 873, 874
Francisco Bueno | 9
free variable | 365
freeze/2 | 152, 415
FROM-clauses | 620
frozon/2 | 415
func/1 | 281
function/1 | 401
functional syntax | 8, 387
functions | 7, 369, 370, 371
functor of a goal | 619
functor/3 | 91
fuzzy/1 | 435, 436
fuzzy_discrete/1 | 435
fuzzy_predicate/1 | 435, 436
fuzzybody/1 | 435, 438
gengraph1 | 730
gengraph2 | 731
genmulibar | 729
Gerda Janssens | 10
German Puebla | 9
get_alias_path/0 | 61
get_arch/1 | 155
get_attribute/2 | 151, 152
get_byte/1 | 165, 199, 200
get_byte/2 | 165, 199, 200, 201
get_char/1 | 164, 199, 202
get_char/2 | 164, 199, 202
get_code/1 | 121
get_code/2 | 121
get_cookies/1 | 569, 573
get_definition_dictionary/2 | 821
get_dictionaries/2 | 855, 863
get_dirs/2 | 377, 381
get_environment/2 | 855, 863
get_environment_name/2 | 855, 859
get_environment_type/2 | 855, 860
get_first_parsed/3 | 855, 866
get_form_input/1 | 569, 572, 576
get_form_value/3 | 569, 572
get_general_options/1 | 881, 883
get_global/2 | 403
get_indentation/2 | 855, 861
get_line/1 | 307
get_line/2 | 307
get_os/1 | 155
get_parsed/2 | 855, 862
get_pid/1 | 169, 237, 241, 385
g_get_primes/2 | 665
get_prototype_definition/2 | 847, 848
get_prototype_dictionary/2 | 821, 822
get_prototype_interface/2 | 847
get_rec_dirs/2 | 377, 381
grec_dirs/3 | 377, 382
grec_dirs/4 | 377, 389
get_row_number/2 | 855, 860
get_stream/2 | 317
get_type/2 | 627
get1_code/1 | 121, 122
get1_code/2 | 121, 122
getcounter/2 | 661
getct/2 | 121, 125

G

G assert body/1 | 277, 281
garbage collection | 254, 257
garbage_collect/0 | 170, 253, 255
garbage_collection_option/1 | 256
gc/0 | 132, 134
gc_result/1 | 257
gcc | 904
gcd/2 | 109
genbar1 | 728
genbar2 | 728
genbar3 | 729
genbar4 | 729
generate_plot/2 | 881, 883, 884
generate_plot/3 | 881, 883, 884
generator/2 | 833
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Page Numbers</th>
<th>Source/Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>issue_debug_messages/1</td>
<td>311, 314, 315</td>
<td></td>
</tr>
<tr>
<td>iterative deepening-based execution</td>
<td>9, 387</td>
<td></td>
</tr>
<tr>
<td>iterative-deepening</td>
<td>427</td>
<td></td>
</tr>
</tbody>
</table>

**J**

- Jan Maluzynski                                                      | 10          |
- Java event handling from Prolog                                    | 636         |
- Java exception handling from Prolog                               | 637         |
- Java to Prolog interface                                           | 9, 473      |
- java_add_listener/3                                                | 643         |
- java_connect/2                                                     | 637, 638    |
- java_constructor/1                                                 | 637, 639    |
- java_create_object/2                                               | 637, 639    |
- java_debug/1                                                       | 647, 649    |
- java_debug_redo/1                                                  | 647, 649    |
- java_delete_object/1                                               | 637, 640    |
- java_disconnect/0                                                   | 637, 638    |
- java_event/1                                                       | 637, 639    |
- java_field/1                                                       | 637, 639    |
- java_get_value/2                                                    | 637, 641    |
- java_invoke_method/2                                               | 637, 640    |
- java_method/1                                                      | 637, 640    |
- java_object/1                                                      | 637, 639    |
- java_query/2                                                       | 647, 648    |
- java_remove_listener/3                                              | 637, 642    |
- java_response/2                                                    | 647, 648    |
- java_set_value/2                                                    | 637, 641    |
- java_start/0                                                        | 637         |
- java_start/1                                                        | 637         |
- java_start/2                                                        | 637, 638    |
- java_stop/0                                                        | 637, 638    |
- java_use_module/1                                                   | 637, 639    |
- javalli/javasock                                                    | 637, 643    |
- javall/jtopl                                                       | 647         |
- javart                                                               | 647         |
- Johan Widen                                                        | 10          |
- John Gallagher                                                     | 10          |
- join_socket_interface/0                                            | 647, 648    |
- jtopl                                                               | 647         |
- justify_entry/1                                                     | 541, 542    |
- justify_text/1                                                      | 563, 565    |
read_top_level/3 .................................. 162, 187, 188
read_vrml_file/2 .................................. 843, 844
readf/2 ............................................. 377, 380
reading/4 ........................................... 835
reading/5 ........................................... 835, 836
reading/6 ........................................... 835, 836
rebuild_foreign_interface/1 ......................... 493
rebuild_foreign_interface_explicit_decls/2 ........ 493, 494
rebuild_foreign_interface_object/1 .................. 493, 495
receive_confirm/2 .................................. 509, 512
receive_event/2 .................................... 509, 511
receive_list/2 ..................................... 509, 511
receive_result/2 ................................... 509, 510
records/3 ......................................... 171, 261
recorded/3 ........................................ 171, 261, 262
records ............................................. 8, 387, 397
recordz/3 ......................................... 171, 261
recursive level .................................... 38
redefined .......................................... 451
redefining/1 ...................................... 78
RedHat 5.0 .......................................... 904
reduceindentation/3 ................................ 855, 862
reexport/1 .......................................... 69
reexport/2 .......................................... 69
reference/1 ........................................ 141
references, to Ciao ................................ 5
referring to Ciao ................................... 5
regedit ............................................ 910
regtype assertion .................................. 286, 287
regtype/1 ........................................... 79, 85, 286, 287
regtype/2 .......................................... 286, 287
regular expressions ................................ 349
regular expressions ................................ 667
regular type ........................................ 286
regular type abstractions ............................. 286
regular type definitions ............................. 283
regular type expression ............................. 286
regular types ...................................... 283
relation name ...................................... 619
relational databases ................................ 603
relief_type/1 ...................................... 515, 518
rem/2 ............................................. 108
remove/ciao_client_rt ............................... 471
remove_code/3 ..................................... 855, 866
remove_comments/4 ................................ 835, 837
rename/2 .......................................... 689, 691
rename_file/2 ...................................... 166, 237, 250, 377, 378, 382
repeat/0 ........................................... 73, 75
replace_strings_in_file/3 .......................... 377, 380
reporting bugs .................................... 9, 895, 912
reserved_words/1 .................................. 817, 818
retract/1 .......................................... 161, 181, 183
retract_fact/1 ....................................... 137, 139, 140, 589, 592, 593, 599,
600, 603, 607
retract_fact_nb/1 .................................... 137, 140
retractall/1 ........................................ 161, 181, 183
retractall_fact/1 .................................... 137, 139, 594, 603, 608
retrieve_list_of_values/2 ......................... 734, 738
retrieve_range/2 ................................... 734, 736
retrieve_store/2 ................................... 734, 736
Roger Nasr ......................................... 10, 38
rooted_subgraph/3 ................................ 673, 675
round/1 ............................................ 108
row/1 ............................................ 798, 799
row_value/1 ....................................... 515, 520
rowspan_value/1 .................................. 515, 520
run-time checks ................................... 272
run-time libraries ................................ 898
run_tester/10 ...................................... 889
running programs ................................ 18, 19, 24, 25
running_queries/2 .................................. 643, 645

S
s_assert_body/1 .................................... 277, 280
safe_write/2 ....................................... 367
Saumya Debray .................................... 10
scattergraph_b1/12 ................................ 730, 776, 780
scattergraph_b1/8 ................................ 730, 776, 779, 780
scattergraph_b2/12 ................................ 731, 785, 789
scattergraph_b2/8 ................................ 731, 785, 789
scattergraph_w1/12 ................................ 730, 776, 781
scattergraph_w1/8 ................................ 730, 776, 780, 790
scattergraph_w2/12 ................................ 731, 785, 791
scattergraph_w2/13 ................................ 791
<table>
<thead>
<tr>
<th>sourcename/1</th>
<th>610, 625</th>
</tr>
</thead>
<tbody>
<tr>
<td>sockets</td>
<td>904</td>
</tr>
<tr>
<td>sockets/sockets</td>
<td>367, 509, 647, 655, 749</td>
</tr>
<tr>
<td>sockets/sockets_c</td>
<td>363</td>
</tr>
<tr>
<td>sockets/sockets_io</td>
<td>647</td>
</tr>
<tr>
<td>Solaris</td>
<td>902</td>
</tr>
<tr>
<td>sort... 53, 159, 166, 177, 191, 289, 339, 377, 418, 669, 673, 677, 679, 685, 689</td>
<td></td>
</tr>
<tr>
<td>sort/2</td>
<td>166, 231</td>
</tr>
<tr>
<td>sort_dict/2</td>
<td>689, 690</td>
</tr>
<tr>
<td>source directory</td>
<td>898</td>
</tr>
<tr>
<td>source-level debugger</td>
<td>45</td>
</tr>
<tr>
<td>source-level debugging</td>
<td>45, 46, 48, 49</td>
</tr>
<tr>
<td>sourcename/1</td>
<td>43, 111, 116, 117</td>
</tr>
<tr>
<td>sourcenames/1</td>
<td>43</td>
</tr>
<tr>
<td>specifications</td>
<td>269</td>
</tr>
<tr>
<td>spy</td>
<td>51</td>
</tr>
<tr>
<td>spy-points</td>
<td>45, 47</td>
</tr>
<tr>
<td>spy/1</td>
<td>46, 47, 52, 53, 54</td>
</tr>
<tr>
<td>SQL</td>
<td>603, 615</td>
</tr>
<tr>
<td>SQL attributes</td>
<td>619</td>
</tr>
<tr>
<td>SQL query</td>
<td>609, 613, 616</td>
</tr>
<tr>
<td>SQL server</td>
<td>616</td>
</tr>
<tr>
<td>SQL table names</td>
<td>619</td>
</tr>
<tr>
<td>SQL tables</td>
<td>615, 619</td>
</tr>
<tr>
<td>SQL-like database interface</td>
<td>9, 473</td>
</tr>
<tr>
<td>sql__attribute/4</td>
<td>615, 616, 619, 633</td>
</tr>
<tr>
<td>sql__relation/3</td>
<td>615, 616, 618, 633</td>
</tr>
<tr>
<td>sql_get_tables/2</td>
<td>615, 606, 609</td>
</tr>
<tr>
<td>sql_goal_tr/2</td>
<td>631</td>
</tr>
<tr>
<td>sql_persistent/3</td>
<td>607, 608, 612</td>
</tr>
<tr>
<td>sql_persistent_location</td>
<td>612</td>
</tr>
<tr>
<td>sql_persistent_location/2</td>
<td>606, 611</td>
</tr>
<tr>
<td>sql_persistent_tr/2</td>
<td>631</td>
</tr>
<tr>
<td>sql_query/3</td>
<td>606, 609, 613</td>
</tr>
<tr>
<td>sql_query_one_tuple/3</td>
<td>613</td>
</tr>
<tr>
<td>sql_table_types/3</td>
<td>606, 610</td>
</tr>
<tr>
<td>sqlstring/1</td>
<td>616, 617, 625, 626</td>
</tr>
<tr>
<td>sqlterm2string/2</td>
<td>616, 618</td>
</tr>
<tr>
<td>sqltype/1</td>
<td>611, 618, 619, 627</td>
</tr>
<tr>
<td>sqltypes</td>
<td>611, 618</td>
</tr>
<tr>
<td>sqrt/1</td>
<td>108</td>
</tr>
<tr>
<td>srandom/1</td>
<td>683, 684</td>
</tr>
<tr>
<td>standalone compiler</td>
<td>17, 899, 901</td>
</tr>
<tr>
<td>standalone utilities</td>
<td>9, 693</td>
</tr>
<tr>
<td>standard total ordering</td>
<td>95</td>
</tr>
<tr>
<td>start_socket_interface/2</td>
<td>647</td>
</tr>
<tr>
<td>start_threads/2</td>
<td>647, 649</td>
</tr>
<tr>
<td>start_vrmlScene/4</td>
<td>835, 837</td>
</tr>
<tr>
<td>status, this manual</td>
<td>3</td>
</tr>
<tr>
<td>steps/2</td>
<td>289, 292</td>
</tr>
<tr>
<td>steps_lb/2</td>
<td>289, 292</td>
</tr>
<tr>
<td>streams</td>
<td>331, 347, 493, 592</td>
</tr>
<tr>
<td>streams_basic:open/3</td>
<td>329</td>
</tr>
<tr>
<td>string/1</td>
<td>79, 83</td>
</tr>
<tr>
<td>string/3</td>
<td>307, 309</td>
</tr>
<tr>
<td>string2term/2</td>
<td>319</td>
</tr>
<tr>
<td>streamcommand/1</td>
<td>274, 278, 280, 281, 282</td>
</tr>
<tr>
<td>strings</td>
<td>311, 504, 509, 513, 529, 569, 579, 749</td>
</tr>
<tr>
<td>strip_clean/2</td>
<td>855, 864</td>
</tr>
<tr>
<td>strip_exposed/2</td>
<td>855, 864</td>
</tr>
<tr>
<td>strip_interface/2</td>
<td>855, 864</td>
</tr>
<tr>
<td>strip_restricted/2</td>
<td>855, 864</td>
</tr>
<tr>
<td>struct/1</td>
<td>79, 81</td>
</tr>
<tr>
<td>style sheets</td>
<td>17, 23, 901, 908</td>
</tr>
<tr>
<td>style_type/1</td>
<td>551, 553</td>
</tr>
<tr>
<td>sub_atom/4</td>
<td>99, 104</td>
</tr>
<tr>
<td>sub_atom/5</td>
<td>165, 205</td>
</tr>
<tr>
<td>sub_times/3</td>
<td>877, 880</td>
</tr>
<tr>
<td>sublist/2</td>
<td>223, 228</td>
</tr>
<tr>
<td>subordlist/2</td>
<td>223, 228</td>
</tr>
<tr>
<td>subterm</td>
<td>52</td>
</tr>
<tr>
<td>subtract/3</td>
<td>663, 664</td>
</tr>
<tr>
<td>success assertion</td>
<td>271, 272</td>
</tr>
<tr>
<td>success/1</td>
<td>270, 271</td>
</tr>
<tr>
<td>sum_list/2</td>
<td>665, 666</td>
</tr>
<tr>
<td>sum_list/3</td>
<td>665, 666</td>
</tr>
<tr>
<td>sum_list_of_lists/2</td>
<td>665, 666</td>
</tr>
<tr>
<td>sum_list_of_lists/3</td>
<td>665, 666</td>
</tr>
<tr>
<td>super class</td>
<td>Swedish Institute of Computer Science</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>sybase2sqltype/2</td>
<td></td>
</tr>
<tr>
<td>sybase2sqltypes_list/2</td>
<td></td>
</tr>
<tr>
<td>sybasetype/1</td>
<td></td>
</tr>
<tr>
<td>symbol/1</td>
<td></td>
</tr>
<tr>
<td>symbol_option/1</td>
<td></td>
</tr>
<tr>
<td>symbol_result/1</td>
<td></td>
</tr>
<tr>
<td>symbolic_link/2</td>
<td></td>
</tr>
<tr>
<td>symbolic_link/3</td>
<td></td>
</tr>
<tr>
<td>symnames</td>
<td></td>
</tr>
<tr>
<td>syntax of regular types</td>
<td></td>
</tr>
<tr>
<td>SYSCALL/1</td>
<td></td>
</tr>
<tr>
<td>system</td>
<td></td>
</tr>
<tr>
<td>system/1</td>
<td></td>
</tr>
<tr>
<td>system/2</td>
<td></td>
</tr>
<tr>
<td>system_extra</td>
<td></td>
</tr>
<tr>
<td>system_lib/1</td>
<td></td>
</tr>
<tr>
<td>terms</td>
<td></td>
</tr>
<tr>
<td>text_characters/1</td>
<td></td>
</tr>
<tr>
<td>terms_check</td>
<td></td>
</tr>
<tr>
<td>terms_file_to_vrml</td>
<td></td>
</tr>
<tr>
<td>terms_file_to_vrml_file/2</td>
<td></td>
</tr>
<tr>
<td>terms_to_vrml</td>
<td></td>
</tr>
<tr>
<td>terms_to_vrml_file/2</td>
<td></td>
</tr>
<tr>
<td>terms_vars</td>
<td></td>
</tr>
<tr>
<td>tester/tester</td>
<td></td>
</tr>
<tr>
<td>tester_func/1</td>
<td></td>
</tr>
<tr>
<td>textvariable_entry/1</td>
<td></td>
</tr>
<tr>
<td>textvariable_label/1</td>
<td></td>
</tr>
<tr>
<td>textvariablevalue_number/1</td>
<td></td>
</tr>
<tr>
<td>textvariablevalue_string/1</td>
<td></td>
</tr>
<tr>
<td>this_module/1</td>
<td></td>
</tr>
<tr>
<td>throw/1</td>
<td></td>
</tr>
<tr>
<td>time/1</td>
<td></td>
</tr>
<tr>
<td>time_option/1</td>
<td></td>
</tr>
<tr>
<td>time_result/1</td>
<td></td>
</tr>
<tr>
<td>title/1</td>
<td></td>
</tr>
<tr>
<td>Tcl/tk interface</td>
<td></td>
</tr>
<tr>
<td>tag_attrib/1</td>
<td></td>
</tr>
<tr>
<td>tar</td>
<td></td>
</tr>
<tr>
<td>target/1</td>
<td></td>
</tr>
<tr>
<td>tcl_delete/1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
why the name Ciao
width/1
width_value/1
Win32
window_class
windows shortcut
WinZip
with/2
Wlodek Drabent

WWW browser
WWW, interfacing with

X
xbarelement1
xbarelement2
xbarelement3
xbarelement4
xdr_handle/xdr_types
xdr_node/1
xdr_tree/1
xdr_tree/3
xdr_xpath/2
xdr2html/2
xdr2html/4
xelement/1
XML
xml_index/1
xml_index_query/3
xml_index_to_file/2
xml_parse/3
xml_parse_match/3
xml_path/xml_path_types
xml_query/3
xml_search/3
xml_search_match/3
xml2terms/2
xrefs/xrefsread

Y
yelement/1