The lpdoc Documentation Generator

An Automatic Documentation Generator for (C)LP Systems
REFERENCE MANUAL
The Ciao Documentation Series
http://www.ciaohome.org/
Generated/Printed on: 15 August 2011
Technical Report CLIP 5/97.1-3.0
Version 3.0 (2011/7/7, 16:33:15 CEST)

Edited by:
Manuel Hermenegildo
José Francisco Morales
Table of Contents

Summary................................................................. 1

1 Introduction....................................................... 3
  1.1 Overview of this document.................................. 3
  1.2 lpdoc operation - source and target files.............. 3
  1.3 lpdoc usage.................................................... 5
  1.4 Version/Change Log.......................................... 5

PART I - LPdoc Reference Manual .................. 13

2 Generating Installing and Accessing Manuals . . 15
  2.1 Generating a manual from the Ciao Emacs mode......... 15
  2.2 Generating a manual......................................... 15
  2.3 Working on a manual......................................... 17
  2.4 Cleaning up the documentation directory............... 17
  2.5 Installing a generated manual in a public area........ 18
  2.6 Enhancing the documentation being generated.......... 18
  2.7 Accessing on-line manuals................................ 19
      2.7.1 Accessing html manuals.............................. 19
      2.7.2 Accessing info manuals.............................. 19
      2.7.3 Accessing man manuals.............................. 20
      2.7.4 Putting it all together............................. 21
  2.8 Some usage tips............................................. 21
      2.8.1 Ensuring Compatibility with All Supported Target
          Formats................................................. 21
      2.8.2 Writing comments which document version/patch
          changes............................................... 21
      2.8.3 Documenting Libraries and/or Applications......... 22
      2.8.4 Documenting files which are not modules........... 22
      2.8.5 Splitting large documents into parts............... 22
      2.8.6 Documenting reexported predicates................ 23
      2.8.7 Separating the documentation from the source file... 23
      2.8.8 Generating auxiliary files (e.g. READMEs)......... 23
  2.9 Troubleshooting............................................. 24

3 Documentation Mark-up Language and
   Declarations................................................... 25
  3.1 Usage and interface (comments)......................... 25
  3.2 Documentation on exports (comments)................... 25
      docstring/1 (prop)...................................... 25
      stringcommand/1 (prop)................................ 26
      version_descriptor/1 (regtype)........................ 32
      filetype/1 (regtype).................................... 32
      doc_id_type/3 (pred).................................. 32
  3.3 Documentation on internals (comments)................ 32
      doc/2 (decl)........................................... 32
      version_number/1 (regtype)............................ 38
      ymd_date/1 (regtype).................................. 38
4 The Ciao assertion package ................. 41

4.1 More info .................................. 41
4.2 Some attention points ...................... 41
4.3 Usage and interface (assertions_doc) ....... 42
4.4 Documentation on new declarations (assertions_doc) ...... 42
   (pred)/1 (decl) ................................ 42
   (pred)/2 (decl) ................................ 43
   (texec)/1 (decl) ............................... 43
   (texec)/2 (decl) ................................ 43
   (calls)/1 (decl) ............................... 43
   (calls)/2 (decl) ................................ 43
   (success)/1 (decl) ............................ 44
   (success)/2 (decl) ............................ 44
   (test)/1 (decl) ............................... 44
   (test)/2 (decl) ................................ 44
   (comp)/1 (decl) ............................... 45
   (comp)/2 (decl) ................................ 45
   (prop)/1 (decl) ............................... 45
   (prop)/2 (decl) ................................ 46
   (entry)/1 (decl) .............................. 46
   (exit)/1 (decl) ............................... 46
   (exit)/2 (decl) ................................ 47
   (modedef)/1 (decl) ........................... 47
   (decl)/1 (decl) .............................. 47
   (decl)/2 (decl) ................................ 47
   doc/2 (decl) ................................. 47
   comment/2 (decl) ............................. 48

4.5 Documentation on exports (assertions_doc) .......... 48
   check/1 (pred) ............................... 48
   trust/1 (pred) ............................... 48
   true/1 (pred) ............................... 49
   false/1 (pred) ............................... 49

5 Types and properties related to assertions .... 51

5.1 Usage and interface (assertions_props) .......... 51
5.2 Documentation on exports (assertions_props) .......... 51
   assert_body/1 (regtype) ..................... 51
   head_pattern/1 (prop) ....................... 52
   complex_arg_property/1 (regtype) ............. 52
   property_conjunction/1 (regtype) ............. 53
   property_startterm/1 (regtype) .............. 53
   complex_goal_property/1 (regtype) .......... 53
   nbody/1 (prop) .............................. 54
   dictionary/1 (regtype) ..................... 54
   c_assert_body/1 (regtype) ................... 54
   s_assert_body/1 (regtype) ................... 54
   g_assert_body/1 (regtype) ................... 55
   assert_status/1 (regtype) ................... 56
   assert_type/1 (regtype) ..................... 56
   predfunctor/1 (regtype) ..................... 56
   propfunctor/1 (regtype) ..................... 56
   docstring/1 (prop) .......................... 56
6 Declaring regular types .......................... 57
6.1 Defining properties .................................. 57
6.2 Usage and interface (regtypes_doc) .................. 59
6.3 Documentation on new declarations (regtypes_doc) .... 60
   (regtype)/1 (decl) .................................. 60
   (regtype)/2 (decl) .................................. 60

7 Basic data types and properties ................. 61
7.1 Usage and interface (basic_props) ................. 61
7.2 Documentation on imports (basic_props) ............ 61
   term/1 (regtype) .................................. 61
   int/1 (regtype) .................................. 62
   nnegint/1 (regtype) ............................... 62
   flt/1 (regtype) .................................. 63
   num/1 (regtype) .................................. 63
   atm/1 (regtype) .................................. 64
   struct/1 (regtype) ................................ 64
   gnd/1 (regtype) .................................. 65
   gndstr/1 (regtype) ................................ 65
   constant/1 (regtype) ................................ 66
   callable/1 (regtype) ................................ 66
   operator_specifier/1 (regtype) ....................... 67
   list/1 (regtype) .................................. 67
   list/2 (regtype) .................................. 68
   nlist/2 (regtype) .................................. 68
   member/2 (prop) .................................. 69
   sequence/2 (regtype) ................................ 69
   sequence_or_list/2 (regtype) ......................... 70
   character_code/1 (regtype) .......................... 70
   string/1 (regtype) .................................. 71
   num_code/1 (regtype) ................................ 71
   predname/1 (regtype) ................................ 71
   atm_or_atm_list/1 (regtype) ......................... 72
   compat/2 (prop) .................................. 72
   inst/2 (prop) .................................. 72
   iso/1 (prop) .................................. 73
   deprecated/1 (prop) ................................ 73
   not_further_inst/2 (prop) ............................ 73
   sideff/2 (prop) .................................. 74
   (regtype)/1 (prop) .................................. 74
   native/1 (prop) .................................. 74
   native/2 (prop) .................................. 74
   no_rtcheck/1 (prop) ................................ 74
   eval/1 (prop) .................................. 75
   equiv/2 (prop) .................................. 75
   bind_ins/1 (prop) .................................. 75
   error_free/1 (prop) ................................ 75
   memo/1 (prop) .................................. 75
   filter/2 (prop) .................................. 75
   flag_values/1 (regtype) ................................ 75
   pe_type/1 (prop) .................................. 76
8 Properties which are native to analyzers .... 77

8.1 Usage and interface (native_props) .................. 77
8.2 Documentation on exports (native_props) ....... 77

clique/1 (prop) .......................................... 77
clique_1/1 (prop) ........................................ 78
constraint/1 (prop) ....................................... 78
covered/1 (prop) ......................................... 78
covered/2 (prop) ......................................... 78
exception/1 (prop) ........................................ 78
exception/2 (prop) ........................................ 78
fails/1 (prop) ............................................... 79
finite_solutions/1 (prop) ................................ 79
have_choicepoints/1 (prop) ............................. 79
indep/1 (prop) ............................................ 79
indep/2 (prop) ............................................ 79
is_det/1 (prop) ........................................... 79
linear/1 (prop) ............................................ 80
mshare/1 (prop) ........................................... 80
mut_exclusive/1 (prop) ................................ 80
no_choicepoints/1 (prop) ............................... 80
no_exception/1 (prop) .................................... 81
no_exception/2 (prop) .................................... 81
no_signal/1 (prop) ....................................... 81
no_signal/2 (prop) ....................................... 81
non_det/1 (prop) .......................................... 81
nonground/1 (prop) ....................................... 81
not_covered/1 (prop) .................................... 81
not_fails/1 (prop) ....................................... 81
not_mut_exclusive/1 (prop) ............................ 82
num_solutions/2 (prop) ................................ 82
solutions/2 (prop) ....................................... 82
possibly_fails/1 (prop) .................................. 82
possibly_nondet/1 (prop) ............................... 83
relations/2 (prop) ....................................... 83
sideff_hard/1 (prop) .................................... 83
sideff_pure/1 (prop) ..................................... 83
sideff_soft/1 (prop) ..................................... 83
signal/1 (prop) .......................................... 83
signal/2 (prop) .......................................... 83
signals/2 (prop) ......................................... 84
size/2 (prop) ............................................. 84
size/3 (prop) ............................................. 84
size_lb/2 (prop) ......................................... 84
size_o/2 (prop) .......................................... 84
size_ub/2 (prop) ......................................... 84
size_metric/3 (prop) ................................... 84
size_metric/4 (prop) ................................... 84
steps/2 (prop) ........................................... 85
steps_lb/2 (prop) ........................................ 85
steps_o/2 (prop) .......................................... 85
steps_ub/2 (prop) ....................................... 85
tau/1 (prop) ................................................ 85
terminates/1 (prop) ..................................... 86
test_type/2 (prop) ....................................... 86
throws/2 (prop) .......................................... 86
user_output/2 (prop) .................................... 86
9 Meta-properties ................................. 87
  9.1 Usage and interface (meta_props)............. 87
  9.2 Documentation on exports (meta_props)....... 87
    call/2 (prop) ................................ 87
    (prop)/2 (prop) .............................. 87
    (regtype)/2 (prop) .......................... 88
  9.3 Documentation on multifiles (meta_props).... 88
    callme/2 (pred) ............................. 88
  9.4 Documentation on internals (meta_props)..... 88
    prop_abs/1 (prop) ........................... 88

10 An Example - Documenting a Library Module

11 Auto Documenter Output for the Example
Module ........................................ 95
  11.1 Usage and interface (example_module) ....... 95
  11.2 Documentation on exports (example_module).... 95
    bar/1 (regtype) ............................. 95
    baz/1 (regtype) ............................. 95
    aorb/1 (regtype) ........................... 95
    tree_of/2 (regtype) ........................ 96
    list_or_aorb/2 (regtype) .................... 96
    q/2 (pred) ................................. 96
    r/1 (pred) ................................. 96
    p/1 (pred) ................................. 97
    p/5 (pred) ................................. 97
    u/3 (pred) ................................. 97
    long/1 (prop) .............................. 97
    w/1 (pred) ................................. 98
    mytype/1 (pred) ............................ 98
    t/5 (pred) ................................. 98
    s/1 (pred) ................................. 98
    q/1 (pred) ................................. 99
    list/1 (regtype) ........................... 99
  11.3 Documentation on multifiles (example_module).... 100
    p/3 (pred) ................................. 100
  11.4 Documentation on internals (example_module).... 101
    list/2 (regtype) ........................... 101
    og/2 (modedef) ............................. 101
    is/2 (pred) ................................. 102

12 Run-time checking of assertions ............... 103
  12.1 Usage and interface (rtchecks_doc) ......... 104

13 Unit Testing Library .......................... 105
  13.1 Additional notes ............................ 105
  13.2 Usage and interface (unittest_doc) ......... 106
14 Installing lpdoc .................................. 107
  14.1 Installing the source distribution (lpdoc) .................. 107
  14.2 Other software packages required (lpdoc) .......... 107

PART II - LPdoc Internals Manual ............ 109

15 Documentation Generation Library .......... 111
  15.1 Usage and interface (autodoc) ..................... 112
  15.2 Documentation on exports (autodoc) .................. 112
    index_comment/2 (pred) ............................. 112
    reset_output_dir_db/0 (pred) ....................... 112
    ensure_output_dir_prepared/2 (pred) ................ 112
    get_autodoc_opts/3 (pred) .......................... 113
    autodoc_gen_doctree/5 (pred) ....................... 113
    fmt_infodir_entry/3 (pred) .......................... 113
    autodoc_compute_grefs/3 (pred) ..................... 113
    autodoc_translate_doctree/3 (pred) .................. 113
    autodoc_finish/1 (pred) ............................ 114
    autodoc_gen_alternative/2 (pred) .................... 114
  15.3 Documentation on multifiles (autodoc) .......... 114
    autodoc_finish_hook/1 (pred) ....................... 114
    autodoc_gen_alternative_hook/2 (pred) ............ 114

16 Internal State for Documentation Generation
  .................................................... 115
  16.1 Usage and interface (autodoc_state) ............... 115
  16.2 Documentation on exports (autodoc_state) .......... 115
    supported_option/1 (prop) .......................... 115
    option_comment/2 (pred) ............................ 116
    backend_id/1 (regtype) ............................. 116
    backend_ignores_components/1 (pred) ............... 116
    backend_alt_format/2 (pred) ....................... 117
    top_suffix/2 (pred) ................................. 117
    docstate/1 (regtype) ............................... 117
    docst_backend/2 (pred) .............................. 117
    docst_currmod/2 (pred) ............................. 117
    docst_set_currmod/3 (pred) ......................... 117
    docst_opts/2 (pred) .................................. 117
    docst_set_opts/3 (pred) ............................. 117
    docst_inputfile/2 (pred) ............................ 117
    docst_new_no_src/4 (pred) ........................... 117
    docst_new_with_src/6 (pred) ....................... 118
    docst_new_sub/3 (pred) ............................. 118
    docst_message/2 (pred) ............................. 118
    docst_message/3 (pred) ............................. 118
    docst_opt/2 (pred) ................................. 118
    docst_currmod_is_main/1 (pred) .................... 118
    docst_no_components/1 (pred) ...................... 118
    docst_modname/2 (pred) ............................. 118
    labgen_init/1 (pred) ............................... 118
    labgen_clean/1 (pred) .............................. 118
    labgen_get/2 (pred) ................................. 118
    docst_mvar_lookup/3 (pred) ......................... 119
    docst_mvar_replace/4 (pred) ...................... 119
17 Documentation Abstract Syntax Tree........ 123

17.1 Usage and interface (autodoc_doctree)............. 123
17.2 Documentation on exports (autodoc_doctree)......... 123
17.3 Documentation on multifiles (autodoc_doctree)....... 127
18 Handling the Document Structure ............ 129
18.1 Usage and interface (autodoc_structure) .................. 129
18.2 Documentation on exports (autodoc_structure) ............ 129
    docstr_node/4 (pred) .................................. 129
    clean_docstr/0 (pred) .................................. 129
    parse_structure/0 (pred) ................................ 129
    standalone_docstr/1 (pred) ............................. 129
    get_mainmod/1 (pred) .................................... 129
    get_mainmod_spec/1 (pred) ............................... 130
    all_component_specs/1 (pred) ........................... 130

19 Access to Default Settings .................... 131
19.1 Usage and interface (autodoc_settings) ............... 131
19.2 Documentation on exports (autodoc_settings) ............ 131
    lpdoc_option/1 (pred) .................................. 131
    verify_settings/0 (pred) ................................ 131
    check_setting/1 (pred) .................................. 131
    setting_value_or_default/2 (pred) ....................... 132
    setting_value_or_default/3 (pred) ....................... 132
    setting_value/2 (pred) .................................. 132
    all_setting_values/2 (pred) ............................. 132
    get_command_option/1 (pred) ............................ 132
    requested_file_formats/1 (pred) ......................... 132
    load_vpaths/0 (pred) ................................... 132
    viewer/3 (pred) .......................................... 132
    xdvi/1 (pred) ........................................... 132
    xdvisize/1 (pred) ....................................... 132
    pdfview/1 (pred) ....................................... 132
    psview/1 (pred) .......................................... 132
    htmlview/1 (pred) ....................................... 132
    bibtex/1 (pred) .......................................... 133
    tex/1 (pred) ............................................... 133
    texindex/1 (pred) ......................................... 133
    dvips/1 (pred) ........................................... 133
    ps2pdf/1 (pred) .......................................... 133
    makeinfo/1 (pred) ........................................ 133
    makertf/1 (pred) ......................................... 133
    rtftohlp/1 (pred) ........................................ 133
    convertc/1 (pred) ....................................... 133

20 Texinfo Backend ............................... 137
20.1 Usage and interface (autodoc Texinfo) ................. 137
20.2 Documentation on exports (autodoc Texinfo) ............ 137
    infodir_base/2 (pred) .................................. 137
20.3 Documentation on multifiles (autodoc Texinfo) ......... 137
    autodoc_escape_string_hook/5 (pred) ................... 137
    autodoc_rw_command_hook/4 (pred) ...................... 137
    autodoc_finish_hook/1 (pred) ........................... 138
    autodoc_gen_alternative_hook/2 (pred) ................. 138

LPdoc Backends ................................. 135
21 HTML Backend .................................................. 139
  21.1 Usage and interface (autodoc_html) ..................... 139
  21.2 Documentation on multifiles (autodoc_html) ........... 139
    autodoc_escape_string_hook/5 (pred) ................. 139
    autodoc_rw_command_hook/4 (pred) ................. 139
    autodoc_finish_hook/1 (pred) ................. 139
    autodoc_gen_alternative_hook/2 (pred) ........... 140

22 Resource Handling for the HTML Backend .......... 141
  22.1 Usage and interface (autodoc_html_resources) ......... 141
  22.2 Documentation on exports (autodoc_html_resources) .... 141
    prepare_web_skel/1 (pred) .................. 141
    prepare_mathjax/0 (pred) .................. 141
    using_mathjax/1 (pred) .................. 141

23 Template Support for the HTML Backend .... 143
  23.1 Usage and interface (autodoc_html_template) .......... 143
  23.2 Documentation on exports (autodoc_html_template) .... 143
    img_url/2 (pred) .................. 143
    fmt_html_template/3 (pred) .................. 143

24 Man Pages (man) Backend ......................... 145
  24.1 Usage and interface (autodoc_man) ..................... 145
  24.2 Documentation on multifiles (autodoc_man) .......... 145
    autodoc_rw_command_hook/4 (pred) ................. 145
    autodoc_finish_hook/1 (pred) ................. 145
    autodoc_gen_alternative_hook/2 (pred) ........... 145

25 Filesystem Abstraction ............................. 147
  25.1 Usage and interface (autodoc_filesystem) .......... 147
  25.2 Documentation on exports (autodoc_filesystem) .... 147
    filename/1 (regtype) ................ 147
    basename/1 (regtype) ................ 147
    subtarget/1 (regtype) ................. 147
    file_format_name/2 (pred) ............... 148
    supported_file_format/1 (pred) ........... 148
    file_format_provided_by_backend/3 (pred) .... 148
    clean_fs_db/0 (pred) .................. 148
    get_output_dir/2 (pred) .................. 148
    get_cache_dir/2 (pred) .................. 148
    ensure_output_dir/1 (pred) .................. 148
    ensure_cache_dir/1 (pred) .................. 148
    main_absfile_in_format/2 (pred) .............. 148
    main_absfile_for_subtarget/3 (pred) .......... 148
    absfile_for_aux/3 (pred) .................. 149
    absfile_for_subtarget/4 (pred) ............... 149
    main_output_name/2 (pred) ................. 149
    get_subbase/3 (pred) .................. 149
    absfile_to_refile/3 (pred) .................. 149
    clean_all/0 (pred) .................. 149
    clean_docs_no_texi/0 (pred) .................. 149
    clean_all_temporal/0 (pred) .................. 149
    clean_intermediate/0 (pred) .................. 150
    clean_tex_intermediate/0 (pred) .................. 150
26 Indexing Commands (Definition and Formatting) ......................................................... 151
26.1 Usage and interface (autodoc_index) ................................................................. 151
26.2 Documentation on exports (autodoc_index) ......................................................... 151
    get_idxsub/2 (pred) .......................................................................................... 151
    get_idxbase/3 (pred) ......................................................................................... 151
    typeindex/5 (pred) ............................................................................................ 151
    idx_get_indices/3 (pred) .................................................................................... 132
    is_index_cmd/1 (pred) ....................................................................................... 152
    codetype/1 (pred) ............................................................................................... 152
    normalize_index_cmd/3 (pred) ........................................................................... 152
    fmt_idx_env/7 (pred) ......................................................................................... 152
    fmt_index/3 (pred) ............................................................................................. 152

27 Database of Documentation References ... ............................................................... 153
27.1 Usage and interface (autodocRefsdb) .............................................................. 153
27.2 Documentation on exports (autodocRefsdb) ....................................................... 153
    computeRefs_and_biblio/1 (pred) ................................................................. 153
    prepare_current_refs/1 (pred) ......................................................................... 153
    clean_current_refs/1 (pred) ............................................................................ 153
    secttree/1 (regtype) ......................................................................................... 154
    secttree_resolve/3 (pred) ................................................................................ 154

28 Error Messages ........................................................................................................ 155
28.1 Usage and interface (autodoc_errors) ............................................................... 155
28.2 Documentation on exports (autodoc_errors) ..................................................... 155
    error_text/3 (pred) ........................................................................................... 155

29 Resolution of Bibliographical References .... ......................................................... 157
29.1 Usage and interface (autodoc_bibrefs) .............................................................. 157
29.2 Documentation on exports (autodoc_bibrefs) ...................................................... 157
    resolve_bibliography/1 (pred) ......................................................................... 157

30 Auxiliary Definitions ............................................................................................. 159
30.1 Usage and interface (autodoc_aux) ................................................................. 159
30.2 Documentation on exports (autodoc_aux) ........................................................ 159
    read_file/2 (pred) ............................................................................................ 159
    ascii_blank_lines/2 (pred) .............................................................................. 159
    sh_exec/2 (pred) ............................................................................................... 159

31 Image Handling ......................................................................................................... 161
31.1 Usage and interface (autodoc_images) ............................................................... 161
31.2 Documentation on exports (autodoc_images) ...................................................... 161
    locate_and_convert_image/4 (pred) .............................................................. 161
    clean_image_cache/0 (pred) ............................................................................ 161

References .................................................................................................................... 163
Library/Module Index ................................................................................................ 165
Predicate/Method Index ................................................................................................. 167
Property Index ........................................ 169
Regular Type Index ............................... 171
Declaration Index ................................. 173
Concept Index ...................................... 175
Author Index ....................................... 177
Global Index ....................................... 179
Summary

lpdoc is an automatic program documentation generator for (C)LP systems.

lpdoc generates a reference manual automatically from one or more source files for a logic program (including ISO-Prolog, Ciao, many CLP systems, ...). It is particularly useful for documenting library modules, for which it automatically generates a description of the module interface. However, lpdoc can also be used quite successfully to document full applications and to generate nicely formatted plain ascii “readme” files. A fundamental advantage of using lpdoc to document programs is that it is much easier to maintain a true correspondence between the program and its documentation, and to identify precisely to what version of the program a given printed manual corresponds.

The quality of the documentation generated can be greatly enhanced by including within the program text:

- assertions (types, modes, etc. ...) for the predicates in the program, and
- machine-readable comments (in the “literate programming” style).

The assertions and comments included in the source file need to be written using the Ciao system assertion language. A simple compatibility library is available to make traditional (constraint) logic programming systems ignore these assertions and comments allowing normal treatment of programs documented in this way.

The documentation is currently generated in HTML or texinfo format. From the texinfo output, printed and on-line manuals in several formats (dvi, ps, info, etc.) can be easily generated automatically, using publicly available tools. lpdoc can also generate ‘man’ pages (Unix man page format) as well as brief descriptions in html or emacs info formats suitable for inclusion in an on-line index of applications. In particular, lpdoc can create and maintain fully automatically WWW and info sites containing on-line versions of the documents it produces.

The lpdoc manual (and the Ciao system manuals) are generated by lpdoc.

lpdoc is distributed under the GNU general public license.

Note: lpdoc is fully supported on Linux, Mac OS X, and other Un*x-like systems. Due to the use of several Un*x-related utilities, some documentation back-ends may require Cygwin under Win32.

This documentation corresponds to version 3.0 (2011/7/7, 16:33:15 CEST).
1 Introduction

lpdoc is an automatic program documentation generator for (C)LP systems.

lpdoc generates a reference manual automatically from one or more source files for a logic program (including ISO-Prolog [DED96], Ciao [Bue95], many CLP [JM94] systems, ...). It is particularly useful for documenting library modules, for which it automatically generates a description of the module interface. However, lpdoc can also be used quite successfully to document full applications and to generate nicely formatted plain ASCII “readme” files. A fundamental advantage of using lpdoc to document programs is that it is much easier to maintain a true correspondence between the program and its documentation, and to identify precisely to what version of the program a given printed manual corresponds.

1.1 Overview of this document

This first part of the document provides basic explanations on how to generate a manual from a set of files that already contain assertions and comments. Examples are given using the files in the examples directory provided with the lpdoc distribution.

These instructions assume that lpdoc (at least the executable and the library) is installed somewhere in your system. Installation instructions can be found in Chapter 14 [Installing lpdoc], page 107.

Other parts of this document provide:

- Documentation on the syntax and meaning of the assertions that lpdoc uses (those defined in the Ciao assertions library [PBH97, PBH98, Bue98]). These include comment assertions (containing basically documentation text), formal assertions (containing properties), and combined assertions.
- Documentation on a basic set of properties, types, etc. which are predefined in the Ciao basic_props, regtypes, native_props, and meta_props libraries. These properties, and any others defined by the user or in other Ciao libraries, can be used in program assertions.
- Documentation on the formatting commands that can be embedded in comments.

This document is also an internals manual, providing information on how the different internal parts of lpdoc are connected, which can be useful if new capabilities need to be added to the system or its libraries are used for other purposes. To this end, the document also provides:

- The documentation for the autodoc automatic documentation library, which provides the main functionality of lpdoc.
- Documentation on the predicates that define the conversion formats used (texinfo, and others), and which are in the autodocformats library.

All of the above have been generated automatically from the assertions in the corresponding sources and can also be seen as examples of the use of lpdoc.

Some additional information on lpdoc can be found in [Her00].

1.2 lpdoc operation - source and target files

The main input used by lpdoc in order to generate a manual are Prolog source files. Basically, lpdoc generates a file in the GNU texinfo format (with a .texi ending) for each Prolog file (see “The GNU Texinfo Documentation System” manual for more info on this format). The Prolog files must have a .pl ending.

If the .pl file does not define the predicates main/0 or main/1, it is assumed to be a library and it is documented as such: the .texi file generated will contain information on the interface (e.g., the predicates exported by the file, the name of the module and usage if it is a module,
etc.), in addition to any other machine readable comments included in the file (see Section 2.6 [Enhancing the documentation being generated], page 18). If, on the contrary, the file defines the predicates main/0 or main/1, it is assumed to be an application and no description of the interface is generated (see Section 2.8 [Some usage tips], page 21).

If needed, files written directly in texinfo can also be used as input files for lpdoc. These files must have a .src (instead of .texi) ending. This is needed to distinguish them from any automatically generated .texi files. Writing files directly in texinfo has the disadvantage that it may be difficult to adhere to all the conventions used by lpdoc. For example, these files will be typically used as chapters and must be written as such. Also, the set of indices used must be the same that lpdoc is generating automatically. Finally, no bibliographic citations can be used. Because of this, and because in the future lpdoc may be able to generate documentation in formats other than texinfo directly (in which case these files would not be useful), writing files in texinfo directly is discouraged. This facility was added mainly to be able to reuse parts of manuals which were already written in texinfo. Note that if a stand-alone file needs to be written (i.e., a piece of documentation that is not associated to any .pl file) it can always be written as a “dummy” .pl file (i.e., one that is not used as code), but which contains machine readable comments).

A manual can be generated either from a single source file (.pl or .src) or from a set of source files. In the latter case, then one of these files should be chosen to be the main file, and the others will be the component files. The main file is the one that will provide the title, author, date, summary, etc. to the entire document. In principle, any set of source files can be documented, even if they contain no assertions or comments. However, the presence of these will greatly improve the documentation (see Section 2.6 [Enhancing the documentation being generated], page 18).

If the manual is generated from a single main file (i.e., component/1, defined below, is empty), then the document generated will be a flat document containing no chapters. If the manual is generated from a main file and one or more components, then the document will contain chapters. The comments in the main file will be used to generate the introduction, while each of the component files will be used to generate a separate chapter. The contents of each chapter will be controlled by the contents of the corresponding component file.

As mentioned before, lpdoc typically generates texinfo files. From the texinfo files, lpdoc can generate printed and on-line manuals in several formats (dvi, ps, ascii, html, info, etc.) automatically, using different (publicly available) packages. Documentation in some other formats (e.g., manl pages) can be generated directly by lpdoc, selecting the appropriate options (see below). lpdoc can also generate directly includes generating (parts of) a master index of documents which can be placed in an installation directory and which will provide pointers to the individual manuals generated. Using this feature, lpdoc can maintain global html and/or info documentation sites automatically (see Section 2.5 [Installing a generated manual in a public area], page 18).

Additionally, lpdoc can provide some data from the main (prolog) documentation file. For this purpose the option getinfo can be used instead of specifying the format. This option reads the asked fields from getinfo variable (defined in SETTINGS.pl or via arguments with -d option). lpdoc will generate files with main documentation file-name as base-name, followed by one underscore, the asked field (got from getinfo), and the extension. The content of each of these files (so also the extension) is specified by getinfo_format, that can take the values html, ascii, texic. For example, to ask for the summary and the author fields from a prolog file called file.pl, with lpdoc documentation, we can execute the command lpdoc -d getinfo=[author,summary] getinfo. The files myfile_author.txt and myfile_summary.txt will be created. If also the option -d getinfo_format=html is used, the files will have html extension (and content).
Chapter 1: Introduction

1.3 Ipdoc usage

The following provides the different command line options available when invoking Ipdoc. This description is intended only for advanced users which might like to use Ipdoc in custom applications. Note that the normal way to use Ipdoc is by setting parameters in an SETTINGS file (see Section 2.2 [Generating a manual], page 15).

TODO: command line options not available here; need cooperation with lpmake

1.4 Version/Change Log

Version 3.0 (2011/7/7, 16:33:15 CEST)
- Major redesign of the documentation generator:
  - LPdoc redesigned to work internally with a 'doctree' representation (a-la Pillow). (Jose Morales)
  - A native HTML backend (not generated from texi). (Jose Morales)
  - Allow custom website generation from LPdoc documents. (Jose Morales)
  - Two passes for document generation, allowing resolution of bibliographical references in all backends (including HTML). (Jose Morales)
  - doc_structure/l in SETTINGS allows structure in LPdoc documents (sections can really be nested inside parts). (Jose Morales)
  - :- doc(_,_) is the recommended syntax for documentation comments now.
  - Replacing :- comment by :- doc in LPdoc code, updated documentation. (Jose Morales)
- General improvements and bug fixes:
  - Designed a logo for LPdoc. (Jose Morales)
  - LPdoc comments can now be written using %! style comment syntax. (Manuel Hermenegildo)
  - Now commas etc. are allowed in section names (so that they can be used in other formats). They are eliminated automatically in texi and info. This avoids wrong section names --and thus dangling pointers-- in generated texinfo files. (Manuel Hermenegildo)
  - Eliminated superfluous copy of summary in info mode. (Manuel Hermenegildo)
  - Eliminated unsupported chars that broke texi manual cross-referencing. (Manuel Hermenegildo)
  - Improved treatment of accents (dotless i and dotless j, o, etc.). (Manuel Hermenegildo)
  - Initial size passed to xdvi more appropriate for current xdvis. (Manuel Hermenegildo)
  - Accents in bibliography fixed. (Manuel Hermenegildo)
  - Now repeated sections are disambiguated. (Manuel Hermenegildo)
  - Eliminated unnecessary escaping (especially for &). (Manuel Hermenegildo)
  - Better detection of when version is not available. (Manuel Hermenegildo)
  - Added new doc(address, _) comment, which is the right place to put address/contact information in manuals (Jose Morales)
• Added new \texttt{\texttt{\texttt{\texttt{version}}}()} command (expands to the version of the software to be documented). (Jose Morales)

• Shorter \texttt{SETTINGS.pl} files (with some rudimentary, assertion-based checking of options) (Jose Morales)

• Bug fix: \texttt{\texttt{\texttt{\texttt{\texttt{include}}} and \texttt{\texttt{\texttt{\texttt{includeverbatim}}} are no longer a problem}} (space can be omitted) (Jose Morales)

• Added and documented a new \texttt{documentation} filetype (for some parts of the manual that contains only documentation). That avoids the old trick of declaring a fake \texttt{main/0} predicate. (Jose Morales)

• Style for subtitle added automatically (in texinfo, it is \textit{emph; in HTML it is normal text with smaller font). The entries in \texttt{subtitle_extra} are free-form. (Jose Morales)

• Bugs and changelog appear now in the global links in the HTML backend. (Jose Morales)

• Merged code that documented \texttt{.pl} and \texttt{.lpdoc} files. (Jose Morales)

• No copyright section if no copyright comment. (Jose Morales)

• Auxiliary documentation files ending in \texttt{'_doc'} displayed incorrect names for the module (ending in \texttt{'_doc'}). E.g., \texttt{use_package(foo_doc)} was displayed instead of \texttt{use_package(foo_doc)}. Fixed. (Jose Morales)

• In \texttt{verbatim} environments, new-line characters are removed from the beginning. (Jose Morales)

• Fix wrong use of \texttt{erase/1} for clauses (which resulted in segmentation fault when documentation generation failed) (Jose Morales)

• Fixed image generation (now uses \texttt{.png} files for HTML) (Jose Morales)

• New code for text escape fixed some problems, like \texttt{'0/1'} operator not being displayed correctly in Info. (Jose Morales)

• Colors for Prolog variables (in HTML). (Jose Morales)

• Added \texttt{\texttt{\texttt{\texttt{\texttt{\texttt{begin}}}alert}} environment for alert messages (like cartouche, but in red). (Jose Morales)

• Supporting \texttt{'\texttt{\texttt{\texttt{\texttt{\texttt{of}}}}}'} command for umlaut, in addition to \texttt{'\texttt{\texttt{\texttt{\texttt{\texttt{of}}}..}}'} (Jose Morales)

• Double quotes correctly translated to HTML (Jose Morales)

• \texttt{\texttt{\texttt{\texttt{\texttt{\texttt{author}}} command to reference authors (changed command referring to people by \texttt{\texttt{\texttt{\texttt{\texttt{author}}}}, in all the documentation) (Jose Morales)

• Simplification of documentation setting files (see the documentation for further details) (Jose Morales)

• Using \texttt{open} for \texttt{lpdoc htmlview} command in MacOS X (Jose Morales)

• Adding \texttt{html} and \texttt{pdf} formats as options for emacs customization of LPdoc (\texttt{html} is the default one now) (Jose Morales)

• Improved detection of external tools for image conversion. (Manuel Hermenegildo)

• Added section name syntax auto-correction. This avoids wrong section names—and thus dangling pointers—in generated texinfo files. (Manuel Hermenegildo)

• Document size more appropriate for current xdvi versions. (Manuel Hermenegildo)

• Lpdoc no longer adds .info filename suffix to .infoindex entries since it breaks Debian’s install-info –remove and goes against standard practice anyway. (Jose Luis Gonzalez)
• Added option \texttt{-cv, \texttt{-comment-version}}, that tells \texttt{lpdoc} if the file has version comment. Formatting of \texttt{lpdoc} version comments completed. (Edison Mera)

• Improved handling of option values. Added \texttt{-d} option to \texttt{lpdoc}, that allows defining additional values in the argument. Added options \texttt{-l} and \texttt{-m} that are similar to the corresponding \texttt{lpmake} options. (Edison Mera)

• Support for in-code sections (experimental):
  • Latex-like font-lock highlight of sectioning documentation comments (\texttt{:-doc(C, "...")}, with \texttt{C} one of \texttt{title, section, and subsection}). Currently the \texttt{section} and \texttt{subsection} comments are still ignored by \texttt{LPdoc}. (Jose Morales)

• Support for mathematical notation (experimental):
  • new \texttt{@math{...}} and \texttt{@begin{displaymath}...@end{displaymath}} environments are supported (see the documentation for more details) (Jose Morales)
  • In documentation strings, single \ must be escaped (e.g. \texttt{@math{\lambda}}) (Jose Morales)
  • Supported in both the texinfo and HTML (using MathJax) backends. (Jose Morales)
  • Added \texttt{@defmathcmd{Cmd}{N}{Def}} and \texttt{@defmathcmd{Cmd}{Def}}, both for texinfo and HTML backends. Those \texttt{LPdoc} commands define new mathematical environments (equivalent to \texttt{
ewcommand}). (Jose Morales)


Last version before moving to subversion. 1.9 and 2.0 were merged. 1.9 (based on makefiles) is deprecated.

• New functionality:
  • Use of \texttt{:-doc} declarations (as a shorthand for \texttt{comment}) now allowed. (Manuel Hermenegildo)
  • Made \texttt{xdvi} viewer, ps viewer, and \texttt{xdvi zoom size} be parameters (the latter since new versions of \texttt{xdvi} display sizes differently than old ones). (Manuel Hermenegildo)
  • Processing options can now be set for each file independently. (Manuel Hermenegildo)
  • Proper pdf generation now achieved in most cases, thanks to newer versions of \texttt{dvips}. (Manuel Hermenegildo)
  • Added option \texttt{-c Target} in \texttt{lpdoc}, that treats Target as a separate component. (Edison Mera)
  • Added option \texttt{-f ConfigFile in lpdoc}, that uses the file \texttt{ConfigFile} instead the default \texttt{LPSETTINGS.pl}. (Edison Mera)
  • Added option ascii that generates documentation in ascii plain format. (Edison Mera)
  • Added \texttt{-help} option. Is equal to \texttt{-h}. (Edison Mera)
  • Added option testsettings to check that the settings file is correctly specified. (Edison Mera)
  • Changed \texttt{generate_html_pointer/5} by \texttt{generate_html_pointer/6} to let it work with any given directory, and not only the working directory. (Edison Mera)
Version 2.0 (1999/8/17, 17:28:52 CEST)
Major change to eliminate need for Makefiles: lpdoc is now a standalone command (Manuel Hermenegildo). Proceeds in parallel with further development of 1.9. Merge pending. Previous changes incorporated since 1.8:
- New functionality:
  - A new parameter \texttt{PAPERTYPE} can be set in the \texttt{SETTINGS} file which controls the format of printed output. (Manuel Hermenegildo)
  - Default pdf viewer is now \texttt{ghostview}, since recent versions handle pdf well. (Manuel Hermenegildo)
  - Changed default style sheet in order to show \texttt{<PRE>} lines with a monospaced font. (Daniel Cabeza Gras)
  - Mode definitions now documented in a separate section. The way they are documented has been improved. (Manuel Hermenegildo)
  - References in files now updated only if \texttt{.refs} file is not empty. (Manuel Hermenegildo)
  - A copy of the html style sheet is now included in \texttt{distributions}. Also \texttt{Copies} of the html and info index head and tail files. (Manuel Hermenegildo)
  - Made pointers relative in library html templates. (Manuel Hermenegildo)
- Bug fixes and other minor improvements:
  - Declarations now documented properly even if they have the same name and arity as a predicate. (Manuel Hermenegildo)
  - Accented i’s now translate correctly in html. (Manuel Hermenegildo)
  - Fixed a funny installation quirk: while we want to install LPdoc in the Ciao group, the manuals produced by LPdoc should be installed in the LPdoc group. (Manuel Hermenegildo)
  - Now using \texttt{lpdoclib} path alias. (Manuel Hermenegildo)
  - Fixed bug in ordering of html indices in recent Linux versions, related to varying file listing order depending on locale. (Manuel Hermenegildo)

Version 1.9 (1999/7/8, 18:19:43 MEST)
In this release the name of the application has changed to \texttt{lpdoc}.
- New commands:
  - \texttt{@begin{cartouche}} and \texttt{@end{cartouche}} commands now supported.
  - \texttt{@foonote} command now supported.
  - New \texttt{gmake htmlview} command (makes a running \texttt{netscape} visit the generated html manual). Suggested by Per Cederberg.
  - New \texttt{gmake distclean} command, intended for software distributions. Leaves the generated documents and eliminates all intermediate files (including \texttt{.texic/.texi} files).
  - Adobe \texttt{pdf} format now supported as a valid target. Unfortunately, embedded \texttt{.eps} figures are not supported at this time in pdf output.
  - The second argument of \texttt{:- comment(hide,...)}. and \texttt{:- comment(doinclude,...)}. declarations can now be a list of predicate names.
  - A \texttt{-u File} option is now supported so that a file including, e.g., path alias definitions can be included (this has the same functionality as the \texttt{-u option in ciaoc}).
  - Now typing just \texttt{gmake} does nothing. In order to do something at least one target should be specified. This was necessary so that recursive invocations with empty arguments did nothing.
• Added a new filetype: part. This allows splitting large documents into parts, each of which groups a series of chapters.

• Other new functionality:
  • A style sheet can now be specified which allows modifying many characteristics of the html output (fonts, colors, background, ...) (thanks to Per Cederberg).
  • Added limited support for changing page numbering (in SETTINGS file).
  • The concept indexing commands (@index, @cindex, and @concept) now work somewhat differently, to make them consistent with other indexing commands.
  • The old usage index is now called, more appropriately, global index. Correspondingly, changed things so that now every definition goes to the global index in addition to its definitional index.
  • Imported files from module user are now documented separately.
  • Now a warning is issued if characters unsupported by info are used in section names.
  • Navigation in html docs was improved.
  • The table of contents in printed manuals now contains entries for the individual descriptions of predicates, props, regtypes, declarations, etc. This can be shut off with the -shorttoc option.
  • Made more silent in normal conditions: file inclusion is muted now unless -v option is selected.
  • A single .texi file is now constructed (by grouping the .texic files generated for all components) in which the references and menus are resolved. This has the advantage that the process of resolving references and menus has now been sped up very significantly. Also, texi is now a valid target (perhaps useful for distributions). The generated files now have texic (texinfo component).
  • Now, declarations are always documented as long as there is a decl assertion. Also, they are now documented in a separate section.

• Bug fixes and other minor improvements:
  • The directory containing html manual is now called BASENAME_html instead of just BASENAME, which was confusing.
  • Now requesting building a .ps only does not leave a .dvi behind (useful for distributions).
  • File names can now include the symbol _ even if they contain figures.
  • TeX-related intermediate files are now cleaned up after each run in order to avoid clutter.
  • Fixed -modes, which was broken since going to the new normalizer (was normalizer problem). Fixed problem with no documentation when only modes given.
  • Fixed duplication of documentation for internal predicates when also exported.
  • Minor formatting problem when no documentation nor definition found for a regtype fixed.
  • Determining exports, imports, etc. now done solely by calls to c_itf library (and, thus, synchronized with ciaoc compiler).

(Manuel Hermenegildo)

This version completes the port to using the ciao 0.8 modular assertion processing library. In addition, it includes the following improvements:

- Now, if the name of a file being documented ends in \_doc, the \_doc part is left out when referring to the file in the documentation (useful if one would like to place the documentation declarations in different file).
- It is now possible to declare (via a comment/2 declaration) the intended use of a file which is not a module (i.e. a package, user, or include file), which results in correct documentation of operator definitions, new declarations, etc. The declaration is only needed for 'user' files (i.e., files to be loaded with \texttt{ensure\_loaded/1}).
- Separated generation of the manuals from their installation. I.e., \texttt{gmake install} now does not force a \texttt{gmake all}, which has to be done by hand. This was necessary to ensure correct installation of distributed manuals, even if modification dates are changed during installation. Previously, in some cases generation was triggered unnecessarily.
- New \texttt{-v} option allows using quieter by default operation when not debugging.
- New option \texttt{-propmods} makes the name of the module in which a property is defined appear in front of the property in the places where the property is used.
- New option \texttt{-noisoline} makes the textual explanation of the \texttt{iso/1} property not appear in the description of the usage (but the \texttt{\textbf{ISO}\bullet} symbol does appear)
- Two new options, \texttt{-nosysmods} and \texttt{-noengmods}, selectively avoid listing the system or engine libraries used.
- If there is no declaration for a predicate, now a line is output with the name and arity and a simple comment saying that there is no further documentation available (this has the great advantage that then it goes in the index, and, for example in ciao, they get added to completion commands!).
- Now, if a property or regtype declaration has no textual comment, the actual definition is given (first level only) in the place where it is documented, and a simple generic message where it is used.
- Added \texttt{@noindent} and \texttt{@iso} commands.
- Nicer spacing now when printing predicate names which are operators, as well as modes, etc.
- Reporting of versions in libraries has been improved: now both the global version and the last version in which the library itself was changed are reported.
- Exported new declarations also documented now for include-type files.
- A module is now documented even if exports nothing at all.
- Engine modules used now documented even if no other modules used (was a reported bug).
- Fixed indexing of names containing \texttt{@} etc. for newer versions of texinfo.
- Tabs in verbatim modes now converted to a number of spaces (8). Not perfect, but produces better output than leaving the tabs in.
- Tex is now run in 'nonstopmode' which means it will typically not stop if there are minor errors (but some errors may go unnoticed...).
- The full path of the version maintenance directory is now computed (correctly) using the directory of the \texttt{.pl} file being documented as base.
- Notices for missing subtitle, copyright, and summary now only given from main file and not for components.
• Added special handling of regtype and generalized it to handle some props specially if there is a certain comp property present.

(Manuel Hermenegildo)

Version 1.7 (1998/12/2, 17:43:50 MET)
Major port to use the ciao 0.8 modular assertion processing library. (Manuel Hermenegildo)

Version 1.6 (1998/9/8, 12:49:26 MET)
Added support for inserting images (.eps files) in text via ©image command, email addresses via ©email command, and url references via ©uref command.

Unix 'man' output much improved. Also, it now includes a usage section. The correspoinding text must be given in a string contained in the first argument of a fact of the usage_message/1 predicate which appears in the program. Also, formatting of 'man' pages has been greatly improved.

A new 'asciid' format is now supported: a simple minded ascii manual (basically, an info file without pointers).

(Manuel Hermenegildo)

Now supporting a ©cite command (YES!). It automatically accesses the bib entries in .bib files (using bibtex) and produces a 'References' appendix. ©cite can be used in the text strings exactly as ite in LaTeX. The set of bib files to be used is given in the SETTINGS file.

Defining the type of version maintenance that should be performed by the emacs ciao.el mode (i.e., whether version numbers are in a given directory or in the file itself) is controlled now via a standard comment/2 declaration. You should now write a declaration such as:

:- comment(version_maintenance,dir('..//version')).

to state that control info is kept in directory ..//version. This has the advantage that it is shorter than the previous solution and that Lpdoc can read this info easily. Using this guarantees that the version numbers of the manuals always coincide with those of the software.

Generation of indices of manuals (.htmlbullet files): if several manuals are installed in the same directory, an index to them is now generated at the beginning of the html cover page describing the directory.

(Manuel Hermenegildo)

The set of paths defined in SETTINGS for finding the source files are now also used to find 'included' files. As a result, full path is not needed any more in, e.g., ©include command.

New ©ref command which can be used to refer to chapter, sections, subsections, etc..

Support for recent minor changes in assertion format, including '#!' as comment separator.

Used modules are now separated in documentation (in the interface description) by type (user, system, engine...).

Supports new 'hide' option in comments, to prevent an exported predicate from being documented. This is useful for example for avoiding mentioning in the documentation multiple predicates which are not intended to be modified by the user.

(Manuel Hermenegildo)
Version 1.3 (1998/7/10, 16:35:2 MET DST)
Exports are now listed in the chapter header separated by kind (pred, types, properties, ...).

The list of other modules used by a module is now separated in the chapter header into User and System modules (controlled by two sets of paths in SETTINGS).

New hide option of comment/2 decl prevents an exported predicate from being included in the documentation: :- comment(hide,p/3).
(Manuel Hermenegildo)

Major overall improvements... (Manuel Hermenegildo)

Incorporated autodoc and autodoformats library to source in order to make distribution standalone. Improvements to installation and documentation. Makefiles now also install documentation in public areas and produce global indices. Several documents can coexist in the same installation directory. (Manuel Hermenegildo)

Version 1.0 (1998/2/24)
First Ciao-native distribution, with installation. (Manuel Hermenegildo)

Version 0.9 (1998/2/24)
Intermediate version, preparing for first major release. Modified Makefile and SETTINGS to handle installation of manuals. (Manuel Hermenegildo)

Version 0.6 (1998/2/10)
Added new indices and options, as well as more orthogonal handling of files. (Manuel Hermenegildo)

Version 0.4 (1998/2/24)
Added support for nroff -m formatting (e.g., for man pages). Added support for optional selection of indices to be generated. Added support for reexported predicates. Added (low level) ascii format. Added option handling (-nobugs -noauthors -noversion -nochangelog -nopatches -modes and -headprops ...). -literalprops. Fixed presentation when there are multiple kinds of assertions. Better error checking for includedefact/includedef. (Manuel Hermenegildo)

Version 0.3 (1998/2/10)
Changed file reader to use Ciao native builtins. As a result, syntax files and full Ciao syntax now supported. Major reorganization of the code to make formatting more orthogonal. Now applications and libraries can be components or main files, standalone or with components interchangeably. @includefact, new predicate types, used libraries now precisely detected, docinclude option. (Manuel Hermenegildo)

Version 0.2 (1997/12/16)
Ported to native ciao. Version handling, selection of indices, @include. Added generation of an html brief description for a global index. Added unix manual page generation. Added support for specifying library paths. -l option for htmlindex and man. Installation improved: now all files for one application in the same directory. (Manuel Hermenegildo)

Version 0.1 (1997/7/30)
First official version (major rewrite from several previous prototypes, autodocumented!). (Manuel Hermenegildo)

Version 0.0 (1996/10/10)
First prototype.
Chapter 2: Generating Installing and Accessing Manuals

Author(s): Manuel Hermenegildo.

Note: significant parts of this are obsolete. They must be updated to describe lpdfoc version 2.0.

This section describes how to generate a manual (semi-)automatically from a set of source files using lpdfoc, how to install it in a public area, and how to access it on line. It also includes some recommendations for improving the layout of manuals, usage tips, and troubleshooting advice.

2.1 Generating a manual from the Ciao Emacs mode

If you use the Emacs editor (highly recommended in all circumstances), then the simplest way to quickly generate a manual is by doing it from the Ciao Emacs mode (this mode comes with the Ciao Prolog distribution and is automatically installed with Ciao). The Ciao Emacs mode provides menu- and keyboard-binding driven facilities for generating a stand-alone document with the documentation corresponding to the file in the buffer being visited by Emacs. This is specially useful while modifying the source of a file, in order to check the output that will be produced when incorporating this file into a larger document. It is also possible to generate more complex documents, by editing the (automatically provided) SETTINGS.pl in the same way as when generating a manual from the command line (see below). However, when generating complex documents, it is best to devote an independent, permanent directory to the manual, and the full procedure described in the rest of this text is preferred.

2.2 Generating a manual

Two possible scenarios are described in this section. The first one is indicated to document quickly a single module and the second one targets the documentation of a larger application or library, in which the settings (which define how the documentation is to be generated, etc.) are read from a file, so that they can be reused as the application / library evolves.

In order to make lpdfoc generate quickly the documentation of a single file it suffices to execute the command lpdfoc -d doc_structure=modulename dvi, where modulename is the module to be documented (without extension) and (in this example) dvi is the desired format of the manual (other accepted formats include html, pdf, ps, etc. – see later). lpdfoc will generate a manual with the name of the module and the format extension (in the example it would be modulename.dvi) in the same directory where it is executed.

For the second scenario, the lpdfoc library directory includes a generic file which is quite useful for the generation of complete manuals: the SETTINGS.pl file. Use of this file is strongly recommended. Generating a manual using this file involves the following steps:

• Create a directory (e.g., doc) in which the documentation will be built. The creation of this directory is recommended, as it will be populated with intermediate files which are best kept separate. This directory is typically created in the top directory of the distribution of the application or library to be documented.

• Execute the command lpdfoc lpdfoc lpsettings in the directory where the documentation is to be created (e.g., doc in the previous point). lpdfoc will create an SETTINGS.pl.generated file with the default settings. This file should be renamed to SETTINGS.pl once the user agrees with its contents.

• Edit SETTINGS.pl to suit your needs. It is recommended that you review, at least, the following points:
• Set the variable `filepath` to include all the directories where the files to be documented can be found.

• Set the variable `systempath` to include all the `system` directories where system files used can be found, regardless whether they are to be documented or not. This will be used to access definitions of types, etc.

It is very important to include `all` related directories either in `filepath` or in `systempath` because on startup `lpdoc` has `no default search paths for files` defined (not even those typically defined by default in the `Prolog` system under which it was compiled! – this allows documenting `Prolog` systems other than that under which `lpdoc` was compiled).

The effect of putting a path in `systempaths` instead of in `filepaths` is that the modules and files in those paths are documented as `system modules` (this is useful when documenting an application to distinguish its parts from those which are in the system libraries).

• Set `doc_structure` to be the `document structure (doc_structure/1)`.

For the rest of the settings in the `SETTINGS.pl` file you can simply use the default values indicated. You may however want to change several of these:

• `doc_mainopts` can be set to a series of options which allow more detailed control of what is included in the documentation for the main file and how (i.e., including bug information, versions and patches or only patches, authors, changelog, explanation of modes, one-sided printing (two-sided is the default), etc.). See `option_comment/2` in `autodoc` or type `lpdoc -help` for a list of these options.

• In the same way `doc_compopts` sets options for the component files. Currently these options are common to all component files but they can be different from `doc_mainopts`. The allowable options are the same as above.

• `docformat` determines the set of formats (dvi, ps, ascii, html, info, man1, ...) in which the documentation should be generated when typing `lpdoc all`. Selecting `htmlindex` and/or `infoindex` requests the generation of (parts of) a master index to be placed in an installation directory and which provide pointers to the documents generated (see below). If the main file is an `application`, and the `man1` option is selected, then `lpdoc` looks for a `usage_message/1` fact, which should contain a string as argument, and will use that string to document the `usage of the application` (i.e., it will be used to fill in the `synopsis section of the man page`).

• `output_name` determines the base file name of the main documents generated by `lpdoc`. By default it is equal to the main file name, or, if the main file name ends with `_doc`, then it is equal to the name without the `_doc` suffix. This is useful when the name of the documentation file to be produced needs to have a name that is not directly related to the main file being documented.

• `index` determines the list of indices to be included at the end of the document. These can include indices for defined predicates, modules, concepts, etc. For a complete list of the types of indices available see `index_comment/2` in `autodoc` or type `lpdoc -help` for a listing. A setting of `all` generates all the supported indices – but beware of limitations in the number of simultaneous indices supported in many `texinfo` installations.

• `bibfile` determines a list of `.bib files` (one file per path), i.e., files containing `bibliographic entries` in `bibtex` format. This is only relevant if you are using citations in the text (using the `@cite command`). In that case those will be the files in which the citations will be searched for. All the references will appear together in a `References appendix` at the end of the manual.

If you are not using citations, then select the `nobiblio` option on the main file, which will prevent an empty `References` appendix from appearing in the manual.
• **startpage** (default value 1) allows changing the page number of the first page of the manual. This can be useful if the manual is to be included in a larger document or set of manuals. Typically, this should be an *odd* number.

• **papertype** (default value `afourpaper`) allows selecting several paper sizes for the printable outputs (dvi, ps, etc.). The currently supported outputs (most of them inherited from `texinfo`) are:

  - `afourpaper`
    The default, usable for printing on A4 *paper*. Rather busy, but saves trees.
  
  - `afourwide`
    This one crams even more stuff than `afourpaper` on an A4 page. Useful for generating manuals in the least amount of space. It saves more trees.
  
  - `afourlatex`
    This one is a little less compressed than `afourpaper`.
  
  - `smallbook`
    Small pages, like in a handbook.
  
  - `letterpaper`
    For printing on American *letter size paper*.
  
  - `afourthesis`
    A *thesis-like style* (i.e., double spaced, wide margins etc.). Useful – for inserting `lpdoc` output as appendices of a thesis or similar document. It does not save trees.

• Type `lpdoc all` to generate all the formats defined. `lpdoc dvi`, `lpdoc html`, `lpdoc ps` or `lpdoc info`, etc. will force the generation of a single target format.

### 2.3 Working on a manual

In order to speed up processing while developing a manual, it is recommended to work by first generating a `.dvi` version only (i.e., by typing `lpdoc dvi`). The resulting output can be easily viewed by tools such as `xdvi` (which can be started by simply typing `lpdoc view`). Note that once an `xdvi` window is started, it is not necessary to restart it every time the document is reformatted (`lpdoc dvi`), since `xdvi` automatically updates its view every time the `.dvi` file changes. This can also be forced by typing `R` in the `xdvi` window. The other formats can be generated later, once the `.dvi` version has the desired contents.

### 2.4 Cleaning up the documentation directory

`lpdoc` can also take care of tidying up the directory where the documentation is being generated:

• `lpdoc clean` deletes all intermediate files, but leaves the targets (i.e., the `.ps`, `.dvi`, `.ascii`, `.html`, etc. files), as well as all the generated `.texic` files.

• `lpdoc distclean` deletes all intermediate files and the generated `.texic` files, leaving only the targets (i.e., the `.ps`, `.dvi`, `.ascii`, `.html`, etc. files). This is the option normally used when building software distributions in which the manuals come ready made in the distribution itself and will not need to be generated during installation.

• `lpdoc docscclean` deletes all intermediate files and the generated targets, but leaves the `.texic` files. This option can be used in software distributions in which the manuals in the different formats will be generated during installation. This is generally more compact, but requires the presence of several tools, such as `tex`, `Emacs`, etc. (see Section 14.2 [Other software packages required (lpdoc)], page 107), in order to generate the manuals in the target formats during installation.
• **lpdoc realclean** performs a complete cleanup, deleting also the .texic files, i.e., it typically leaves only the SETTINGS.pl file. This is the most compact, but requires the presence of the tools mentioned above, the source files from which the manuals are generated and lpdoc in order to re generate the manuals in the target formats during installation.

### 2.5 Installing a generated manual in a public area

Note: This part is obsolete. It must be updated to describe lpdoc version 2.0. – EMM

Once the manual has been generated in the desired formats, the **Makefile** provided also allows automatic installation in a different area, specified by the docdir option in the SETTINGS.pl file. This is done by typing `lpdoc install`.

As mentioned above, lpdoc can generate directly brief descriptions in html or Emacs info formats suitable for inclusion in an on-line index of applications. In particular, if the `htmlindex` and/or `infoindex` options are selected, `lpdoc install` will create the installation directory, place the documentation in the desired formats in this directory, and produce and place in the same directory suitable `index.html` and/or `dir` files. These files will contain some basic info on the manual (extracted from the summary and title, respectively) and include pointers to the relevant documents which have been installed. The `infodirheadfile` / `infodirtailfile` (default examples, used in the CLIP group at UPM, are included with the distribution) should point to files which will be used as head and tail templates when generating the `dir` files. Several manuals, coming from different doc directories, can be installed in the same docdir directory. In this case, the descriptions of and pointers to the different manuals will be automatically combined (appearing in alphabetic order) in the `index.html` and/or `dir` indices, and a contents area will appear at the beginning of the html index page. Important Note: In order for the different components to appear in the correct positions in the index pages mentioned above the traditional ('C') Lexical order must be active. In recent Un*x systems (e.g., in most current Linux systems) this may not be the case. There are several possible fixes:

- For csh put `setenv LC_COLLATE C` in your .cshrc.
- For bash put `export LC_COLLATE=C` in your .profile.
- In many systems this can be done globally by the super-user. E.g., in many Linux systems set `LANG="C"` in `/etc/sysconfig/i18n`.

Note that, depending on the structure of the manuals being generated, some formats are not very suitable for public installation. For example, the .dvi format has the disadvantage that it is not self contained if images are included in the manual. Typing `lpdoc uninstall` in a doc directory will uninstall from docdir the manuals corresponding to the **Makefile** in that doc directory. If a manual is already installed and changes in the number of formats being installed are desired, `lpdoc uninstall` should be made before changing the docformats variable and doing `lpdoc install` again. This is needed in order to ensure that a complete cleanup is performed.

### 2.6 Enhancing the documentation being generated

The quality of the documentation generated can be greatly enhanced by including within the program text:

- **assertions**, and
- **machine-readable comments**.

**Assertions** are declarations which are included in the source program and provide the compiler with information regarding characteristics of the program. Typical assertions include type
declarations, modes, general properties (such as \textit{does not fail}), standard compiler directives (such as \texttt{dynamic/1}, \texttt{op/3}, \texttt{meta_predicate/1...}), etc. When documenting a module, \texttt{lpdfdoc} will use the assertions associated with the module interface to construct a textual description of this interface. In principle, only the exported predicates are documented, although any predicate can be included in the documentation by explicitly requesting it (see the documentation for the \texttt{doc/2} declaration). Judicious use of these assertions allows at the same time documenting the program code, documenting the external use of the module, and greatly improving the debugging process. The latter is possible because the assertions provide the compiler with information on the intended meaning or behaviour of the program (i.e., the specification) which can be checked at compile-time (by a suitable preprocessor/static analyzer) and/or at run-time (via checks inserted by a preprocessor).

\textbf{Machine-readable comments} are also declarations included in the source program but which contain additional information intended to be read by humans (i.e., this is an instantiation of the \textit{literate programming} style of Knuth [Knu84]). Typical comments include title, author(s), bugs, changelog, etc. Judicious use of these comments allows enhancing at the same time the documentation of the program text and the manuals generated from it.

\texttt{lpdfdoc} requires these assertions and comments to be written using the \texttt{Ciao} system \textit{assertion language}. A simple compatibility library is available in order to make it possible to compile programs documented using assertions and comments in traditional (constraint) logic programming systems which lack native support for them (see the \texttt{compatibility} directory in the \texttt{lpdfdoc} library). Using this library, such assertions and comments are simply ignored by the compiler. This compatibility library also allows compiling \texttt{lpdfdoc} itself under (C)LP systems other than the \texttt{Ciao} system under which it is developed.

\section*{2.7 Accessing on-line manuals}

As mentioned previously, it is possible to generate on-line manuals automatically from the \texttt{.texic} files, essentially \texttt{.html}, \texttt{.info}, and \texttt{man} files. This is done by simply including the corresponding options in the list of \texttt{docformats} in the \texttt{SETTINGS.pl} file and typing \texttt{lpdfdoc all}. We now address the issue of how the different manuals can be read on-line.

\subsection*{2.7.1 Accessing html manuals}

Once generated, the \texttt{.html} files can be viewed using any standard \texttt{WWW} browser, e.g., Firefox (a command \texttt{lpdfdoc htmlview} is available which, if there is an instance of a web browser running in the machine, will make that instance visit the manual in \texttt{html} format). To make these files publicly readable on the \texttt{WWW}, they should be copied into a directory visible by browsers running in other machines, such as \texttt{/home/clip/public_html/lpdfdoc_docs}, \texttt{/usr/home/httpd/htmldocs/lpdfdoc_docs}, etc. As mentioned before, this is easily done by setting the \texttt{docdir} variable in the \texttt{SETTINGS.pl} file to this directory and typing \texttt{lpdfdoc install}.

\subsection*{2.7.2 Accessing info manuals}

Generated \texttt{.info} files are meant to be viewed by the \texttt{Emacs} editor or by the standalone \texttt{info} application, both publicly available from the GNU project sites. To view the a generated \texttt{info} file from \texttt{Emacs} manually (i.e., before it is installed in a common area), type \texttt{C-u M-x info}. This will prompt for an info file name. Input the name of the info file generated by \texttt{lpdfdoc} (main.info) and \texttt{Emacs} will open the manual in info mode.

There are several possibilities in order to install an \texttt{.info} file so that it is publicly available, i.e., so that it appears automatically with all other \texttt{info} manuals when starting \texttt{info} or typing \texttt{C-u M-x info} in \texttt{Emacs}:

- \textbf{Installation in the common info directory:}
• Move the .info file to the common info directory (typically /usr/info, /usr/local/info, ...). This can be done automatically by setting the docdir variable in the SETTINGS.pl file to this directory and typing lpdoc install.

  **Warning:** if you are installing in an info directory that is not maintained automatically by lpdoc, make sure that you have not selected the infoidx option in docformats, since this will overwrite the existing dir file.

• Add an entry to the info index in that directory (normally a file in that directory called dir). The manual should appear as part of the normal set of manuals available when typing M-x info in Emacs or info in a shell. See the Emacs manual for details.

• **Installation in a different info directory:** you may want to place one or more manuals generated by lpdoc in their own directory. This has the advantage that lpdoc will maintain automatically an index for all the lpdoc generated manuals installed in that directory. In order for such manuals to appear when typing M-x info in Emacs or info in a shell there are two requirements:

  • This directory must contain a dir index. The first part of the process can all be done automatically by setting the docdir variable in the SETTINGS.pl file to this directory, including the infoidx option in docformats, and typing lpdoc install. This will install the info manual in directory docdir and update the dir file there. **lpdoc uninstall** does the opposite, eliminating also the manual from the index.

  • The directory must be added to the *info path list*. The easiest way to do this is to set the INFOPATH environment variable. For example, assuming that we are installing the info manual in /home/clip/public_html/lpdoc_docs and that /usr/info is the common info directory, for csh in .cshrc:

    ```
    setenv INFOPATH /usr/info:/home/clip/public_html/lpdoc_docs
    ```

    Adding the directory to the info path list can also be done within Emacs, by including the following line in the .Emacs file:

    ```
    (defun add-info-path (newpath)
      (setq Info-default-directory-list
            (cons (expand-file-name newpath) Info-default-directory-list))
      (add-info-path "/home/clip/public_html/lpdoc_docs")
      (add-info-path "/usr/info")
    )
    ```

    However, this has the disadvantage that it will not be seen by the standalone info command.

Automatic, direct on-line access to the information contained in the info file (e.g., going automatically to predicate descriptions by clicking on predicate names in programs in an Emacs buffer) can be easily implemented via existing .el packages such as word-help, written by Jens T. Berger Thielemann (jensthi@ifi.uio.no). word-help may already be in your Emacs distribution, but for convenience the file word-help.el and a word-help-setup.el file, providing suitable initialization are included in the lpdoc library. A suitable interface for word-help is also provided by the ciao.el Emacs file that comes with the Ciao system distribution (i.e., if ciao.el is loaded it is not necessary to load or initialize word-help).

### 2.7.3 Accessing man manuals

The Unix man format manuals generated by lpdoc can be viewed using the Unix man command. In order for man to be able to locate the manuals, they should be copied to one of the subdirectories (e.g., /usr/local/man/man1) of one of the main man directories (in the previous case the main directory would be /usr/local/man). As usual, any directory can be used as a man main directory, provided it is included in the environment variable MANPATH. Again, this process can be performed automatically by setting the docdir variable in the SETTINGS.pl file to this directory and typing lpdoc install.
2.7.4 Putting it all together

A simple, powerful, and very convenient way to use the facilities provided by \texttt{lpdoc} for automatic installation of manuals in different formats is to install all manuals in all formats in the same directory \texttt{docdir}, and to choose a directory which is also accessible via \texttt{WWW}. After setting \texttt{docdir} to this directory in the \texttt{SETTINGS.pl} file, and selecting \texttt{infoindex} and \texttt{htmlindex} for the \texttt{docformats} variable, \texttt{lpdoc install/lpdoc uninstall} will install/uninstall all manuals in all the selected formats in this directory and create and maintain the corresponding \texttt{html} and \texttt{info} indices. Then, setting the environment variables as follows (e.g., for \texttt{csh} in \texttt{.cshrc}):

\begin{verbatim}
setenv DOCDIR /home/clip/public_html/lpdoc_docs
setenv INFOPATH /usr/local/info:${DOCDIR}:
setenv MANPATH ${DOCDIR}}:${MANPATH}:
\end{verbatim}

Example files for inclusion in user’s or common shell initialization files are included in the \texttt{lpdoc} library.

More complex setups can be accommodated, as, for example, installing different types of manuals in different directories. However, this currently requires changing the \texttt{docformats} and \texttt{docdir} variables and performing \texttt{lpdoc install} for each installation format/directory.

2.8 Some usage tips

This section contains additional suggestions on the use of \texttt{lpdoc}.

2.8.1 Ensuring Compatibility with All Supported Target Formats

One of the nice things about \texttt{lpdoc} is that it allows generating manuals in several formats which are quite different in nature. Because these formats each have widely different requirements it is sometimes a little tricky to get things to work successfully for all formats. The following recommendations are intended to help in achieving useful manuals in all formats:

- The best results are obtained when documenting code organized as a series of libraries, and with a well-designed module structure.

- \texttt{texinfo} supports only a limited number of indices. Thus, if you select too many indices in the \texttt{SETTINGS.pl} file you may exceed \texttt{texinfo}'s capacity (which it will signal by saying something like “No room for a new \@write”).

- The GNU info format requires all nodes (chapters, sections, etc.) to have different names. This is ensured by \texttt{lpdoc} for the automatically generated sections (by appending the module or file name to all section headings). However, care must be taken when writing section names manually to make them different. For example, use “lpdoc usage” instead of simply “Usage”, which is much more likely to be used as a section name in another file being documented.

- Also due to a limitation of the \texttt{info} format, do not use : or , or -- in section, chapter, etc. headings.

- The character “\_” in names may sometimes give problems in indices, since current versions of \texttt{texinfo} do not always handle it correctly.

2.8.2 Writing comments which document version/patch changes

When writing version comments (:- \texttt{doc(version(...), "..."))}, it is useful to keep in mind that the text can often be used to include in the manual a list of improvements made to the software since the last time that it was distributed. For this to work well, the textual comments should describe the significance of the work done for the user. For example, it is more useful to write "added support for \texttt{pred} assertions" than "modifying file so \texttt{pred} case is also handled".
Sometimes one would like to write version comments which are internal, i.e., not meant to appear in the manual. This can easily be done with standard Prolog comments (which \texttt{lpdoc} will not read). An alternative and quite useful solution is to put such internal comments in \texttt{patch} changes (e.g., 1.1\#2 to 1.1\#3), and put the more general comments, which describe major changes to the user and should appear in the manual, in \texttt{version} changes (e.g., 1.1\#2 to 1.2\#0). Selecting the appropriate options in \texttt{lpdoc} then allows including in the manual the version changes but not the patch changes (which might on the other hand be included in an \texttt{internals manual}).

\subsection*{2.8.3 Documenting Libraries and/or Applications}

As mentioned before, for each a \texttt{.pl} file, \texttt{lpdoc} tries to determine whether it is a library or the main file of an application, and documents it accordingly. Any combination of libraries and/or main files of applications can be used arbitrarily as components or main files of a \texttt{lpdoc} manual. Some typical combinations are:

- \textit{Main file is a library, no components:} A manual of a simple library, which appears externally as a single module. The manual describes the purpose of the library and its interface.
- \textit{Main file is an application, no components:} A manual of a simple application.
- \textit{Main file is a library, components are also libraries:} This can be used for example for generating an internals manual of a library. The main file describes the purpose and use of the library, while the components describe the internal modules of the library.
- \textit{Main file is an application, components are libraries:} This can be used similarly for generating an internals manual of an application. The main file describes the purpose and use of the application, while the components describe the internal modules which compose the application.
- \textit{Main file is a (pseudo-)application, components are libraries:} A manual of a complex library made up of smaller libraries (for example, the \texttt{Prolog} library). The (pseudo-)application file contains the introductory material (title, version, etc.). Each chapter describes a particular library.
- \textit{Main file is a (pseudo-)application, components are applications:} This can be used to generate a manual of a set of applications (e.g., a set of utilities). The (pseudo-)application file contains the introductory material (title, version, etc.). Each chapter describes a particular component application.

\subsection*{2.8.4 Documenting files which are not modules}

Sometimes it is difficult for \texttt{lpdoc} to distinguish include files and Ciao packages from normal \texttt{user} files (i.e., normal code files but which are not modules). The distinction is important because the former are quite different in their form of use (they are loaded via \texttt{include/1} or \texttt{use_package/1} declarations instead of \texttt{ensure_loaded/1}) and effect (since they are included, they 'export' operators, declarations, etc.), and should typically be documented differently. There is a special \texttt{doc/2} declaration (\texttt{:- doc(filetype,...).}) which provides a way of defining the intended use of the file. This declaration is normally not needed in modules, include files, or packages, but should be added in user files (i.e., those meant to be loaded using \texttt{ensure_loaded/1}). Adding this declaration will, for example, avoid spurious documentation of the declarations in the \texttt{assertions} package themselves when this package is included in a user file.

\subsection*{2.8.5 Splitting large documents into parts}

As mentioned before, in \texttt{lpdoc} each documented file (each component) corresponds to a chapter in the generated manual. In large documents, it is sometimes convenient to build a super-structure of parts, each of which groups several chapters. There is a special value of the
Chapter 2: Generating Installing and Accessing Manuals

2.8.6 Documenting reexported predicates

Reexported predicates, i.e., predicates which are exported by a module m1 but defined in another module m2 which is used by m1, are normally not documented in the original module, but instead a simple reference is included to the module in which it is defined. This can be changed, so that the documentation is included in the original module, by using a doc/2 declaration with doinclude in the first argument (see the comments library). This is often useful when documenting a library made of several components. For a simple user’s manual, it is often sufficient to include in the lpdoc SETTINGS.pl file the principal module, which is the one which users will do a use_module/1 of, in the manual. This module typically exports or reexports all the predicates which define the library’s user interface. Note, however, that currently, due to limitations in the implementation, only the comments inside assertions (but not those in doc/2 declarations) are included for reexported predicates.

2.8.7 Separating the documentation from the source file

Sometimes one would not like to include long introductory comments in the module itself but would rather have them in a different file. This can be done quite simply by using the @include command. For example, the following declaration:

```
:- doc(module,"@include{Intro.lpdoc}").
```

will include the contents of the file Intro.lpdoc as the module description.

Alternatively, sometimes one may want to generate the documentation from a completely different file. Assuming that the original module is ml.pl, this can be done by calling the module containing the documentation ml_doc.pl. This ml_doc.pl file is the one that will be included in the lpdoc SETTINGS.pl file, instead of ml.pl. lpdoc recognizes and treats such _doc files specially so that the name without the _doc part is used in the different parts of the documentation, in the same way as if the documentation were placed in file ml.

2.8.8 Generating auxiliary files (e.g. READMEs)

Using lpdoc it is often possible to use a common source for documentation text which should appear in several places. For example, assume a file INSTALLATION.lpdoc contains text (with lpdoc formatting commands) describing an application. This text can be included in a section of the main file documentation as follows:

```
:- doc(module,"  ...
@section{Installation instructions}
@include{INSTALLATION.lpdoc}
  ...
  ").
```

Note: significant parts of this are obsolete. They must be updated to describe lpdoc version 2.0.
At the same time, this text can be used to generate a nicely formatted INSTALLATION file in ascii, which can perhaps be included in the top level of the source directory of the application. To this end, an INSTALL.pl file as follows can be constructed:

```prolog
:- use_package([assertions]).
:- doc(filetype, application). %% forces file to be documented as an application
:- doc(title,"Installation instructions").
:- doc(module,"@include{INSTALLATION.lpdoc}").
```

Then, the ascii INSTALLATION file can be generated by simply running lpdoc ascii in a directory with a SETTINGS.pl file where MAIN is set to INSTALLATION.pl.

2.9 Troubleshooting

These are some common errors which may be found using lpdoc and the usual fix:

- Sometimes, messages of the type:
  ```
gmake: *** No rule to make target `myfile.texic', needed by `main.texic'. Stop.
```
appear (i.e., in the case above when running (g)make main, target). Since lpdoc definitely knows how to make a .texic file given a .pi file, this means (in make's language) that it cannot find the corresponding .pi file (myfile.pl in the case above). The usual reason for this is that there is no directory path to this file declared in the SETTINGS.pl file.

- Messages of the type:
  ```
  ! No room for a new `@write .
  ```
while converting from .texi to .dvi (i.e., while running tex). These messages are tex's way of saying that an internal area (typically for an index) is full. This is normally because more indices were selected in the INDICES variable of the SETTINGS.pl file than the maximum number supported by the installed version of tex/ texinfo installations, as mentioned in Section 2.2 [Generating a manual], page 15. The easiest fix is to reduce the number of indices generated. Alternatively, it may be possible to recompile your local tex/ texinfo installation with a higher number of indices.

- Missing links in info files (a section which exists in the printed document cannot be accessed in the on-line document) can be due to the presence of a colon (:), a comma (,), a double dash (--), or other such separators in a section name. Due to limitations of info section names cannot contain these symbols.

- Menu listings in info which do not work (i.e., the menu listings are there, but they cannot be followed): see if they are indented. In that case it is due to an itemize or enumerate which was not closed.
Chapter 3: Documentation Mark-up Language and Declarations

Author(s): Manuel Hermenegildo.

This defines the admissible uses of the doc/2 declaration (which is used mainly for adding machine readable comments to programs), the formatting commands which can be used in the text strings inside these comments, and some related properties and data types. These declarations are ignored by the compiler in the same way as classical comments. Thus, they can be used to document the program source in place of (or in combination with) the normal comments typically inserted in the code by programmers. However, because they are more structured and they are machine-readable, they can also be used to generate printed or online documentation automatically, using the lpdoc automatic documentation generator. These textual comments are meant to be complementary to the formal statements present in assertions (see the assertions library).

3.1 Usage and interface (comments)

- Library usage:
  It is not necessary to use this library in user programs. The recommended procedure in order to make use of the doc/2 declarations that this library defines is to include instead the assertions package, which provides efficient support for all assertion- and comment-related declarations, using one of the following declarations, as appropriate:
  
  ```prolog
  :- module(...,...,[assertions]).
  :- use_package(assertions).
  ```

- Exports:
  - Predicates:
    doc_id_type/3.
  - Properties:
    docstring/1, stringcommand/1.
  - Regular Types:
    version_descriptor/1, filetype/1.

- Other modules used:
  - System library modules:
    strings.
  - Internal (engine) modules:
    term_basic, arithmetic, atomic_basic, attributes, basic_prop, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

3.2 Documentation on exports (comments)

docstring/1: PROPERTY
  Defines the format of the character strings which can be used in machine readable comments (doc/2 declarations) and assertions. These character strings can include certain formatting commands.
• All printable characters are admissible in documentation strings except "@", "{," and "}". To produce these characters the following escape sequences should be used, respectively: @@, @{, and @}.

• In order to allow better formatting of on-line and printed manuals, in addition to normal text, certain formatting commands can be used within these strings. The syntax of all these commands is:
  @command
  (followed by either a space or {}), or
  @command{body}

  where command is the command name and body is the (possibly empty) command body.

  The set of commands currently admitted can be found in the documentation for the predicate stringcommand/1.

Usage: docstring(Text)
— Description: Text is a documentation string.

stringcommand/1:

PROPERTY

Defines the set of structures which can result from parsing a formatting command admissible in comment strings inside assertions.

In order to make it possible to produce documentation in a wide variety of formats, the command set is kept small. The names of the commands are intended to be reminiscent of the commands used in the LaTeX text formatting system, except that "@" is used instead of "\". Note that \ would need to be escaped in ISO-Prolog strings, which would make the source less readable (and, in any case, many ideas in LaTeX were taken from scribe, where the escape character was indeed @!).

The following are the currently admissible commands.

• Formatting commands:

  The following commands are used to format certain words or sentences in a special font, build itemized lists, introduce sections, include examples, etc.

  @comment{text}
  text will be treated as a comment and will be ignored.

  @begin{itemize}
  marks the beginning of an itemized list. Each item should be in a separate paragraph and preceded by an @item command.

  @item
  marks the beginning of a new item in an itemized list.

  @end{itemize}
  marks the end of an itemized list.

  @begin{enumerate}
  marks the beginning of an enumerated list. Each item should be in a separate paragraph and preceded by an @item command.

  @end{enumerate}
  marks the end of an enumerated list.

  @begin{description}
  marks the beginning of a description list, i.e., a list of items and their description (this list describing the different allowable commands is in fact a description list). Each item should be in a separate paragraph and contained in an @item{itemtext} command.
marks the beginning of a new item in description list, and contains the header for the item.

marks the end of a description list.

marks the beginning of fixed format text, such as a program example. A fixed-width, typewriter-like font is used.

marks the end of formatted text.

marks the beginning of a section of text in a framed box, with round corners.

marks the end of a section of text in a framed box.

marks the beginning of a section of text in a framed box, for alert messages.

marks the end of the alert message.

starts a section whose title is text. Due to a limitation of the info format, do not use : or - or , in section, subsection, title (chapter), etc. headings.

starts a subsection whose title is text.

places text in a footnote.

introduces horizontal filling space (may be ignored in certain formats).

text will be formatted in bold face or any other strong face.

text will be formatted in italics face or any other emphasis face.

text will be formatted in a fixed-width font (i.e., typewriter-like font).

key is the identifier of a keyboard key (i.e., a letter such as a, or a special key identifier such as RET or DEL) and will be formatted as (LFD) or in a fixed-width, typewriter-like font.

generates N blank lines of space. Forces also a paragraph break.

forces a paragraph break; in the same way as leaving one or more blank lines.

used at the beginning of a paragraph, states that the first line of the paragraph should not be indented. Useful, for example, for avoiding indentation on paragraphs that are continuations of other paragraphs, such as after a verbatim.

Indexing commands:
The following commands are used to mark certain words or sentences in the text as concepts, names of predicates, libraries, files, etc. The use of these commands is highly recommended, since it results in very useful indices with little effort.
@index{text}

_text_ will be printed in an emphasized font and will be included in the concept definition index (and also in the usage index). This command should be used for the first or _definitional_ appearance(s) of a concept. The idea is that the concept definition index can be used to find the definition(s) of a concept.

@cindex{text}

_text_ will be included in the concept index (and also in the usage index), but it is not printed. This is used in the same way as above, but allows sending to the index a different text than the one that is printed in the text.

@concept{text}

_text_ will be printed (in a normal font). This command is used to mark that some text is a defined concept. In on-line manuals, a direct access to the corresponding concept definition may also be generated. A pointer to the place in which the @concept command occurs will appear only in the usage index.

@pred{predname}

_predname_ (which should be in functor/arity form) is the name of a predicate and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a predicate (or a property or type) in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding predicate definition may also be generated.

@op{operatorname}

_operatorname_ (which should be in functor/arity form) is the name of an operator and will be printed in fixed-width, typewriter-like font. This command should be used when referring to an operator in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding operator definition may also be generated.

@decl{declname}

_declname_ (which should be in functor/arity form) is the name of a declaration and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a declaration in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding declaration definition may also be generated.

@lib{libname}

_libname_ is the name of a library and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a module or library in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding module definition may also be generated.

@apl{aplname}

_aplname_ is the name of an application and will be printed in fixed-width, typewriter-like font. This command should be used when referring to an application in a documentation string. A reference will be included in the usage index.
Chapter 3: Documentation Mark-up Language and Declarations

@file{filename}

_filename_ is the name of a file and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a file in a documentation string. A reference will be included in the usage index.

@var{varname}

_varname_ is the name of a variable and will be formatted in an emphasized font. Note that when referring to variable names in a _pred/1_ declaration, such names should be enclosed in @var commands for the automatic documentation system to work correctly.

- **Referencing commands:**
  
The following commands are used to introduce bibliographic citations and references to sections, urls, email addresses, etc.

@cite{keyword}

_keyword_ is the identifier of a bibliographic entry. Such entry is assumed to reside in one of a number of _bibtex_ files (.bib files). A reference in brackets ([ ]) is inserted in the text and the full reference is included at the end, with all other references, in an appendix. For example, @cite{iso-prolog} will introduce a citation to a bibliographic entry whose keyword is iso-prolog. The list of bibliography files which will be searched for a match is determined by the _BIBFILES_ variable of the _lpdoc SETTINGS_ file.

@ref{section title}

Introduces at point a reference to the section or node _section title_, where _section title_ must be the exact text of the section title.

@uref{URL}

Introduces at point a reference to the Universal Resource Locator (i.e., a WWW address ' _URL_').

@uref{text}{URL}

Introduces at point a reference to the Universal Resource Locator URL, associated to the text _text_.

@email{address}

Introduces at point a reference to _email address_ address.

@email{text}{address}

Introduces at point a reference to the email address address, associated to the text _text_.

@author{text}

text will be printed (in a normal font). This command is used to reference the name of an author (not necessarily establishing the module authorship).

- **Date and Version:**

  @today prints the current _date_.

  @version prints the _version_ of the current manual.

- **Mathematics:**

  The following commands are used to format text in mathematical.

  @math{text}

  In-line typeset the _text_ formula.
\begin{displaymath}
marks the beginning of a formula (useful for long formulas).
\end{displaymath}

\begin{displaymath}
marks the end of the (long) formula.
\end{displaymath}

\defmathcmd{cmd}{n}{def}
defines the math command \texttt{cmd}, taking \texttt{n} arguments, which is expanded as \texttt{def}. Arguments are denoted as \#1, ..., \#n inside \texttt{def}.

\defmathcmd{cmd}{def}
defines the math command \texttt{cmd}, which is expanded as \texttt{def} (with no arguments).

- **Inclusion commands:**
The following commands are used to include code or strings of text as part of documentation. The latter may reside in external files or in the file being documented. The former must be part of the module being documented. There are also commands for inserting and scaling images.

\begin{itemize}
  \item \texttt{\include{filename}}
  \begin{description}
  \item[the contents of filename] will be included in-line, as if they were part of the string. This is useful for common pieces of documentation or storing in a separate file long explanations if they are perceived to clutter the source file.
  \end{description}

  \item \texttt{\includeverbatim{_/i/ename}}
  \begin{description}
  \item[as above, but the contents of the file are included verbatim, i.e., commands within the file are not interpreted.] This is useful for including code examples which may contain \texttt{@}'s, etc. Note that this only means that the file will be included as is. If you want the string to be represented in verbatim mode in the output, you must surround the \texttt{\includeverbatim{_/t/ename}} with \texttt{\begin{verbatim}} and \texttt{\end{verbatim}}.
  \end{description}

  \item \texttt{\includefact{factname}}
  \begin{description}
  \item[it is assumed that the file being documented contains a fact of the predicate \texttt{factname/1}, whose argument is a character string.] The contents of that character string will be included in-line, as if they were part of the documentation string. This is useful for sharing pieces of text between the documentation and the running code. An example is the text which explains the usage of a command (options, etc.).
  \end{description}

  \item \texttt{\includedef{predname}}
  \begin{description}
  \item[it is assumed that the file being documented contains a definition for the predicate \texttt{predname}.] The clauses defining this predicate will be included in-line, in verbatim mode, as if they were part of the documentation string.
  \end{description}

  \item \texttt{\image{epsfile}}
  \begin{description}
  \item[including an image] at point, contained in file \texttt{epsfile}. The image file should be in \texttt{encapsulated postscript} format.
  \end{description}

  \item \texttt{\image{epsfile}{width}{height}}
  \begin{description}
  \item[same as above, but \texttt{width} and \texttt{height} should be integers which provide a size (in points) to which the image will be scaled.]
  \end{description}
\end{itemize}

- **Accents and special characters:**
The following commands can be used to insert accents and special characters.
\[ \theta \{o\} \Rightarrow \theta \]
\[ \theta'\{o\} \Rightarrow \theta \]
\[ \theta^{-}\{o\} \Rightarrow \theta \]
\[ \theta..\{o\} \Rightarrow \theta \]
\[ \theta^"\{o\} \Rightarrow \theta \]
\[ \theta^{-}\{o\} \Rightarrow \theta \]
\[ \theta=\{o\} \Rightarrow \theta \]
\[ \theta..\{o\} \Rightarrow \theta \]
\[ \thetau\{o\} \Rightarrow \theta \]
\[ \thetav\{o\} \Rightarrow \theta \]
\[ \thetaH\{o\} \Rightarrow \theta \]
\[ \theta t\{oo\} \Rightarrow \theta \]
\[ \theta c\{o\} \Rightarrow \theta \]
\[ \theta d\{o\} \Rightarrow \theta \]
\[ \theta b\{o\} \Rightarrow \theta \]
\[ \theta e \Rightarrow \theta \]
\[ \theta E \Rightarrow \theta \]
\[ \theta ae \Rightarrow \theta \]
\[ \theta AE \Rightarrow \theta \]
\[ \theta aa \Rightarrow \theta \]
\[ \theta AA \Rightarrow \theta \]
\[ \theta o \Rightarrow \theta \]
\[ \theta O \Rightarrow \theta \]
\[ \theta l \Rightarrow \theta \]
\[ \theta L \Rightarrow \theta \]
\[ \theta ss \Rightarrow \theta \]
\[ \theta ? \Rightarrow \theta \]
\[ \theta ! \Rightarrow \theta \]
\[ \theta i \Rightarrow \theta \]
\[ \theta j \Rightarrow \theta \]
\[ \theta copyright \Rightarrow \theta \]
\[ \theta iso \Rightarrow \theta \]
\[ \theta bullet \Rightarrow \theta \]
\[ \theta result \Rightarrow \theta \]

Usage: \texttt{stringcommand(CO)}

- \textit{Description}: \texttt{CO} is a structure denoting a command that is admissible in strings inside assertions.
version_descriptor/1:
A structure denoting a complete version description:

version_descriptor([]).

version_descriptor(version(Version, Date)) :-
  version_number(Version),
  ymd_date(Date).

version_descriptor(version(Version, Date, Time)) :-
  version_number(Version),
  ymd_date(Date),
  time_struct(Time).

Usage: version_descriptor(Descriptor)
  — Description: Descriptor is a complete version descriptor.

filetype/1:
Intended uses of a file:

filetype(module).
filetype(user).
filetype(include).
filetype(package).
filetype(part).

Usage: filetype(Type)
  — Description: Type describes the intended use of a file.

3.3 Documentation on internals (comments)

doc/2:
This declaration provides one of the main means for adding machine readable comments
to programs (the other one is adding documentation strings to assertions).

Usage 1: :- doc(CommentType, TitleText).
  — Description: Provides a title for the module, library, or application. When generating
documentation automatically, the text in TitleText will be used appropriately (e.g.,
in the cover page as document title or as chapter title if part of a larger document).
This will also be used as a brief description of the manual in on-line indices. There
should be at most one of these declarations per module.
  — Example:
    :- doc(title,"Documentation-Oriented Assertions").
  — The following properties should hold upon exit:
    CommentType=title
    TitleText is a documentation string.

Usage 2: :- doc(CommentType, SubtitleText).
Chapter 3: Documentation Mark-up Language and Declarations

— *Description:* Provides a *subtitle*, an explanatory or alternate *title*. The subtitle will be displayed under the proper title.

— *Example:*
  
  ```prolog
  :- doc(title,"Dr. Strangelove").
  :- doc(subtitle,"How I Learned to Stop Worrying and Love the Bomb").
  ```

— *The following properties should hold upon exit:*

  **CommentType=subtitle**  
  
  **SubtitleText** is a documentation string.  

  Usage 3: `:- doc(CommentType,SubtitleText).`

— *Description:* Provides additional *subtitle* lines. This can be, e.g., an explanation of the application to add to the title, the address of the author(s) of the application, etc. When generating documentation automatically, the text in *SubtitleText* will be used accordingly. Several of these declarations can appear per module, which is useful for, e.g., multiple line addresses.

— *Example:*

  ```prolog
  :- doc(subtitle_extra,"A Reference Manual").
  :- doc(subtitle_extra,"Technical Report 1/1.0").
  ```

— *The following properties should hold upon exit:*

  **CommentType=subtitle_extra**  
  
  **SubtitleText** is a documentation string.  

  Usage 4: `:- doc(CommentType,SubtitleText).`

— *Description:* The name of the logo image for the manual.

— *The following properties should hold upon exit:*

  **CommentType=logo**  
  
  **SubtitleText** is any term.  

  Usage 5: `:- doc(CommentType,AuthorText).`

— *Description:* Provides the author(s) of the module or application. If present, when generating documentation for the module automatically, the text in *AuthorText* will be placed in the corresponding chapter or front page. There can be more than one of these declarations per module. In order for author indexing to work properly, please use one author declaration per author. If more explanation is needed (who did what when, etc.) use an acknowledgements comment.

— *Example:*

  ```prolog
  :- doc(author,"Alan Robinson").
  ```

— *The following properties should hold upon exit:*

  **CommentType=author**  
  
  **AuthorText** is a documentation string.  

  Usage 6: `:- doc(CommentType,Text).`

— *Description:* Provides the physical and electronic *address*, or any other contact information for the authors of the module or application.

— *Example:*

  ```prolog
  :- doc(address,"Syracuse University").
  ```

— *The following properties should hold upon exit:*

  **CommentType=address**  
  
  **Text** is a documentation string.  

  Usage 6: `:- doc(CommentType,Text).`
Usage 7: :- doc(CommentType,AckText).
   — Description: Provides acknowledgements for the module. If present, when generating
documentation for the module automatically, the text in AckText will be placed in
the corresponding chapter or section. There can be only one of these declarations per
module.
   — Example:
     :- doc(ack,"Module was written by Alan, but others helped.").
   — The following properties should hold upon exit:
     CommentType=ack
     AckText is a documentation string.

Usage 8: :- doc(CommentType,CopyrightText).
   — Description: Provides a copyright text. This normally appears somewhere towards
the beginning of a printed manual. There should be at most one of these declarations
per module.
   — Example:
     :- doc(copyright,"Copyright © 2001 FSF.").
   — The following properties should hold upon exit:
     CommentType=copyright
     CopyrightText is a documentation string.

Usage 9: :- doc(CommentType,SummaryText).
   — Description: Provides a brief global explanation of the application or library. The
text in SummaryText will be used as the abstract for the whole manual. There should
be at most one of these declarations per module.
   — Example:
     :- doc(summary,"This is a @bf{very} important library.").
   — The following properties should hold upon exit:
     CommentType=summary
     SummaryText is a documentation string.

Usage 10: :- doc(CommentType,CommentText).
   — Description: Provides the main comment text for the module or application. When
generating documentation automatically, the text in CommentText will be used as the introduction or main body of the corresponding chapter or manual. There should
be at most one of these declarations per module. CommentText may use sections if
substructure is needed.
   — Example:
     :- doc(module,"This module is the @lib{comments} library.").
   — The following properties should hold upon exit:
     CommentType=module
     CommentText is a documentation string.

Usage 11: :- doc(CommentType,CommentText).
   — Description: Provides additional comments text for a module or application. When
generating documentation automatically, the text in CommentText will be used in one
of the last sections or appendices of the corresponding chapter or manual. There
should be at most one of these declarations per module. CommentText may use subsections if substructure is needed.
Chapter 3: Documentation Mark-up Language and Declarations

Example:

```prolog
:- doc(appendix,"Other module functionality...").
```

The following properties should hold upon exit:

- `CommentType=appendix` ( = /2)
- `CommentText` is a documentation string. ( docstring/1)

Usage 12: `- doc(CommentType,CommentText).`

Description: Provides a description of how the library should be loaded. Normally, this information is gathered automatically when generating documentation automatically. This declaration is meant for use when the module needs to be treated in some special way. There should be at most one of these declarations per module.

Example:

```prolog
:- doc(usage,"Do not use: still in development!").
```

The following properties should hold upon exit:

- `CommentType=usage` ( = /2)
- `CommentText` is a documentation string. ( docstring/1)

Usage 13: `- doc(CommentType,Section).`

Description: Insert a program section with name `Section`. Sectioning commands allow a structured separation of the program into parts. The division is only for documentation purposes, so visibility and scope of definitions is not affected by sectioning commands.

Example:

```prolog
:- doc(section,"Main Steps of the Algorithm").
```

The following properties should hold upon exit:

- `CommentType=section` ( = /2)
- `Section` is a documentation string. ( docstring/1)

Usage 14: `- doc(CommentType,SubSection).`

Description: Insert a program subsection with name `SubSection` (see program section command for more details).

Example:

```prolog
:- doc(subsection,"Auxiliary Definitions").
```

The following properties should hold upon exit:

- `CommentType=subsection` ( = /2)
- `SubSection` is a documentation string. ( docstring/1)

Usage 15: `- doc(PredName,CommentText).`

Description: Provides an introductory comment for a given predicate, function, property, type, etc., denoted by `PredName`. When generating documentation for the module automatically, the text in `Text` will be used as the introduction of the corresponding predicate/function/... description. There should be at most one of these declarations per predicate, function, property, or type.

Example:

```prolog
:- doc(doc/2,"This declaration provides one of the main means for adding @concept{machine readable comments} to programs.").
```

The following properties should hold upon exit:

- `PredName` is a Name/Arity structure denoting a predicate name:
predname(P/A) :-
    atm(P),
    nnegint(A).

CommentText is a documentation string.

Usage 16: :- doc(CommentType,CommentText).
  — Description: Documents a known bug or planned improvement in the module or application. Several of these declarations can appear per module. When generating documentation automatically, the text in the Text fields will be used as items in an itemized list of module or application bugs.
  — Example:
    :- doc(bug,"Comment text still has to be written by user.").
  — The following properties should hold upon exit:
    CommentType=bug
    CommentText is a documentation string.

Usage 17: :- doc(Version,CommentText).
  — Description: Provides a means for keeping a log of changes. Version contains the version number and date corresponding to the change and CommentText an explanation of the change. Several of these declarations can appear per module. When generating documentation automatically, the texts in the different CommentText fields typically appear as items in an itemized list of changes. The emacs Ciao mode helps tracking version numbers by prompting for version comments when files are saved. This mode requires version comments to appear in reverse chronological order (i.e., the topmost comment should be the most recent one).
  — Example:
    :- doc(version(l*l+21,1998/04/18,15:05*01+'EST'), "Added some missing comments. (Manuel Hermenegildo)").
  — The following properties should hold upon exit:
    Version is a complete version descriptor.
    CommentText is a documentation string.

Usage 18: :- doc(CommentType,VersionMaintenanceType).
  — Description: Defines the type of version maintenance that should be performed by the emacs Ciao mode.
  — Example:
    :- doc(version_maintenance,dir('../version')).
Version control info is kept in directory ../version. See the definition of version_maintenance_type/1 for more information on the different version maintenance modes. See the documentation on the emacs Ciao mode in the Ciao manual for information on how to automatically insert version control doc/2 declarations in files.

The version maintenance mode can also be set alternatively by inserting a comment such as:

```%
% Local Variables:
% mode: CIAO
% update-version-comments: "off"
%
% End:
```

The lines above instruct emacs to put the buffer visiting the file in emacs Ciao mode and to turn version maintenance off. Setting the version maintenance mode in this way
has the disadvantage that \texttt{lpdoc} will not be aware of the type of version maintenance being performed (the lines above are comments for Prolog). However, this can be useful in fact for setting the \textit{version maintenance mode for packages} and other files meant for inclusion in other files, since that way the settings will not affect the file in which the package is included.

\begin{itemize}
  \item \textit{The following properties should hold upon exit:}
    \begin{description}
      \item[CommentType] = \texttt{version\_maintenance} \hfill ( = /2)
      \item[VersionMaintenanceType] a type of version maintenance for a file. \hfill (version\_maintenance\_type/1)
    \end{description}

\end{itemize}

\textbf{Usage 19}: \texttt{:- doc(CommentType, PredName).}

\begin{itemize}
  \item \textit{Description}: This is a special case that is used to control which predicates are included in the documentation. Normally, only exported predicates are documented. A declaration \texttt{:- doc(doinclude, PredName)} forces documentation for predicate (or type, property, function, ...) \texttt{PredName} to be included even if \texttt{PredName} is not exported. Also, if \texttt{PredName} is reexported from another module, a declaration \texttt{:- doc(doinclude, PredName)} will force the documentation for \texttt{PredName} to appear directly in this module.

  \item \textit{Example}:
    \begin{verbatim}
    :- doc(doinclude, p/3).
    \end{verbatim}

\end{itemize}

\textbf{Usage 20}: \texttt{:- doc(CommentType, PredName).}

\begin{itemize}
  \item \textit{Description}: A different usage which allows the second argument of \texttt{:- doc(doinclude, ...)} to be a list of predicate names.

  \item \textit{The following properties should hold upon exit:}
    \begin{description}
      \item[CommentType] = \texttt{doinclude} \hfill ( = /2)
      \item[PredName] is a Name/Arity structure denoting a predicate name:
        \begin{verbatim}
        prename(P/A) :-
          atm(P),
          nnegint(A).
        \end{verbatim}
        ( prename/1)
    \end{description}

\end{itemize}

\textbf{Usage 21}: \texttt{:- doc(CommentType, PredName).}

\begin{itemize}
  \item \textit{Description}: This is similar to the previous usage but has the opposite effect: it signals that an exported predicate should \textit{not} be included in the documentation.

  \item \textit{Example}:
    \begin{verbatim}
    :- doc(hide, p/3).
    \end{verbatim}

  \item \textit{The following properties should hold upon exit:}
    \begin{description}
      \item[CommentType] = \texttt{hide} \hfill ( = /2)
      \item[PredName] is a Name/Arity structure denoting a predicate name:
        \begin{verbatim}
        prename(P/A) :-
          atm(P),
          nnegint(A).
        \end{verbatim}
        ( prename/1)
    \end{description}

\end{itemize}

\textbf{Usage 22}: \texttt{:- doc(CommentType, PredName).}
Description: A different usage which allows the second argument of :-doc(hide,...) to be a list of predicate names.

The following properties should hold upon exit:

- CommentType=hide
- PredName is a list of prednames.

Usage 23: :- doc(CommentType,FileType).

Description: Provides a way of defining the intended use of the file. This use is normally easily inferred from the contents of the file itself, and therefore such a declaration is in general not needed. The exception is the special case of include files and Ciao packages, which are typically indistinguishable from normal user files (i.e., files which are not modules), but are however quite different in their form of use (they are loaded via include/1 or use_package/1 declarations instead of ensure_loaded/1) and effect (since they are included, they 'export' operators, declarations, etc.). Typically, it is assumed by default that files which are not modules will be used as include files or packages. Thus, a doc/2 declaration of this kind strictly only needs to be added to user-type files.

Example:

```prolog
:- doc(filetype,user).
```

There is another special case: the value `part`. This `filetype` is used to flag files which serve as introductions to boundaries between major parts in large documents. See Section 2.8.5 [Splitting large documents into parts], page 22 for details.

The following properties should hold upon exit:

- CommentType=filetype
- FileType describes the intended use of a file.

Usage 24: :- doc(CommentType,FileName).

Description: Do not document anything that comes from a file whose name (after taking away the path and the suffix) is FileName. This is used for example when documenting packages to avoid the documenter from including documentation of certain other packages which the package being documented uses.

Example:

```prolog
:- doc(nodoc,assertions).
```

The following properties should hold upon exit:

- CommentType=nodoc
- FileName is an atom.

version_number/1:

Version is a structure denoting a complete version number (major version, minor version, and patch number):

```prolog
version_number(Major*Minor+Patch) :-
    int(Major),
    int(Minor),
    int(Patch).
```

Usage: version_number(Version)

Description: Version is a complete version number
Chapter 3: Documentation Mark-up Language and Declarations

ymd_date/1:  
A Year/Month/Day structure denoting a date:  

ymd_date(Y/M/D) :-  
    int(Y),  
    int(M),  
    int(D).

Usage: ymd_date(Date)  
— Description: Date is a Year/Month/Day structure denoting a date.

time_struct/1:  
A structure containing time information:  

time_struct(Hours:Minutes*Seconds+TimeZone) :-  
    int(Hours),  
    int(Minutes),  
    int(Seconds),  
    atm(TimeZone).

Usage: time_struct(Time)  
— Description: Time contains time information.

version_maintenance_type/1:  
Possible kinds of version maintenance for a file:  

version_maintenance_type(on).  
version_maintenance_type(off).  
version_maintenance_type(dir(Path)) :-  
    atm(Path).

• on: version numbering is maintained locally in the file in which the declaration occurs,  
  i.e., an independent version number is kept for this file and the current version is given  
  by the most recent doc(version(...),...) declaration.  
• off: no version numbering maintained.  
• dir(Path): version numbering is maintained (globally) in directory Path. This is  
  useful for maintaining a common global version for an application which involves  
  several files.

The automatic maintenance of version numbers is typically done by the Ciao emacs mode.  
Usage: version_maintenance_type(Type)  
— Description: Type a type of version maintenance for a file.
4 The Ciao assertion package

Author(s): Manuel Hermenegildo, Francisco Bueno, German Puebla.

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 contain formal specifications (such as sets of preconditions, post-conditions, or descriptions of computations) as well as 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.

4.1 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 [PBH00]. 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.

4.2 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.
4.3 Usage and interface (assertions_doc)

- **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]).
  :- use_package([assertions]).
  ```

- **Exports:**
  - **Predicates:**
    - `check/1`, `trust/1`, `true/1`, `false/1`.

- **New operators defined:**
  - `=>/2` [975,xfx], `::/2` [978,xfx], `decl/1` [1150,fx], `decl/2` [1150,xfx], `pred/1` [1150,fx], `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], `test/1` [1150,fx], `test/2` [1150,xfx], `texec/1` [1150,fx], `texec/2` [1150,xfx], `comp/1` [1150,fx], `comp/2` [1150,xfx], `entry/1` [1150,fx], `exit/1` [1150,fx], `exit/2` [1150,xfx].

- **New declarations defined:**
  - `pred/1`, `pred/2`, `texec/1`, `texec/2`, `calls/1`, `calls/2`, `success/1`, `success/2`, `test/1`, `test/2`, `comp/1`, `comp/2`, `prop/1`, `prop/2`, `entry/1`, `exit/1`, `exit/2`, `modedef/1`, `decl/1`, `decl/2`, `doc/2`, `comment/2`.

- **Other modules used:**
  - **System library modules:**
    - `assertions/assertions_props`.
  - **Internal (engine) modules:**
    - `term_basic`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_compare`, `term_typing`, `hiord_rt`, `debugger_support`.

4.4 Documentation on new declarations (assertions_doc)

`pred/1:`

**DECLARATION**

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/assertions_props`.

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: :- pred AssertionBody.
  - The following properties should hold at call time:
    AssertionBody is an assertion body. (assert_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: :- AssertionStatus pred AssertionBody.
  - The following properties should hold at call time:
    AssertionStatus is an acceptable status for an assertion. (assert_status/1)
    AssertionBody is an assertion body. (assert_body/1)

texec/1: DECLARATION
This assertion is similar to a calls/1 assertion but it is used to provide input data and
execution commands to the unit-test driver.

Usage: :- texec AssertionBody.
  - The following properties should hold at call time:
    AssertionBody is a call assertion body. (c_assert_body/1)

texec/2: DECLARATION
This assertion is similar to a texec/1 assertion but it is explicitly qualified with an
assertion status. Non-qualified texec/1 assertions are assumed to have check status.

Usage: :- AssertionStatus texec AssertionBody.
  - The following properties should hold at call time:
    AssertionStatus is an acceptable status for an assertion. (assert_status/1)
    AssertionBody is a call assertion body. (c_assert_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. (c_assert_body/1)
calls/2:
  DECLARATION
  This assertion is similar to a calls/1 assertion but it is explicitly qualified with an assertion status. Non-qualified calls/1 assertions are assumed to have check status.
  Usage: :- AssertionStatus calls AssertionBody.
     — The following properties should hold at call time:
       AssertionStatus is an acceptable status for an assertion.  \textit{(assert\_status/1)}
       AssertionBody is a call assertion body.  \textit{(c\_assrt\_body/1)}

success/1:
  DECLARATION
  This assertion is similar to a \texttt{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 \texttt{length/2} predicate if it is called as in the first mode of usage above (note that the previous \texttt{pred} assertion already conveys such information, however it also compelled the predicate calls, while the success assertion does not):
  \begin{verbatim}
  :- success length(L,N) : list * var => list * integer.
  \end{verbatim}
     — The following properties should hold at call time:
       AssertionBody is a predicate assertion body.  \textit{(s\_assrt\_body/1)}

success/2:
  DECLARATION
  success assertion This assertion is similar to a \texttt{success/1} assertion but it is explicitly qualified with an assertion status. The status of non-qualified \texttt{success/1} assertions is assumed to be check.
  Usage: :- AssertionStatus success AssertionBody.
     — The following properties should hold at call time:
       AssertionStatus is an acceptable status for an assertion.  \textit{(assert\_status/1)}
       AssertionBody is a predicate assertion body.  \textit{(s\_assrt\_body/1)}

test/1:
  DECLARATION
  This assertion is similar to a success assertion but it specifies a concrete test case to be run in order verify (partially) that the predicate is working as expected. For example, the following test will verify that the length predicate works well for the particular list given:
  \begin{verbatim}
  :- test length(L,N) : ( L = [1,2,5,2] ) => ( N = 4 ).
  \end{verbatim}
  Usage: :- test AssertionBody.
     — The following properties should hold at call time:
       AssertionBody is a predicate assertion body.  \textit{(s\_assrt\_body/1)}
**test/2:**

This assertion is similar to a `test/1` assertion but it is explicitly qualified with an assertion status. Non-qualified `test/1` assertions are assumed to have check status. In this context, check means that the test should be executed when the developer runs the test battery.

**Usage:** 

- The following properties should hold at call time:
  - `AssertionStatus` is an acceptable status for an assertion.  
    
  - `AssertionBody` is a predicate assertion body.

**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):

```prolog
:- comp append(Xs,Ys,Zs) : var * var * var + not_fail.
```

**Usage:** 

- The following properties should hold at call time:
  - `AssertionBody` is a comp assertion body.

**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:** 

- The following properties should hold at call time:
  - `AssertionStatus` is an acceptable status for an assertion.
  - `AssertionBody` is a comp assertion body.

**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 `ciaopp` 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 6 [Declaring regular types], page 57 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:**

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:** 
:- AssertionStatus prop AssertionBody.

- The following properties should hold at call time:
  
  **AssertionStatus** is an acceptable status for an assertion.  
  **AssertionBody** is an assertion body.

**entry/1:**

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.

**exit/1:**

This type of assertion provides information about the answers that an exported predicate provides for external calls. It is identical syntactically to a success/1 assertion. However, it describes only external answers, i.e., answers to the exported predicates of a module from outside the module, or answers to the predicates in a non-modular file from other files (or the user). The described answers may be conditioned to a particular way of calling the predicate. E.g.:

```prolog
:- exit length(L,N) : list * var => list * integer.
```

**Usage:** 
:- exit AssertionBody.

- The following properties should hold at call time:
  
  **AssertionBody** is a predicate assertion body.
exit/2:  
DECLAREATION

exit assertion This assertion is similar to an exit/1 assertion but it is explicitly qualified with an assertion status. Non-qualified exit/1 assertions are assumed the qualifier check.

Usage: :- AssertionStatus exit AssertionBody.
— The following properties should hold at call time:
   AssertionStatus is an acceptable status for an assertion. (assrt_status/l)
   AssertionBody is a predicate assertion body. (s_assrt_body/l)

modedef/1:  
DECLAREATION

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. (assrt_body/l)

decl/1:  
DECLAREATION

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. (assrt_body/l)

decl/2:  
DECLAREATION

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: :- AssertionStatus decl AssertionBody.
— The following properties should hold at call time:
   AssertionStatus is an acceptable status for an assertion. (assrt_status/l)
   AssertionBody is an assertion body. (assrt_body/l)
The lpdoc Documentation Generator

\[ \text{doc/2:} \]
\text{DECLARATION}

Usage: :- doc(Pred,Comment).

- Description: Documentation. This assertion provides a text Comment for a given predicate Pred.
- The following properties should hold at call time:
  \( \text{Pred} \) is a head pattern. \( \text{(head_pattern/1)} \)
  Comment is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by \text{stringcommand/1}. See the \text{lpdoc} manual for documentation on comments. \( \text{(docstring/1)} \)

\[ \text{comment/2:} \]
\text{DECLARATION}

Usage: :- \text{comment} (Pred, Comment).

- Description: An alias for \text{doc/2} (deprecated, for compatibility with older versions).
- The following properties should hold at call time:
  \( \text{Pred} \) is a head pattern. \( \text{(head_pattern/1)} \)
  Comment is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by \text{stringcommand/1}. See the \text{lpdoc} manual for documentation on comments. \( \text{(docstring/1)} \)

4.5 Documentation on exports (assertions_doc)

\[ \text{check/1:} \]
\text{PREDICATE}

Usage: check(PropertyConjunction)

- Description: This assertion provides information on a clause program point (position in the body of a clause). Calls to a \text{check/1} assertion can appear in the body of a clause in any place where a literal can normally appear. The property defined by \text{PropertyConjunction} should hold in all the run-time stores corresponding to that program point. See also Chapter 12 [Run-time checking of assertions], page 103.
- The following properties should hold at call time:
  \text{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. \( \text{(property_conjunction/1)} \)

\[ \text{trust/1:} \]
\text{PREDICATE}

Usage: trust(PropertyConjunction)

- Description: This assertion also provides information on a clause program point. It is identical syntactically to a \text{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 \text{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, \text{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. 

  \( \text{true}(\text{PropertyConjunction}) \)

  Usage: \( \text{true}(\text{PropertyConjunction}) \)

  Description: This assertion is identical syntactically to a \text{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. 

    \( \text{false}(\text{PropertyConjunction}) \)

    Usage: \( \text{false}(\text{PropertyConjunction}) \)

    Description: This assertion is identical syntactically to a \text{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.
5 Types and properties related to assertions

Author(s): Manuel Hermenegildo.

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.

5.1 Usage and interface (assertions_props)

- Library usage:
  ```prolog
  :- use_module(library(assertions_props)).
  ```
- Exports:
  - Properties:
    ```prolog
    head_pattern/1, nabody/1, docstring/1.
    ```
  - Regular Types:
    ```prolog
    assert_body/1, complex_arg_property/1, property_conjunction/1, property_starterm/1, complex_goal_property/1, dictionary/1, c_assert_body/1, s_assert_body/1, g_assert_body/1, assert_status/1, assert_type/1, predfunctor/1, propfunctor/1.
    ```
- Other modules used:
  - Internal (engine) modules:
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

5.2 Documentation on exports (assertions_props)

assert_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] [# CD]
```

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 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:

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(Paramter,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.

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:  
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
\[ :- \text{comp} \text{ var}(A) + \text{not\_further\_inst}(A). \]
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:
\[ \text{Pr} : \text{CP} \# \text{CO} \]
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_assrt_body/1: REGTYPE
This predicate defines the different types of syntax admissible in the bodies of pred/1, func/1, etc. assertions. The following are admissible:

\[
\begin{align*}
\text{Pr} : & \text{CP} \Rightarrow \text{AP} \# \text{CO} \\
\text{Pr} : & \text{CP} \Rightarrow \text{AP} \\
\text{Pr} : & \Rightarrow \text{AP} \# \text{CO} \\
\text{Pr} : & \Rightarrow \text{AP}
\end{align*}
\]

where:
- \( \text{Pr} \) is a head pattern (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 (complex_arg_property/1) which applies to the calls to the predicate.
- \( \text{AP} \) 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.
- \( \text{CO} \) 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: s_assrt_body(X)
- Description: X is a predicate assertion body.

g_assrt_body/1: REGTYPE
This predicate defines the different types of syntax admissible in the bodies of comp/1 assertions. The following are admissible:

\[
\begin{align*}
\text{Pr} : & \text{CP} + \text{GP} \# \text{CO} \\
\text{Pr} : & \text{CP} + \text{GP} \\
\text{Pr} + & \text{GP} \# \text{CO} \\
\text{Pr} + & \text{GP}
\end{align*}
\]

where:
- \( \text{Pr} \) is a head pattern (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 (complex_arg_property/1) which applies to the calls to the predicate.
- \( \text{GP} \) contains (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.
- \( \text{CO} \) 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: g_assrt_body(X)
- Description: 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(exit)`.
- `assrt_type(test)`.
- `assrt_type(texec)`.
- `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.
6 Declaring regular types

Author(s): Manuel Hermenegildo, Pedro López, Francisco Bueno.

This library package adds declarations and new operator definitions which provide simple
syntactic sugar to write regular type definitions in source code. Regular types are just properties
which have the additional characteristic of being regular types (basic_props:regtype/1),
defined below.

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.

6.1 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 asser­
tions 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 inte­
gers.” 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 [], [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_­
cat([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:
:- calls sumlist(L,N) : instantiated_to_intlist(L).

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:

:- 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):

:- 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:

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 intlist(X) it would bind X to []). 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 Chapter 12 [Run-time checking of assertions], page 103) 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:
Chapter 6: Declaring regular types

:- calls sumlist(L,N) : intlist(L).

has the desired effect. Also, the same properties can often be used as compatibility checks by writing them in the assertions as compat(Property) (basic_props:compat/1). Thus, writing:

:- success string_concat(A,B,C) => ( compat(intlist(A)),
               compat(intlist(B)),
               compat(intlist(C)) ).

also has the desired effect.

As a general rule, the properties that can be used directly for checking for compatibility should be 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 instantiation checks and should not be used as compatibility properties nor appear in the definition of a property that is to be used with compat. Examples of such predicates (for Prolog) are ==, 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, properties of computations refer to the entire execution of the call(s) that the assertion relates to. One such property is, for example, not_fail/1 (note that although it has been used as in :- comp append(Xs,Ys,Zs) + not_fail, it is in fact read as not_fail(append(Xs,Ys,Zs)); see 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 not_fail/1 for runtime checking:

not_fail(Goal):-
  if( call(Goal),
      true,               %% then
      warning(Goal) ).  %% else

where the 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) 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.

6.2 Usage and interface (regtypes_doc)

- 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.
  – Internal (engine) modules:
    term_basic.
6.3 Documentation on new declarations (regtypes_doc)

regtype/1: DECLARATION
This assertion is similar to a prop assertion but it flags that the property being documented is also a "regular type." Regular types are properties whose definitions are regular programs (see below). This allows for example checking whether it is in the class of types supported by the regular type checking and inference modules.

A regular program is defined by a set of clauses, each of the form:

\[ p(x, v_1, \ldots, v_n) :- body_1, \ldots, body_k. \]

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, \( p(f(X, Y)) :- \ldots \) is valid, but \( p(f(X, X)) :- \ldots \) 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. \( n \geq 0 \) and \( p/n \) is a parametric type functor (whereas the predicate defined by the clauses is \( p/n+1 \)).
4. \( v_1, \ldots, v_n \) 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. \( q(y, t_1, \ldots, t_m) \) where \( m \geq 0 \), \( q/m \) is a parametric type functor, not in the set of functors \( =/2, ~/2, ./3 \).
      \( t_1, \ldots, t_m \) 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.

A parametric type functor is a regular type, defined by a regular program, or a basic type. Basic types are defined in Chapter 7 [Basic data types and properties], page 61.

The set of regular 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: :- regtype AssertionBody.
- The following properties should hold at call time:
  AssertionBody is an assertion body. (assrt_body/1)

regtype/2: DECLARATION
This assertion is similar to a regtype/1 assertion but it is explicitly qualified. Non-qualified regtype/1 assertions are assumed the qualifier check. Note that checking regular type definitions should be done with the ciaopp preprocessor.

Usage: :- AssertionStatus regtype AssertionBody.
- The following properties should hold at call time:
  AssertionStatus is an acceptable status for an assertion. (assrt_status/1)
  AssertionBody is an assertion body. (assrt_body/1)
Chapter 7: Basic data types and properties

Author(s): Daniel Cabeza, Manuel Hermenegildo.

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.

7.1 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, inst/2, iso/1, deprecated/1, not_further_inst/2, sideff/2, regtype/1, native/1, native/2, no_rtcheck/1, eval/1, equiv/2, bind_ins/1, error_free/1, memo/1, filter/2, pe_type/1.
  - Regular Types:
    - term/1, int/1, nnegint/1, flt/1, num/1, atm/1, struct/1, gnd/1, gndstr/1, constant/1, callable/1, operator_specifier/1, list/1, list/2, nlist/2, sequence/2, sequence_or_list/2, character_code/1, string/1, num_code/1, predname/1, atm_or_atm_list/1, flag_values/1.

- Other modules used:
  - System library modules:
    assertions/native_props, terms_check.
  - Internal (engine) modules:
    term_basic, arithmetic, atomic_basic, attributes, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

7.2 Documentation on exports (basic_props)

term/1:

The most general type (includes all possible terms).

General properties: 

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

term(X)

- The following properties hold globally:
  - term(X) is evaluable at compile-time. (eval/1)

term(X)

- The following properties hold globally:
  - term(X) is equivalent to true. (equiv/2)

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

**int/1:**  REGTYPE
The type of integers. The range of integers is \([-2^{31}, 2^{31})\). 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. (sideff/2)

**int(T)**
— If the following properties hold at call time:
  T is currently a term which is not a free variable. (nonvar/1)
  then the following properties hold globally:
  int(T) is evaluable at compile-time. (eval/1)
  All calls of the form int(T) are deterministic. (is_det/1)

**int(T)**
— The following properties hold upon exit:
  T is an integer. (int/1)
— The following properties hold globally:
  Indicates the type of test that a predicate performs. Required by the nonfailure analysis. (test_type/2)

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

**nnegint/1:**  REGTYPE
The type of non-negative integers, i.e., natural numbers.

**General properties: nnegint(T)**
— The following properties hold globally:
  nnegint(T) is side-effect free. (sideff/2)

**nnegint(T)**
— If the following properties hold at call time:
  T is currently a term which is not a free variable. (nonvar/1)
  then the following properties hold globally:
  nnegint(T) is evaluable at compile-time. (eval/1)

**nnegint(T)**
— The following properties hold upon exit:
  T is a non-negative integer. (nnegint/1)
Chapter 7: Basic data types and properties

The following properties hold globally:
Indicates the type of test that a predicate performs. Required by the nonfailure analysis. (test_type/2)

Usage: nnegint(T)

- Description: T is a non-negative integer.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (native/1)

flt/1: REGTYPE

The type of floating-point numbers. The range of floats is the one provided by the C 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 0.0Nan

General properties: flt(T)

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

flt(T)

- If the following properties hold at call time:
  T is currently a term which is not a free variable. (nonvar/1)
  then the following properties hold globally:
  flt(T) is evaluable at compile-time. (eval/1)
  All calls of the form flt(T) are deterministic. (is_det/1)

flt(T)

- The following properties hold upon exit:
  T is a float. (flt/1)
- The following properties hold globally:
  Indicates the type of test that a predicate performs. Required by the nonfailure analysis. (test_type/2)

Usage: flt(T)

- Description: T is a float.
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (native/1)

num/1: REGTYPE

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

General properties: num(T)

- The following properties hold globally:
  num(T) is side-effect free. (sideff/2)
  num(T) is binding insensitive. (bind_ins/1)
— If the following properties hold at call time:
  T is currently a term which is not a free variable.  
then the following properties hold globally:
  num(T) is evaluable at compile-time. 
  All calls of the form num(T) are deterministic. 
num(T)
— The following properties hold upon exit:
  T is a number. 
— The following properties hold globally:
  Indicates the type of test that a predicate performs. Required by the nonfailure
  analysis. 
Usage: num(T)
— Description: T is a number.
— The following properties hold globally:
  This predicate is understood natively by CiaoPP. 

atm/1:  
The type of atoms, or non-numeric constants. The size of atoms is unbound.  
General properties: atm(T)
  — The following properties hold globally:
    atm(T) is side-effect free. 
atm(T)
  — If the following properties hold at call time:
    T is currently a term which is not a free variable.  
then the following properties hold globally:
    atm(T) is evaluable at compile-time. 
    All calls of the form atm(T) are deterministic. 
atm(T)
  — The following properties hold upon exit:
    T is an atom. 
  — The following properties hold globally:
    Indicates the type of test that a predicate performs. Required by the nonfailure
    analysis. 
Usage: atm(T)
  — Description: T is an atom.
  — The following properties hold globally:
    This predicate is understood natively by CiaoPP. 

struct/1:  
The type of compound terms, or terms with non-zeroary functors. By now there is a limit
of 255 arguments.  
General properties: struct(T)
— The following properties hold globally:
  \texttt{struct}(T) is side-effect \texttt{free}.
\texttt{struct}(T)
— If the following properties hold at call time:
  \( T \) is currently a term which is not a free variable.
  \texttt{then the following properties hold globally:}
  \texttt{struct}(T) is evaluable at compile-time.
\texttt{struct}(T)
— The following properties hold upon exit:
  \( T \) is a compound term.
Usage: \texttt{struct}(T)
— Description: \( T \) is a compound term.
— The following properties hold globally:
  This predicate is understood natively by CiaoPP.

\texttt{gnd/1}:
The type of all terms without variables.

General properties: \texttt{gnd}(T)
— The following properties hold globally:
  \texttt{gnd}(T) is side-effect \texttt{free}.
\texttt{gnd}(T)
— If the following properties hold at call time:
  \( T \) is currently ground (it contains no variables).
  \texttt{then the following properties hold globally:}
  \texttt{gnd}(T) is evaluable at compile-time.
  All calls of the form \texttt{gnd}(T) are deterministic.
\texttt{gnd}(T)
— The following properties hold upon exit:
  \( T \) is ground.
Usage: \texttt{gnd}(T)
— Description: \( T \) is ground.
— The following properties hold globally:
  This predicate is understood natively by CiaoPP.

\texttt{gndstr/1}:
General properties: \texttt{gndstr}(T)
— The following properties hold globally:
  \texttt{gndstr}(T) is side-effect \texttt{free}.
gndstr(T)
- If the following properties hold at call time:
  T is currently ground (it contains no variables).
then the following properties hold globally:
  gndstr(T) is evaluable at compile-time.
  All calls of the form gndstr(T) are deterministic.

Usage: gndstr(T)
- Description: T is a ground compound term.

constant/1:
- General properties: constant(T)
  - The following properties hold globally:
    constant(T) is side-effect free.

constant(T)
- If the following properties hold at call time:
  T is currently a term which is not a free variable.
then the following properties hold globally:
  constant(T) is evaluable at compile-time.
  All calls of the form constant(T) are deterministic.

Usage: constant(T)
- Description: T is an atomic term (an atom or a number).

callable/1:
- General properties: callable(T)
  - The following properties hold globally:
    callable(T) is side-effect free.

callable(T)
- If the following properties hold at call time:
  T is currently a term which is not a free variable.
then the following properties hold globally:
  callable(T) is evaluable at compile-time.
  All calls of the form callable(T) are deterministic.
Chapter 7: Basic data types and properties

The following properties hold upon exit:

- T is currently a term which is not a free variable. (nonvar/1)

Usage: callable(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:

- **x**fx: 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.
- **x**fy: 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.
- **y**fx: Infix, left-associative: same as above, but the other way around.
- **f**x: Prefix, non-associative: the subexpression must be of lower precedence than the operator.
- **f**y: Prefix, associative: the subexpression can be of the same precedence as the operator.
- **x**f: Postfix, non-associative: the subexpression must be of lower precedence than the operator.
- **y**f: 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. (sideff/2)

operator_specifier(X)

- If the following properties hold at call time:
  - X is currently a term which is not a free variable. (nonvar/1)
  - then the following properties hold globally:
    - operator_specifier(X) is evaluable at compile-time. (eval/1)
    - All calls of the form operator_specifier(X) are deterministic. (is_det/1)
    - Goal operator_specifier(X) produces 7 solutions. (relations/2)

operator_specifier(T)

- The following properties hold upon exit:
  - T specifies the type and associativity of an operator. (operator_specifier/1)

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
list([],).
list([a|L]) :-
    list(L).

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

list(L)
- If the following properties hold at call time:
  L is currently ground (it contains no variables).
then the following properties hold globally:
  list(L) is evaluable at compile-time.
  All calls of the form list(L) are deterministic. (eval/1)

list(T)
- The following properties hold upon exit:
  T is a list. (list/1)

Usage: list(L)
- Description: L is a list.

list/2: REGTYPE
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. (sideff/2)

list(L,T)
- If the following properties hold at call time:
  L is currently ground (it contains no variables).
  T is currently ground (it contains no variables).
then the following properties hold globally:
  list(L,T) is evaluable at compile-time. (eval/1)

list(X,T)
- The following properties hold upon exit:
  X is a list. (list/1)

Usage: list(L,T)
- Description: L is a list of Ts.

nlist/2: REGTYPE
Meta-predicate with arguments: nlist(?,(pred 1)).

General properties: nlist(L,T)
Chapter 7: Basic data types and properties

— The following properties hold globally:
  \text{nlist}(L,T) \text{ is side-effect free.} \quad (\text{sideff/2})
\text{nlist}(L,T)
— If the following properties hold at call time:
  \begin{itemize}
  \item L is currently ground (it contains no variables). \quad (\text{ground/1})
  \item T is currently ground (it contains no variables). \quad (\text{ground/1})
\end{itemize}
then the following properties hold globally:
\text{nlist}(L,T) \text{ is evaluable at compile-time.} \quad (\text{eval/1})
\text{nlist}(X,T)
— The following properties hold upon exit:
  X \text{ is any term.} \quad (\text{term/1})

Usage: \text{nlist}(L,T)
— Description: L is T or a nested list of Ts. Note that if T is term, this type is equivalent to term, this fact explain why we do not have a \text{nlist/1} type

\text{member/2:}

\text{General properties: member}(X,L)
— The following properties hold globally:
  \begin{itemize}
  \item member}(X,L) \text{ is side-effect free.} \quad (\text{sideff/2})
  \item member}(X,L) \text{ is binding insensitive.} \quad (\text{bind_ins/1})
\end{itemize}
\text{member}(X,L)
— If the following properties hold at call time:
  \begin{itemize}
  \item L is a list. \quad (\text{list/1})
\end{itemize}
then the following properties hold globally:
\text{member}(X,L) \text{ is evaluable at compile-time.} \quad (\text{eval/1})
\text{member}(X,L)
— The following properties hold upon exit:
  \begin{itemize}
  \item L is a list. \quad (\text{list/1})
\end{itemize}
\text{member}(X,L)
— If the following properties hold at call time:
  \begin{itemize}
  \item L is currently ground (it contains no variables). \quad (\text{ground/1})
\end{itemize}
then the following properties hold upon exit:
  \begin{itemize}
  \item X is currently ground (it contains no variables). \quad (\text{ground/1})
\end{itemize}

Usage: \text{member}(X,L)
— Description: X \text{ is an element of L.}

\text{sequence/2:}

\text{A sequence is formed with zero, one or more occurrences of the operator }`,`,/2. \text{ For example, } a, b, c \text{ is a sequence of three atoms, } a \text{ is a sequence of one atom.}
\text{Meta-predicate} with arguments: \text{sequence(?,(pred 1)).}

\text{General properties: sequence}(S,T)
sequence(S,T) — The following properties hold globally:
sequence(S,T) is side-effect free.

sequence(E,T) — If the following properties hold at call time:

sequence(E,T) — The following properties hold upon exit:
E is currently a term which is not a free variable.
T is currently ground (it contains no variables).

Usage: sequence(S,T)
— Description: S is a sequence of Ts.

sequence_or_list/2: 
Meta-predicate with arguments: sequence_or_list(?,(pred 1)).

character_code/1: 
Character code.
Chapter 7: Basic data types and properties

character_code(I)

- The following properties hold upon exit:
  I is an integer which is a character code. (character_code/1)

Usage: character_code(T)
- Description: T is an integer which is a character code.

string/1:

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 globally:
  string(T) is side-effect free. (sideff/2)

string(T)
- If the following properties hold at call time:
  T is currently ground (it contains no variables).
  then the following properties hold globally:
  string(T) is evaluable at compile-time. (eval/1)

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

num_code/1:

These are the ASCII codes which can appear in decimal representation of floating point and integer numbers, including scientific notation and fractionary part.

predname/1:

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

predname(P)
- If the following properties hold at call time:
  P is currently ground (it contains no variables).
  then the following properties hold globally:
  predname(P) is evaluable at compile-time. (eval/1)

Usage: predname(P)
- Description: P is a Name/Arity structure denoting a predicate name:
predname(P/A) :-
    atm(P),
    nnegint(A).
(predname/1)

Usage: predname(P)
— Description: P is a Name/Arity structure denoting a predicate name:

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 list/1, whereas the terms f(a) and [1|2] are not.

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

Usage: compat(Term,Prop)
— Description: Term is compatible with Prop
### inst/2:

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

**General properties:** `inst(Term, Prop)`
- The following properties hold globally:
  - `inst(Term, Prop)` is side-effect free.  

**Usage:** `inst(Term, Prop)`
- *Description:* `Term` is instantiated enough to satisfy `Prop`.

### iso/1:

*Meta-predicate* with arguments: `iso(goal)`.

**General properties:** `iso(G)`
- The following properties hold globally:
  - `iso(G)` is side-effect free.

**Usage:** `iso(G)`
- *Description:* Complies with the ISO-Prolog standard.

### deprecated/1:

*Meta-predicate* with arguments: `deprecated(goal)`.

**General properties:** `deprecated(G)`
- The following properties hold globally:
  - `deprecated(G)` is side-effect free.

**Usage:** `deprecated(G)`
- *Description:* DEPRECATED.

### not_further_inst/2:

*Meta-predicate* with arguments: `not_further_inst(goal, ?)`.

**General properties:** `not_further_inst(G, V)`
- The following properties hold globally:
  - `not_further_inst(G, V)` is side-effect free.

**Usage:** `not_further_inst(G, V)`
- *Description:* `V` is not further instantiated.
sideff/2:
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. (native/1) (sideff/2)

Usage: sideff(G,X)
- Description: G is side-effect X.
- If the following properties hold at call time:
  X is an element of [free,soft,hard]. (member/2)

regtype/1:
Meta-predicate with arguments: regtype goal.

General properties: regtype G
- The following properties hold globally:
  regtype G is side-effect free. (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. (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. (sideff/2)

Usage: native(Pred,Key)
- Description: This predicate is understood natively by CiaoPP as Key.
Chapter 7: Basic data types and properties

no_rtcheck/1:
  Meta-predicate with arguments: no_rtcheck(goal).
  General properties: no_rtcheck(G)
  – The following properties hold globally:
    no_rtcheck(G) is side-effect free. (sideff/2)
  Usage: no_rtcheck(G)
  – Description: Declares that the assertion in which this comp property appears must not be checked at run-time.

eval/1:
  Meta-predicate with arguments: eval(goal).
  Usage: eval(Goal)
  – Description: Goal is evaluable at compile-time.

equiv/2:
  Meta-predicate with arguments: equiv(goal,goal).
  Usage: equiv(Goal1,Goal2)
  – Description: Goal1 is equivalent to Goal2.

bind_ins/1:
  Meta-predicate with arguments: bind_ins(goal).
  Usage: bind_ins(Goal)
  – Description: Goal is binding insensitive.

error_free/1:
  Meta-predicate with arguments: error_free(goal).
  Usage: error_free(Goal)
  – Description: Goal is error free.

memo/1:
  Meta-predicate with arguments: memo(goal).
  Usage: memo(Goal)
  – Description: Goal should be memoized (not unfolded).

filter/2:
  Usage: filter(Vars,Goal)
  – Description: Vars should be filtered during global control).
flag_values/1:  
Usage: flag_values(X)  
— Description: Define the valid flag values

pe_type/1:  
Meta-predicate with arguments: pe_type(goal).  
Usage: pe_type(Goal)  
— Description: Goal will be filtered in partial evaluation time according to the PE types defined in the assertion.
8 Properties which are native to analyzers

Author(s): Francisco Bueno, Manuel Hermenegildo, Pedro López, Edison Mera.

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.

8.1 Usage and interface (native_props)

- Library usage:
  ```prolog
  :- use_module(library(assertions(native_props))).
  
  or also as a package :- use_package(nativeprops).
  
  Note the different names of the library and the package.
  
- Exports:
  - Properties:
    - clique/1,
      clique_1/1, constraint/1, covered/1, covered/2, exception/1, exception/2,
      fails/1, finite_solutions/1, have_choicepoints/1, indep/1, indep/2, is_det/1,
      linear/1, mshare/1, mut-exclusive/1, no_choicepoints/1, no_exception/1, no_exception/2,
      no_signal/1, no_signal/2, non_det/1, nonground/1, not_covered/1,
      not_fails/1, not_mut-exclusive/1, num_solutions/2, solutions/2, possibly_fails/1,
      possibly_nondet/1, relations/2, sideeff_hard/1, sideeff_pure/1,
      sideeff_soft/1, signal/1, signal/2, signals/2, size/2, size/3, size_lb/2, size_o/2,
      size_ub/2, size_metric/3, size_metric/4, steps/2, steps_lb/2, steps_o/2,
      steps_ub/2, tau/1, terminates/1, test_type/2, throws/2, user_output/2.
  
  - Other modules used:
    - System library modules:
      terms_check, terms_vars, sort, lists, streams, file_utils, system.
    - Internal (engine) modules:
      term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
      data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
      system_info, term_compare, term_typing, hiord_rt, debugger_support,
      internals.

8.2 Documentation on exports (native_props)

clique/1: PROPERTY

clique(X)

X is a set of variables of interest, much the same as a sharing group but X represents all the sharing groups in the powerset of those variables. Similar to a sharing group, a clique is often translated to ground/1, indep/1, and indep/2 properties.

Usage: clique(X)

- Description: The clique pattern is X.

- The following properties should hold globally:

  This predicate is understood natively by CiaoPP as clique(X). (native/2)
clique_1/1:

PROPERTY

clique_1(X)

X is a set of variables of interest, much the same as a sharing group but X represents all the sharing groups in the powerset of those variables but disregarding the singletons. Similar to a sharing group, a clique_1 is often translated to ground/1, indep/1, and indep/2 properties.

Usage: clique_1(X)

- Description: The 1-clique pattern is X.

- The following properties should hold globally:
  This predicate is understood natively by CiaoPP as clique_1(X). (native/2)

constraint/1:

PROPERTY

constraint(C)

C contains a list of linear (in)equalities that relate variables and int values. For example, [A < B + 4] is a constraint while [A < BC + 4] or [A = 3.4, B >= C] are not.

Usage: constraint(C)

- Description: C is a list of linear equations

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

covered/1:

PROPERTY

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.

covered/2:

PROPERTY

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. (native/1)

exception/1:

PROPERTY

Meta-predicate with arguments: exception(goal).

Usage: exception(Goal)

- Description: Calls of the form Goal throw an exception.
Chapter 8: Properties which are native to analyzers

exception/2: PROPERTY

Meta-predicate with arguments: exception(goal,?).
Usage: exception(Goal,E)
  — Description: Calls of the form Goal throw an exception that unifies with E.

fails/1: PROPERTY

fails(X)
Calls of the form X fail.
Meta-predicate with arguments: fails(goal).
Usage: fails(X)
  — Description: Calls of the form X fail.
  — The following properties hold globally:
    This predicate is understood natively by CiaoPP. (native/1)

finite_solutions/1: PROPERTY

finite_solutions(X)
Calls of the form X produce a finite number of solutions [DLGH97].
Meta-predicate with arguments: finite_solutions(goal).
Usage: finite_solutions(X)
  — Description: All the calls of the form X have a finite number of solutions.

have_choicepoints/1: PROPERTY

Meta-predicate with arguments: have_choicepoints(goal).
Usage: have_choicepoints(X)
  — Description: A call to X creates choicepoints.

indep/1: PROPERTY

Usage: indep(X)
  — Description: The variables in pairs in X are pairwise independent.
  — The following properties hold globally:
    This predicate is understood natively by CiaoPP as indep(X). (native/2)

indep/2: PROPERTY

Usage: indep(X,Y)
  — Description: X and Y do not have variables in common.
  — The following properties hold globally:
    This predicate is understood natively by CiaoPP as indep([X,Y]). (native/2)
**is_det/1:**

```prolog
is_det(X)
```

**PROPERTY**

All calls of the form \( X \) are deterministic, i.e., produce at most one solution, or do not terminate. In other words, if \( X \) succeeds, it can only succeed once. It can still leave choice points after its execution, but when backtracking into these, it can only fail or go into an infinite loop.

*Meta-predicate* with arguments: `is_det(goal)`.

**Usage:** `is_det(X)`

- **Description:** All calls of the form \( X \) are deterministic.

**linear/1:**

```prolog
linear(X)
```

**PROPERTY**

\( X \) 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. (native/1)

**mshare/1:**

```prolog
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)`. (native/2)

**mut_exclusive/1:**

```prolog
mut_exclusive(X)
```

**PROPERTY**

For any call of the form \( X \) at most one clause succeeds, i.e., clauses are pairwise exclusive.

*Meta-predicate* with arguments: `mut_exclusive(goal)`.

**Usage:** `mut_exclusive(X)`

- **Description:** For any call of the form \( X \) at most one clause succeeds.

**no_choicepoints/1:**

*Meta-predicate* with arguments: `no_choicepoints(goal)`.

**Usage:** `no_choicepoints(X)`

- **Description:** A call to \( X \) does not create choicepoints.
no_exception/1:  
Meta-predicate with arguments: no_exception(goal).
Usage: no_exception(Goal)
— Description: Calls of the form Goal do not throw any exception.

no_exception/2:  
Meta-predicate with arguments: no_exception(goal,?).
Usage: no_exception(Goal,E)
— Description: Calls of the form Goal do not throw exception E.

no_signal/1:  
Meta-predicate with arguments: no_signal(goal).
Usage: no_signal(Goal)
— Description: Calls of the form Goal do not send any signal.

no_signal/2:  
Meta-predicate with arguments: no_signal(goal,?).
Usage: no_signal(Goal,E)
— Description: Calls of the form Goal do not send the signal E.

non_det/1:  
non_det(X)
All calls of the form X are non-deterministic, i.e., produce several solutions.
Meta-predicate with arguments: non_det(goal).
Usage: non_det(X)
— Description: All calls of the form X are non-deterministic.

nonground/1:  
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). (native/2)

not_covered/1:  
not_covered(X)
There is some call of the form X for which there is no clause whose test succeeds [DLGH97].
Usage: not_covered(X)
— Description: Not all of the calls of the form X are covered.
not_fails/1:  PROPERTY

not_fails(X)
Calls of the form X produce at least one solution, or do not terminate [DLGH97].

Meta-predicate with arguments: not_fails(goal).

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. (native/1)

not_mut_exclusive/1:  PROPERTY

not_mut_exclusive(X)
For calls of the form X more than one clause may succeed. I.e., clauses are not disjoint for
some call.

Meta-predicate with arguments: not_mut_exclusive(goal).

Usage: not_mut_exclusive(X)
- Description: For some calls of the form X more than one clause may succeed.

num_solutions/2:  PROPERTY

Usage 1: num_solutions(X,N)
- Description: All the calls of the form X have N solutions.
- If the following properties should hold at call time:
  X is a term which represents a goal, i.e., an atom or a structure. (callable/1)
  N is an integer. (int/1)

Usage 2: num_solutions(Goal,Check)
- Description: For a call to Goal, Check(X) succeeds, where X is the number of solu­
tions.
- If the following properties should hold at call time:
  Goal is a term which represents a goal, i.e., an atom or a structure. (callable/1)
  Check is a term which represents a goal, i.e., an atom or a structure. (callable/1)

solutions/2:  PROPERTY

Usage: solutions(Goal,Sols)
- Description: Goal Goal produces the solutions listed in Sols.
- If the following properties should hold at call time:
  Goal is a term which represents a goal, i.e., an atom or a structure. (callable/1)
  Sols is a list. (list/1)

possibly_fails/1:  PROPERTY

possibly_fails(X)
Non-failure is not ensured for any call of the form X [DLGH97]. In other words, nothing

Meta-predicate with arguments: possibly_fails(goal).

Usage: possibly_fails(X)
- Description: Non-failure is not ensured for calls of the form X.
possibly_nondet/1:

```
possibly_nondet(X)
```

PROPERTY
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.

relations/2:

```
relations(X,N)
```

PROPERTY
The goal X produces N solutions. In other words, N is the cardinality of the solution set of X.
Meta-predicate with arguments: relations(goal,?).
Usage: relations(X,N)
— Description: Goal X produces N solutions.

sideff_hard/1:

```
sideff_hard(X)
```

PROPERTY
Meta-predicate with arguments: sideff_hard(goal).
Usage: sideff_hard(X)
— Description: X has hard side-effects, i.e., those that might affect program execution (e.g., assert/retract).

sideff_pure/1:

```
sideff_pure(X)
```

PROPERTY
Meta-predicate with arguments: sideff_pure(goal).
Usage: sideff_pure(X)
— Description: X is pure, i.e., has no side-effects.

sideff_soft/1:

```
sideff_soft(X)
```

PROPERTY
Meta-predicate with arguments: sideff_soft(goal).
Usage: sideff_soft(X)
— Description: X has soft side-effects, i.e., those not affecting program execution (e.g., input/output).

signal/1:

```
signal(Goal)
```

PROPERTY
Meta-predicate with arguments: signal(goal).
Usage: signal(Goal)
— Description: Calls of the form Goal throw a signal.

signal/2:

```
signal(Goal,E)
```

PROPERTY
Meta-predicate with arguments: signal(goal,?).
Usage: signal(Goal,E)
— Description: A call to Goal sends a signal that unifies with E.
The lpdoc Documentation Generator

**signals/2**: Property
Meta-predicate with arguments: signals(goal,?).
Usage: signals(Goal,Es)
  
  Description: Calls of the form Goal can generate only the signals that unify with the terms listed in Es.

**size/2**: Property
Usage: size(X,Y)
  
  Description: Y is the size of argument X, for any approximation.

**size/3**: Property
Usage: size(A,X,Y)
  
  Description: Y is the size of argument X, for the approximation A.

**size_lb/2**: Property
Usage: size_lb(X,Y)
  
  Description: Y is a lower bound on the size of argument X.

**size_o/2**: Property
Usage: size_o(X,Y)
  
  Description: The size of argument X is in the order of Y.

**size_ub/2**: Property
Usage: size_ub(X,Y)
  
  Description: Y is an upper bound on the size of argument X.

**size_metric/3**: Property
Meta-predicate with arguments: size_metric(goal,?,?).
Usage: size_metric(Head,Var,Metric)
  
  Description: Metric is the metric of the variable Var, for any approximation.
size_metric/4: PROPERTY

Meta-predicate with arguments: size_metric(goal,?,?,?).
Usage: size_metric(Head,Approx,Var,Metric)

- Description: Metric is the metric of the variable Var, for the approximation Approx.
  Currently, Metric can be: int/1, size/1, length/1, depth/2, and void/1.

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
Meta-predicate with arguments: steps(goal,?).
Usage: steps(X,Y)

- Description: Y is the cost (number of resolution steps) of any call of the form 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]
Meta-predicate with arguments: steps_lb(goal,?).
Usage: steps_lb(X,Y)

- Description: Y is a lower bound on the cost of any call of the form X.

steps_o/2: PROPERTY

steps_o(X,Y)
Meta-predicate with arguments: steps_o(goal,?).
Usage: steps_o(X,Y)

- Description: Y is the complexity order of 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].
Meta-predicate with arguments: steps_ub(goal,?).
Usage: steps_ub(X,Y)

- Description: Y is a upper bound on the cost of any call of the form X.

tau/1: PROPERTY

tau(Types)
Types contains a list with the type associations for each variable, in the form V/[T1,...,TN]. Note that tau is used in object-oriented programs only
Usage: tau(TypeInfo)

- Description: Types is a list of associations between variables and list of types
- The following properties hold globally:
  This predicate is understood natively by CiaoPP. (native/1)
terminates/1:
    terminates(X)
    Calls of the form X always terminate [DLGH97].
    Meta-predicate with arguments: terminates(goal).
    Usage: terminates(X)
    — Description: All calls of the form X terminate.

test_type/2:                      PROPERTY
    Meta-predicate with arguments: test_type(goal,?).
    Usage: test_type(X,T)
    — Description: Indicates the type of test that a predicate performs. Required by the nonfailure analysis.

throws/2:                         PROPERTY
    Meta-predicate with arguments: throws(goal,?).
    Usage: throws(Goal,Es)
    — Description: Calls of the form Goal can throw only the exceptions that unify with the terms listed in Es.

user_output/2:                    PROPERTY
    Meta-predicate with arguments: user_output(goal,?).
    Usage: user_output(Goal,S)
    — Description: Calls of the form Goal write S to standard output.

instance/2:                       PROPERTY
    Usage: instance(Term1,Term2)
    — Description: Term1 is an instance of Term2.
    — The following properties hold globally:
        This predicate is understood natively by CiaoPP. (native/1)
9 Meta-properties

Author(s): Francisco Bueno.

This library allows the use of some meta-constructs which provide for specifying properties of terms which are unknown at the time of the specification, or expressed with a shorthand for the property definition, i.e., without really defining it.

An example of such use is an assertion which specifies that any property holding upon call will also hold upon exit:

\[- \text{pred } p(X) : \text{Prop}(X) \Rightarrow \text{Prop}(X).\]

Another example is using shorthands for properties when documenting:

\[- \text{pred } p(X) : \text{regtype}(X, (\text{(list;list)}; \text{list})).\]

(See below for an explanation of such a regular type.)

9.1 Usage and interface (meta_props)

- Library usage:
  \[- \text{use}_\text{module}(\text{library(} \text{assertions(meta_props)})\)]
  or also as a package: \[- \text{use}_\text{package(metaprops)}.\]
  Note the different names of the library and the package.

- Exports:
  - Properties:
    call/2, prop/2, regtype/2.
  - Multifiles:
    callme/2.

- Other modules used:
  - Internal (engine) modules:
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.

9.2 Documentation on exports (meta_props)

call/2: PROPERTY

call(P,A)
A has property P (provided that P is a property). Equivalent to P(A).
Usage: call(P,A)
- Description: A has property P.
- If the following properties hold at call time:
  P is a term which represents a goal, i.e., an atom or a structure. (callable/1)
prop/2:
Usage: A prop P
  — Description: A has property P.
  — If the following properties hold at call time:
    P has property " (callable;prop_abs).

regtype/2:
Usage: A regtype T
  — Description: A is of type T.
  — If the following properties hold at call time:
    T has property " ((regtype);prop_abs).

9.3 Documentation on multifiles (meta_props)

callme/2:
(User defined.) A hook predicate you have to define as callme(P,X):- P(X), !. in the
program that uses this library. This is done automatically if the package is used instead
of the library module (but then you should not define callme/2 in your program).
The predicate is multifile.
Usage: callme(A,B)
  — The following properties should hold at call time:
    A is a term which represents a goal, i.e., an atom or a structure.

9.4 Documentation on internals (meta_props)

prop_abs/l:
Usage: prop_abs(Prop)
  — Description: Prop is a property abstraction, i.e., a parametric property, or a term formed of property
abstractions, where the functors used in the term are escaped by ".
One particular case of property abstractions are parametric regular type abstractions, i.e.,
a parametric type functor or a "-escaped term formed of regular type abstractions.
Such abstractions are a short-hand for a corresponding regular type (correspondingly,
property). For example, the following abstraction:

"(list;list);list

denotes terms of the form (X;Y) where list(X) and list(Y) hold and also terms T such
that list(T) holds. It is equivalent to the regular type:

abstract_type((X;Y)):- list(X), list(Y).
abstract_type(T):- list(T).

Usage: prop_abs(Prop)
  — Description: Prop is a property abstraction.
10 An Example - Documenting a Library Module

Author(s): Manuel Hermenegildo.

A simple example of the use of Ipdoc is this manual, which can be built in the doc directory of the Ipdoc distribution. Other examples of manuals generated using Ipdoc can be found in the Ciao system and preprocessor doc directories (i.e., most of the Ciao manuals are generated using Ipdoc). Some simpler examples can be found in the examples directory of the Ipdoc distribution. In particular, the chapter following this one contains the documentation generated automatically for the module defined by file examples/example_module.pl (which for simplicity contains only assertions, i.e., no actual code) and which is included in source form below. Comparing this code with the output in the following chapter illustrates the use and some of the capabilities of Ipdoc:

%% The module headers produce documentation on the module interface
:- module(example_module,
   [bar/1,baz/1,aorb/1,tree_of/2,list_or_aorb/2,q/2,r/1, p/1, p/5, u/3, 
    long/1, w/1, mytype/1, t/5, s/1, q/1],
   [assertions,basicmodes,fsyntax,regtypes,hiord,nativeprops]).

%% Exported predicates (+ properties and types) are documented by default
:- module(example_module,
   [bar/1,baz/1,aorb/1,tree_of/2,list_or_aorb/2,q/2,r/1, p/1, p/5, u/3, 
    long/1, w/1, mytype/1, t/5, s/1, q/1],
   [assertions,basicmodes,fsyntax,regtypes,hiord,nativeprops]).

%% We import two types: list/1 and list/2 (now in basic_props, which is
%% exported by default from assertions).
:- reexport(library(engine(basic_props)), [list/1]).

%% We reexport list/1
:- use_module(library(lists), [length/2]).
:- use_module(bar).
:- ensure_loaded(foo).

%% "doc" declarations provide additional information
:- doc(title,"Auto Documenter Output for the Example Module").

:- doc(author,"Anonymous Author 1").
:- doc(author,"Anonymous Author 2").

:- doc(summary,"This is a brief summary description of the module
or file. In this case the file is a library.").

:- doc(module,"This is where general comments on the file go. In
this case the file is a library which contains some assertion examples
for testing the @em{automatic documentation system}. ").

%% An example of a comment documenting a bug
:- doc(bug,"Library is hard to execute: no actual code!").

%% Standard declarations are documented with the corresponding predicate
:- data r/1.
:- dynamic q/2.
:- multifile p/3.
:- dynamic p/3.
:- meta_predicate p(?,:,:),?).
Uncommenting this would make these not appear in the documentation
:- doc(hide,[bar/1,baz/1]).

This is a type definition in Prolog syntax: declaration and code:
:- true regtype bar(X) # "@var{X} is an acceptable kind of bar.".
bar(night).
bar(day).

This is another type definition in Prolog syntax, with no comment.
:- true regtype baz/1.
baz(a).
baz(b).

Two type definitions in 'typedef' syntax (will be expanded to code as above):
:- typedef aorb : := ~a;~b.
:- typedef listof_or_aorb(X) : := list(X);aorb.

Using functional notation:
:- regtype aorb/1.
aorb := a.
aorb := b.

Should use the other function syntax which uses *first argument* for return:
:- regtype tree_of/2.
tree_of(_) := void.
tree_of(T) := tree("call(T),~tree_of(T),~tree_of(T)).

tree_of(_, void).
tree_of(T, tree(X,L,R)) :-
  T(X),
tree_of(T,L),
tree_of(T,R).

:- regtype list_or_aorb/2.
list_or_aorb(T) := ~list(T).
list_or_aorb(_T) := ~aorb.

This is a property definition
This comment appears only in the place where the property itself
is documented.
:- doc(long/1,"This is a property, describing a list that is longish. The definition is:

@includedef{long/l}
Chapter 10: An Example - Documenting a Library Module

7.7

% The comment here will be used to document any predicate which has an assertion which uses the property:

\[-\text{prop long(L)} \# @\text{var}\{L\} \text{ is rather long.}\].

long(L) :-
    length(L,N),
    N>100.

% Now, a series of assertions:

% This declares the entry mode of this exported predicate (i.e., how it is called from outside):

\[-\text{entry p/3 : gnd * var * var}.\]

% This describes all the calls:

\[-\text{calls p/3 : foo * bar * baz}.\]

foo(_).

% This describes the successes (for a given type of calls):

\[-\text{success p/3 : int * int * var =\to int * int * gnd}.\]

% This describes a global property (for a given type of calls):

\[-\text{comp p/3 : int * int * var + not_fails}.\]

\[-\text{doc(p/3,"A @bf{general comment} on the predicate." )}.\]

% Documenting some typical usages of the predicate:

\[-\text{pred p/3 : int * int * var}\]
\[\Rightarrow \text{int * int * list}\]
\[+\ (\text{iso,not_fails})\]
\[\# "\text{This mode is nice."}].\]
\[-\text{pred p(Preds,Value,Assoc)} : \text{var * var * list}\]
\[\Rightarrow \text{int * int * list}\]
\[+\ \text{not_fails} \ # "\text{This mode is also nice."}.\]
\[-\text{pred p/3}\]
\[\Rightarrow \text{list * int * list}\]
\[+\ (\text{not_fails,not_fails})\]
\[\# "\text{Just playing around."}.\]

\[-\text{pred q(A)} : \text{list(A)}\]
\[\Rightarrow \ (\text{list(A),gnd(A)})\]
\[+\ \text{not_fails}\]
\[\# "\text{Foo}".\]
\[-\text{pred q(A)}\]
\[\# "\text{Not a bad use at all."}.\]
:- pred q/2
  : var * {gnd,int}
  => {gnd,int} * int.
:- pred q/2
  :: int * list
  # "Non-moded types are best used this way.".
q(_).
:- pred p/1 : var => list.
p(_).

:- pred r(A)
  : list(A)
  => (list(A,int),gnd(A))
  + not_fails
  # "This uses parametric types".

:- doc(doinclude,s/1). %% Forces documentation even if not exported
:- pred s(A)
  : list(A)
  => (list(A),gnd(A))
  + not_fails.
s(_).

:- doc(doinclude,[list/2,list/1]). %% Forces (local) documentation even if not exported

:- mode def og(A)
  => gnd(A)
  # "This is a @em{mode} definition: the output is ground.".

:- doc(doinclude,og/2).

:- mode def og(A,T)
  :: T(A)
  => gnd(A)
  # "This is a @em{parametric mode definition}".

:- pred t(+A,-B,?C,?D,og(E))
  :: list * list * int * int * list
  : long(B)
  => (gnd(C),gnd(A))
  + not_fails
  # "This predicate uses @em{modes} extensively".
t(_, _, _, _, _).

%% Some other miscellaneous assertions:
Check is default assertion status anyway...
:- check pred u(+,-,og).
:- check pred u(int,list(mytype),int).

u(_, _, _).

"true" status is normally compiler output
:- true pred w(+list(mytype)).

mytype(_).
w(_).

:- doc(doinclude,is/2).

:- trust pred is(Num,Expr) : arithexpression(Expr) => num(Num)
   # "Typical way to describe/document an external predicate (e.g.,
    written in C).".

:- doc(doinclude,p/5).
:- pred p(og(int),in,@list(int),-,+A) + steps_lb(1+length(A)).

p(_, _, _, _, _) :- _ is 1.

Version information. The ciao.el emacs mode allows automatic maintenance
11 Auto Documenter Output for the Example Module

Author(s): Anonymous Author 1, Anonymous Author 2.

This is where general comments on the file go. In this case the file is a library which contains some assertion examples for testing the automatic documentation system.

11.1 Usage and interface (example_module)

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

- Exports:
  - Predicates:
    - q/2, r/1, p/1, p/5, u/3, w/1, mytype/1, t/5, s/1, q/1.
  - Properties:
    - long/1.
  - Regular Types:
    - bar/1, baz/1, aorb/1, tree_of/2, list_or_aorb/2.
  - Multifiles:
    - p/3.

- Other modules used:
  - Files of module user:
    - foo.
  - System library modules:
    - assertions/native_props, engine/basic_props, lists.
  - Internal (engine) modules:
    - term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

11.2 Documentation on exports (example_module)

bar/1:  
Usage: bar(X)  
- Description: X is an acceptable kind of bar.

baz/1:  
A regular type, defined as follows:

  baz(a).
  baz(b).
aorb/1:  
A regular type, defined as follows:
  aorb(a).
aorb(b).

REGTYPE

tree_of/2:  
A regular type, defined as follows:
  tree_of(_1,void).
  tree_of(T,tree(_1,_2,_3)) :-
      call(T,_1),
      tree_of(T,_2),
      tree_of(T,_3).

REGTYPE

list_or_aorb/2:  
A regular type, defined as follows:
  list_or_aorb(T,_1) :-
      list(T,_1).
  list_or_aorb(_T,_1) :-
      aorb(_1).

REGTYPE

q/2:  
The predicate is of type dynamic.
Usage 1:
  - *The following properties should hold at call time:*
    Arg1 is a free variable.  (var/1)
    Arg2 is ground.  (gnd/1)
    Arg2 is an integer.  (int/1)
  - *The following properties should hold upon exit:*
    Arg1 is ground.  (gnd/1)
    Arg1 is an integer.  (int/1)
    Arg2 is an integer.  (int/1)

Usage 2:
  - *Description: Non-moded types are best used this way.*
  - *Call and exit should be compatible with:*
    Arg1 is an integer.  (int/1)
    Arg2 is a list.  (list/1)

PREDICATE

r/1:  
The predicate is of type data.
Usage: r(A)
  - *Description: This uses parametric types*
Chapter 11: Auto Documenter Output for the Example Module

The following properties should hold at call time:
A is a list.

The following properties should hold upon exit:
A is a list of ints.
A is ground.

The following properties should hold globally:
All the calls of the form \( r(A) \) do not fail.

p/1:
Usage:

– The following properties should hold at call time:
  Arg1 is a free variable.

– The following properties should hold upon exit:
  Arg1 is a list.

p/5:
Usage: \( p(\text{Arg1}, \text{Arg2}, \text{Arg3}, \text{Arg4}, A) \)

– Call and exit should be compatible with:
  Arg1 is an integer.
  Arg3 is a list of ints.

– The following properties should hold at call time:
  Arg2 is currently ground (it contains no variables).
  Arg4 is a free variable.
  A is currently a term which is not a free variable.

– The following properties should hold upon exit:
  Arg1 is ground.
  Arg2 is currently ground (it contains no variables).

– The following properties should hold globally:
  Arg3 is not further instantiated.
  \( 1 + \text{length}(A) \) is a lower bound on the cost of any call of the form \( p(\text{Arg1}, \text{Arg2}, \text{Arg3}, \text{Arg4}, A) \).

u/3:
Usage 1:

– The following properties should hold at call time:
  Arg1 is currently a term which is not a free variable.
  Arg2 is a free variable.

– The following properties should hold upon exit:
  Arg3 is ground.
long/1:
This is a property, describing a list that is longish. The definition is:
\[
\text{long}(L) :-
\text{length}(L,N),
N > 100.
\]
Usage: long(L)
— Description: L is rather long.

w/1:
Usage:
— Calls should, and exit will be compatible with:
Arg1 is a list of mytypes.
— The following properties should hold at call time:
Arg1 is currently a term which is not a free variable.

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

t/5:
Usage: t(A,B,C,D,E)
— Description: This predicate uses modes extensively.
— Call and exit should be compatible with:
A is a list.
B is a list.
C is an integer.
D is an integer.
E is a list.
— The following properties should hold at call time:
A is currently a term which is not a free variable.
B is a free variable.
B is rather long.
— The following properties should hold upon exit:
E is ground.
C is ground.
A is ground.
— The following properties should hold globally:
D is not further instantiated.
All the calls of the form t(A,B,C,D,E) do not fail.
Chapter 11: Auto Documenter Output for the Example Module

s/1:  
Usage: s(A)  
— The following properties should hold at call time:  
  A is a list.  
— The following properties should hold upon exit:  
  A is a list.  
  A is ground.  
— The following properties should hold globally:  
  All the calls of the form s(A) do not fail.

q/1:  
Usage 1: q(A)  
  Description: Foo  
— The following properties should hold at call time:  
  A is a list.  
— The following properties should hold upon exit:  
  A is a list.  
  A is ground.  
— The following properties should hold globally:  
  All the calls of the form q(A) do not fail.
Usage 2: q(A)  
  Description: Not a bad use at all.

list/1:  
General properties: list(L)  
— The following properties hold globally:  
  list(L) is side-effect free.
list(L)  
— If the following properties hold at call time:  
  L is currently ground (it contains no variables).  
  then the following properties hold globally:  
  list(L) is evaluable at compile-time.  
  All calls of the form list(L) are deterministic.
list(T)  
— The following properties hold upon exit:  
  T is a list.
11.3 Documentation on multifiles (example_module)

p/3:

A general comment on the predicate.
The predicate is multifile.
The predicate is of type dynamic.

General properties:

- If the following properties hold at call time:
  Arg1 is ground. (gnd/1)
  Arg2 is a free variable. (var/1)
  Arg3 is a free variable. (var/1)

- The following properties should hold at call time:
  foo(Arg1) (undefined property)
  Arg2 is an acceptable kind of bar. (bar/1)
  baz(Arg3) (baz/1)

- If the following properties hold at call time:
  Arg1 is an integer. (int/1)
  Arg2 is an integer. (int/1)
  Arg3 is a free variable. (var/1)
  then the following properties should hold upon exit:
  Arg1 is an integer. (int/1)
  Arg2 is an integer. (int/1)
  Arg3 is ground. (gnd/1)

- If the following properties hold at call time:
  Arg1 is an integer. (int/1)
  Arg2 is an integer. (int/1)
  Arg3 is a free variable. (var/1)
  then the following properties should hold globally:
  All the calls of the form \( p(\text{Arg1}, \text{Arg2}, \text{Arg3}) \) do not fail. (not_fails/1)

Usage 1:

- Description: This mode is nice.

- The following properties should hold at call time:
  Arg1 is an integer. (int/1)
  Arg2 is an integer. (int/1)
  Arg3 is a free variable. (var/1)

- The following properties should hold upon exit:
  Arg1 is an integer. (int/1)
  Arg2 is an integer. (int/1)
  Arg3 is a list. (list/1)

- The following properties should hold globally:
  Complies with the ISO-Prolog standard. (iso/1)
  All the calls of the form \( p(\text{Arg1}, \text{Arg2}, \text{Arg3}) \) do not fail. (not_fails/1)

Usage 2: \( p(\text{Preds}, \text{Value}, \text{Assoc}) \)
— *Description:* This mode is also nice.

— The following properties should hold at call time:
  
  **Preds** is a free variable.  
  **Value** is a free variable.  
  **Assoc** is a list.

— The following properties should hold upon exit:
  
  **Preds** is an integer.  
  **Value** is an integer.  
  **Assoc** is a list.

— The following properties should hold globally:
  
  All the calls of the form \( p(\text{Preds}, \text{Value}, \text{Assoc}) \) do not fail.

Usage 3:

— *Description:* Just playing around.

— The following properties should hold upon exit:
  
  **Arg1** is a list.  
  **Arg2** is an integer.  
  **Arg3** is a list.

— The following properties should hold globally:
  
  All the calls of the form \( p(\text{Arg1}, \text{Arg2}, \text{Arg3}) \) do not fail.

11.4 **Documentation on internals** (example_module)

**list/2:**

General properties: \( \text{list}(L,T) \)

— The following properties hold globally:
  
  \( \text{list}(L,T) \) is side-effect free.

**list(L,T):**

— If the following properties hold at call time:
  
  \( L \) is currently ground (it contains no variables).
  \( T \) is currently ground (it contains no variables).

  then the following properties hold globally:

  \( \text{list}(L,T) \) is evaluable at compile-time.

**list(X,T):**

— The following properties hold upon exit:
  
  \( X \) is a list.

Usage: \( \text{list}(L,T) \)

— Description: \( L \) is a list of Ts.
og/2:
Usage: og(A,T)

Description: This is a parametric mode definition.

Call and exit are compatible with:

- call(T,A)

(undefined property)

The following properties are added upon exit:

A is ground.

MODE

is/2:
Usage: Num is Expr

Description: Typical way to describe/document an external predicate (e.g., written in C).

The following properties should hold at call time:

- Expr is an arithmetic expression.

(arithexpression/1)

The following properties hold upon exit:

- Num is a number.

(num/1)

PREDECATE
12 Run-time checking of assertions

Author(s): Edison Mera.

This package provides a complete implementation of run-time checks of predicate assertions. The program is instrumented to check such assertions at run time, and in case a property does not hold, the error is reported. Note that there is also an older package called rtchecks, by David Trallero. The advantage of this one is that it can be used independently of CiaoPP and also has updated functionality.

There are two main applications of run-time checks:

• To improve debugging of certain predicates, specifying some expected behavior that is checked at run-time with the assertions.

• To avoid manual implementation of run-time checks that should be done in some predicates, leaving the code clean and understandable.

The run-time checks can be configured using prolog flags. Below we itemize the valid prolog flags with its values and a brief explanation of the meaning:

• rtchecks_level
  • exports: Only use rtchecks for external calls of the exported predicates.
  • inner: Use also rtchecks for internal calls. Default.

• rtchecks_trust
  • no: Disable rtchecks for trust assertions.
  • yes: Enable rtchecks for trust assertions. Default.

• rtchecks_entry
  • no: Disable rtchecks for entry assertions.
  • yes: Enable rtchecks for entry assertions. Default.

• rtchecks_exit
  • no: Disable rtchecks for exit assertions.
  • yes: Enable rtchecks for exit assertions. Default.

• rtchecks_test
  • no: Disable rtchecks for test assertions. Default.
  • yes: Enable rtchecks for test assertions. Used for debugging purposes, but is better to use the unittest library.

• rtchecks_inline
  • no: Instrument rtchecks using call to library predicates present in rtchecks_rt.pl, nativeprops.pl and basic_props.pl. In this way, space is saved, but sacrificing performance due to usage of meta calls and external methods in the libraries. Default.
  • yes: Expand library predicates inline as far as possible. In this way, the code is faster, because its avoids metacalls and usage of external methods, but the final executable could be bigger.

• rtchecks_asrloc Controls the usage of locators for the assertions in the error messages. The locator says the file and lines that contains the assertion that had failed. Valid values are:
  • no: Disabled.
  • yes: Enabled. Default.

• rtchecks_predloc Controls the usage of locators for the predicate that caused the run-time check error. The locator says the first clause of the predicate that the violated assertion refers to.
  • no: Disabled.
- yes: Enabled, Default.
- rtchecks_callloc
  - no: Do not show the stack of predicates that caused the failure
  - predicate: Show the stack of predicates that caused the failure. Instrument it in the predicate. Default.
  - literal: Show the stack of predicates that caused the failure. Instrument it in the literal. This mode provides more information, because reports also the literal in the body of the predicate.
- rtchecks_namefmt
  - long: Show the name of predicates, properties and the values of the variables
  - short: Only show the name of the predicate in a reduced format. Default.

12.1 Usage and interface (rtchecks_doc)

- Library usage:
  :- use_package(rtchecks).
  or
  :- module(...,...,[rtchecks]).
- Other modules used:
  - Internal (engine) modules:
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
13 Unit Testing Library

Author(s): Edison Mera.

This library provides an extension of the Ciao assertion language which allows writing unit tests. The central idea is to use the assertion language to provide specifications of test cases for a given predicate. The package also provides some special properties that are convenient when specifying unit tests and the required run-time libraries.

In general, a test assertion is written as follows:

```
:- test predicate(A1, A2, ..., An)  
  : <Precondition>  
  => <Postcondition>  
  + <Global properties>  
  # <Comment>.
```

Where the fields of the test assertion have the usual meaning in Ciao assertions, i.e., they contain conjunctions of properties which must hold at certain points in the execution. Here we give a somewhat more operational ("test oriented"), reading to these fields: `predicate/n` is the predicate to be tested. `Precondition` is a goal that is called before the predicate being tested, and can be used to generate values of the input parameters. `Postcondition` is a goal that should succeed after `predicate/n` has been called. The idea appears to be simple, but note that due to the non-determinism of logic programs, the test engine needs to test all the solutions that can be tested up to given limits (for example, a maximum number of solutions, or a given time-out). `Properties` specifies some global properties that the predicate should meet, for example, `not_fails` means that the program does not fail, `exception(error(a,b))` means that the program should throw the exception `error(a,b)`, and so on. But there are some specific properties that only applies to testing specified in the module `unittest_props.pl`, for example `times(N)` specifies that the given test should be executed N times, `try_sols(N)` specifies that the first N solutions of the predicate `predicate/n` are tested. `Comment` is a string that document the test.

A convenient way to run these tests is by selecting options in the CiaoDbg menu within the development environment:

1. Run tests in current module: execute only the tests specified in the current module.
2. Run tests in all related modules: execute the tests specified in the module and in all the modules being used by this.
3. Show untested predicates: show the exported predicates that do not have any test assertion.

13.1 Additional notes

1. The test assertions allow performing unit testing, i.e., in Ciao, performing tests at the predicate level.
2. The tests currently can only be applied to exported predicates.
3. If you need to write tests for predicates that are spread over several modules, but work together, then it is best to create a separate module, and reexport to the predicates required to build the test. This allows performing integration testing, using the same syntax of the unit tests.
4. The Ciao system includes a good (and growing) number of unit tests. To run all the tests among the other standard tests within the CiaoDE run the following (at the top level of the source tree):

```
./ciaosetup runtests
```
13.2 Usage and interface (unittest_doc)

- **Library usage:**
  
  ```prolog
  :- use_module(library(unittest)).
  ```

- **Other modules used:**
  - *Internal (engine) modules:*
    
    - term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
Chapter 14: Installing lpdoc

14 Installing lpdoc

Author(s): Manuel Hermenegildo.

This documentation is outdated.

The source distribution contains all the source code and libraries and can be compiled on a supported Prolog system (lpdoc is developed using Ciao Prolog). The latest publically distributed version of lpdoc is available from http://www.clip.dia.fi.upm.es/Software/Ciao. A newer version in Beta test is often available in http://www.clip.dia.fi.upm.es/Software/Beta/Ciao.

14.1 Installing the source distribution (lpdoc)

• Before installing lpdoc, you may want to read Section 14.2 [Other software packages required (lpdoc)], page 107. Make sure that emacs is installed in your system
• Uncompress (using gunzip) and unpackage (using tar -xpf) the distribution in a suitable directory. This will create a new directory called lpdoc as well as a link lpdoc-X.Y to this directory, 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.
• Enter the newly created directory and if needed edit the file LPDOCSETTINGS.pl in a text editor, but in general the default options works well (edit the one in that directory, not the ones in the subdirectories).
  • Decide which Prolog/CLP system you will use for compiling lpdoc (actually, currently only Ciao is supported – but porting to, e.g., SICStus Prolog should not be too difficult) and modify the first part of the LPDOCSETTINGS.pl file accordingly. The DOCDIR directory should not be an existing info directory, since this will overwrite the dir file in that directory.
  • Select the directories in which you want the lpdoc binaries, libraries, and documents installed, by setting the corresponding variables in LPDOCSETTINGS.pl.
• Type lpmake all. This should create the lpdoc executable and compile related libraries.
• Type lpmake install. This should install it in the BINDIR directory, install the lpdoc library in a separate directory in the LIBDIR directory, and install the lpdoc documentation in the DOCDIR directory.
• In order for the lpdoc documentation to be available to users automatically, certain environment variables have to be set. The installation leaves files suitable for inclusion in initialization scripts (e.g., DOTcshrc for csh) in the lpdoc library.

14.2 Other software packages required (lpdoc)

The most basic functionality of lpdoc (generating manuals in .texi format, short manual entries in .man format, generating index files) is essentially self contained. However, using the full capabilities of lpdoc requires having several other software packages installed in the system. Fortunately, all of these packages are public domain software and they will normally be already installed in, e.g., a standard Linux distribution. It should be relatively easy to get and install the required packages in other Unix-like packages or even in Windows, under the Cygwin environment.
  • Basic requirements: the Makefiles used by lpdoc require GNU Make (gmake), and for now have only been designed for UN*X-like operating systems.
• **Generating .dvi files**: `lpdoc` normally generates .texi files (actually, a number of .texic files). From the .texi files, .dvi files are generated using the standard `tex` package directly. The .dvi files can also be generated with the GNU Texinfo package, which provides, among others, the `texi2dvi` command. However, Texinfo itself requires the standard `tex` document processing package. In order to use `texi2dvi` instead of `tex` when processing documents you should change the variable `TEX` in the `Makefile.skel` file in the `lib` directory before installing `lpdoc`. Generating the .dvi file requires that the `texinfo.tex` file (containing the relevant macros) be available to `tex`. This file is normally included with modern `tex` distributions, although it may be obsolete. An appropriate and up-to-date one for `lpdoc` is provided with the `lpdoc` distribution, stored in the `lpdoc` library during installation, and used automatically when `lpdoc` runs `tex`. The `texindex` package is required in order to process the indices. If you use references in your manual, then the `bibtex` package is also needed. `texindex` and `bibtex` are included with most `tex` distributions.

• **Generating .ps files**: .ps files are generated from the .dvi files using the `dvips` command (this, again, can be changed in the `Makefile.skel` file in the `lib` directory). This command is included with standard `tex` distributions.

• **Generating .pdf files**: .pdf files are currently generated from the .texi file using the `pdftex` command (this, again, can be changed in the `Makefile.skel` file in the `lib` directory). This command is included in current Linux distributions.

• **Generating .html files**: .html files are generated directly from the .texi file using the `texi2html` command (this, again, can be changed in the `Makefile.skel` file in the `lib` directory). This command is a perl script and is included with the `lpdoc` distribution, and installed in the library (so that it does not overwrite other existing versions). It is also typically included in the `Texinfo` distribution. A required intermediate step is to resolve the link references which appear in the .texi file (the .texi file includes all the .texic files and has all references resolved). This is done using the `emacs` editor in batch mode, calling functions in the `emacs-library.el` file included in the `lib` directory of the `lpdoc` distribution. Thus, a recent version of `emacs` is required for this purpose.

• **Generating .info files**: .info files are also generated directly from the .texi file using the `makeinfo` command (this, again, can be changed in the `Makefile.skel` file in the `lib` directory). This command is included in the `Texinfo` distribution. Resolving the link references in the .texi file is also required as above.

• If pictures are used in the manual, and html output is selected, the commands `pstogif` and `cjpeg` are also required, in order to convert the figures from .eps to .jpg format.
PART II - LPdoc Internals Manual
15 Documentation Generation Library

Author(s): Manuel Hermenegildo, Jose F. Morales.

This library provides some predicates which generate documentation automatically for a given module or application, using the declarations and assertions used in the module itself as input (see the assertions library). By default, only the exported predicates of the module appear in the documentation. The predicates will be documented in the order in which they appear in the module/1 or module/2 declaration.

The idea of this package is on one hand to reuse the information present in the assertions and on the other to help ensure that code and documentation are kept as coherent as possible. Hopefully, keeping them close together should help in this always difficult task. The resulting documentation is somewhat rigidly structured, but generally sufficient for a reference manual, provided a little effort is put into the assertions and comments. The end product understandably depends heavily on how much work is put into adding additional comments to the source. Some documentation will be generated in any case, but it is recommended that, at the minimum, a module title and a comment for each of the exported predicates be provided.

\textbf{Note: This part is obsolete. – JFMC}

The output format in which the documentation is generated is defined by the backend modules \{\texttt{autodoc\_texinfo}, \texttt{autodoc\_html}, \texttt{autodoc\_man}, etc.\}.

The main output format supported is \texttt{texinfo} (see The GNU Texinfo Documentation System manual for more info), from which printed manuals and several other printing and on-line formats can be easily generated automatically (including info, html, etc.). There is also some limited support for direct output in unix \texttt{man} format and direct \texttt{html} (but note that html can also be generated in a better way by first generating texinfo and then using one of the available converters). For texinfo, the documentation for a module is a texinfo chapter, suitable for inclusion in a wrapper “main” document file. A simple example of the use of this library for generating a texinfo reference manual (including a driver script, useful Makefiles, etc.) is included with the library source code. Other examples can be found in the Ciao documentation directory (i.e., the Ciao manuals themselves).

A simple example of the use of this library for generating a \texttt{texinfo} reference manual (including a driver script, useful Makefiles, etc.) is included with the library source code. Other examples can be found in the Ciao documentation directory (i.e., the Ciao manuals themselves).
15.1 Usage and interface (autodoc)

- **Library usage:**
  ```prolog```
  ```:- use_module(library(autodoc)).```  
- **Exports:**
  - Predicates:
    ```prolog```
    ```index_comment/2, reset_output_dir_db/0, ensure_output_dir_prepared/2,``` 
    ```get_autodoc_opts/3, autodoc_gen_doctree/5, fmt_infodir_entry/3, autodoc_compute_grefs/3, autodoc_translate_doctree/3, autodoc_finish/1, autodoc_gen_alternative/2.``` 
  - Multifiles:
    ```prolog```
    ```autodoc_finish_hook/1, autodoc_gen_alternative_hook/2.``` 
- **Other modules used:**
  - Application modules:
    ```prolog```
    ```lpdocsr(src(autodoc_state)), lpdocsr(src(autodoc_settings)),``` 
    ```lpdocsr(src(autodoc_filesystem)), lpdocsr(src(autodoc_structure)),``` 
    ```lpdocsr(src(autodoc_doctree)), lpdocsr(src(autodoc_refsd)),``` 
    ```lpdocsr(src(autodoc_parse)), lpdocsr(src(autodoc_index)), lpdocsr(src(comments)), lpdocsr(src(autodoc_html_resources)),``` 
    ```lpdocsr(src(autodoc_texpinfo)).``` 
  - System library modules:
    ```prolog```
    ```format, ttyout, aggregates, read,``` 
    ```make/make_rt, dict, compiler/compiler, assertions/assert_lib, compiler/c_itf,``` 
    ```assertions/assertions_props, messages, filenames, lists, terms, make/system_ extra, system.``` 
  - Internal (engine) modules:
    ```prolog```
    ```term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,``` 
    ```data_facts, exceptions, io_aux, io_basic, prolog_flags, streams Basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.``` 

15.2 Documentation on exports (autodoc)

**index_comment/2:**  
**PREDICATE**

**Usage:** `index_comment(Index,Text)`

- **Description:** Type is a type of index which is supported. Text describes the index contents.
- **The following properties should hold upon exit:**
  ```prolog```
  ```Index is currently instantiated to an atom. (atom/1) Text is a string (a list of character codes). (string/1)`

**reset_output_dir_db/0:**  
**PREDICATE**

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

Usage: ensure_output_dir_prepared(Backend, Opts)

- **Description**: Ensure that the output directories for backend `Backend` are prepared.

get_autodoc_opts/3:

Usage: get_autodoc_opts(Backend, Mod, Opts)

- **Description**: Get the list of documentation options `Opts` for the `FileBase` file.
  
  - The following properties should hold at call time:
    - `Backend` is an atom. *(atm/L)*
    - `Mod` is an atom. *(atm/L)*
    - `Opts` is a list of `supported_options`. *(list/L)*

autodoc_gen_doctree/5:

Usage: autodoc_gen_doctree(Backend, FileBase, SourceSuffix, Opts, Mod)

- **Description**: `FileBase` is the module specifier of the source file being documented (without extension, `SourceSuffix` is the suffix of the source). The output is a file whose contents document the main file, based on any assertions present in that file. The documentation is produced in the format given by `Backend` (the name of the output file also depends on `Backend`). The formats supported are given by `backend_id/L`.

  - **Call and exit should be compatible with**:
    - `Backend` is a supported backend. *(backend_id/L)*
    - `FileBase` is the base name of a file (without extension). *(basename/L)*
    - `SourceSuffix` is an atom. *(atm/L)*
    - `Opts` is a list of `supported_options`. *(list/L)*
    - `Mod` is an atom. *(atm/L)*

fmt_infodir_entry/3:

Usage: fmt_infodir_entry(DocSt, Version, Mod)

- **Description**: Generates a one line description (ASCII) of the application or library in a file for the directory of `emacs info` manuals.

  - The following properties should hold at call time:
    - `docstate(DocSt)` *(docstate/L)*
    - `Version` is any term. *(term/L)*
    - `Mod` is the base name of a file (without extension). *(basename/L)*

autodoc_compute_grefs/3:

Usage:

- **Description**: Compute the globally resolved references (including bibliography)
**autodoc_translate_doctree/3:**
Usage:
- *Description:* Translate the doctree using the specific backend

**autodoc_finish/1:**
No further documentation available for this predicate.

**autodoc_gen_alternative/2:**
No further documentation available for this predicate.

### 15.3 Documentation on multifiles (autodoc)

**autodoc_finish_hook/1:**
No further documentation available for this predicate. The predicate is *multifile*.

**autodoc_gen_alternative_hook/2:**
No further documentation available for this predicate. The predicate is *multifile*. 
Chapter 16: Internal State for Documentation Generation

16 Internal State for Documentation Generation

Author(s): Manuel Hermenegildo, Jose F. Morales.
This module defines the internal state of the documentation generation (for a single module).

16.1 Usage and interface (autodoc_state)

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

- Exports:
  - **Predicates:**
    ```prolog
    option_comment/2, backend_ignores_components/1, backend_alt_format/2, top_suffix/2, docst_backend/2, docst_curtmod/2, docst_set_curtmod/3, docst_opta/2, docst_set_opts/3, docst_inputfile/2, docst_new_no_src/4, docst_new_with_src/6, docst_new_sub/3, docst_message/2, docst_message/3, docst_opt/2, docst_curtmod_is_main/1, docst_no_components/1, docst_modname/2, labgen_init/1, labgen_clean/1, get/2, docst_mvar_lookup/3, docst_mvar_replace/4, docst_mvar_get/3, docst_mdname_clean/1, docst_mdname_assertz/2, docst_mdname_save/1, docst_gdata/3, docst_gdata_clean/1, docst_gdata_query/2, docst_gdata_query/3, docst_gdata_restore/1, docst_gdata_clean/1, docst_gvar_save/2, docst_gvar_restore/2, docst_has_index/2, all_indices/2, get_doc/4, get_doc_field/3, get_doc_field_dict/3, bind_dict_varnames/1, get_mod_doc/3, docst_modtype/2, get_first_loc_for_pred/3.
    ```
  - **Properties:**
    ```prolog
    supported_option/1.
    ```
  - **Regular Types:**
    ```prolog
    backend_id/1, docstate/1, modtype/1.
    ```

- Other modules used:
  - **Application modules:**
    ```prolog
    lpdocsr(src(autodoc_settings)), lpdocsr(src(autodoc_filesystem)),
    lpdocsr(src(autodoc_structure)), lpdocsr(src(autodoc_doctree)),
    lpdocsr(src(autodoc_refsdb)), lpdocsr(src(autodoc_parse)),
    lpdocsr(src(autodoc_index)), lpdocsr(src(comments)).
    ```
  - **System library modules:**
    ```prolog
    make/make_rt, dict, compiler/compiler, assertions/assert_lib, compiler/c_itf,
    assertions/assertions_props, messages, filenames, lists, terms, make/system_extra,
    write, read, system, aggregates.
    ```
  - **Internal (engine) modules:**
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

16.2 Documentation on exports (autodoc_state)

**supported_option/1:**

PROPERTY

Usage: supported_option(Option)
option_comment/2: PREDICATE
Usage: option_comment(Option,Text)

- Description: Option is a documentation option which is supported. Text describes the effect of selecting that option. Currently supported options are:

  option_comment(quiet, "Quiet output (good for testing). ").
  option_comment(no_bugs, "Do not include information on bugs. ").
  option_comment(no_authors, "Do not include author names. ").
  option_comment(no_stability, "Do not include stability comment. ").
  option_comment(no_version, "Do not include version information. ").
  option_comment(no_versioned_output, "Do not include version in the output name. ").
  option_comment(no_changelog, "Do not include change log. ").
  option_comment(no_patches, "Do not include comments for patches. ").
  option_comment(modes, "Do not translate modes and their arguments (except for properties) ").
  option_comment(head_props, "Do not move head properties to body. ").
  option_comment(literal_props, "Do not use text to document properties. ").
  option_comment(no_propnames, "Do not include property names in prop text. ").
  option_comment(no_undefined, "Do not signal undefined properties in text. ").
  option_comment(no_propsepln, "Do not put each property in a separate line. ").
  option_comment(no_biblio, "Do not include a bibliographical 'References' appendix. ").
  option_comment(no_sysmods, "Do not include system modules in list of libraries used. ").
  option_comment(no_engmods, "Do not include system engine modules in list of libraries used. ").
  option_comment(no_isoline, "Do not include *textual* description that a given usage conforms to the ISO standard. ").
  option_comment(propmods, "Include module name to which props belong. ").
  option_comment(no_propuses, "Do not include property uses (from assertions) in propdef. ").
  option_comment(shorttoc, "Produce shorter table of contents (no entries for individual defs of preds, props, etc.). ").
  option_comment(regtype_props, "Include in the doc for regtypes the global prop stating that they are indeed regtypes. ").
  option_comment(onesided, "For printing on one side (default is two). ").
  option_comment(no_math, "Disable mathematical environments. ").

- The following properties should hold upon exit:
  Option is a supported documentation option.  
  Text is a string (a list of character codes).

backend_id/1: REGTYPE
Usage: backend_id(Id)

- Description: Id is a supported backend.

backend_ignores_components/1: PREDICATE
Usage: backend_ignores_components(Id)

- Description: Id does not take into account components (only documents the mainfile)
backend_alt_format/2:
  Usage: backend_alt_format(Id,Ext)
  - Description: Ext is an alternative file format that can be generated by the backend Id

top_suffix/2:
  Usage: top_suffix(FileFormat,PrincipalExt)
  - Description: PrincipalExt is extension of the target file that will generate the file with FileFormat extension.

docstate/1:
  A regular type, defined as follows:
  
  docstate(docstate(Backend,Name,Opts,MVarDic,I)) :-
  backend_id(Backend),
  atom(Name),
  list(Opts,supported_option),
  dictionary(MVarDic),
  filename(I).

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

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

docst_set_currmod/3:
  No further documentation available for this predicate.

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

docst_set_opts/3:
  No further documentation available for this predicate.

docst_inputfile/2:
  No further documentation available for this predicate.
docst_new_no_src/4:
No further documentation available for this predicate.

docst_new_with_src/6:
No further documentation available for this predicate.

docst_new_sub/3:
No further documentation available for this predicate.

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

docst_message/3:
No further documentation available for this predicate.

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

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

docst_no_components/1:
Usage: docst_no_components(DocSt)
      — Description: DocSt specify an empty list of components

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

labgen_clean/1:
No further documentation available for this predicate.
labgen_get/2:
   No further documentation available for this predicate.

docst_mvar_lookup/3:
   No further documentation available for this predicate.

docst_mvar_replace/4:
   No further documentation available for this predicate.

docst_mvar_get/3:
   No further documentation available for this predicate.

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

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

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

docst_gdata/3:
   No further documentation available for this predicate.
   The predicate is of type data.

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

docst_gdata_query/3:
   No further documentation available for this predicate.

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

docst_gdata_clean/1:
   No further documentation available for this predicate.
The lpdoc Documentation Generator

**PREDICATE**

**docst_gvar_save/2:**
No further documentation available for this predicate.

**docst_gvar_restore/2:**
No further documentation available for this predicate.

**docst_has_index/2:**
No further documentation available for this predicate.

**all_indices/2:**
No further documentation available for this predicate.

**get_doc/4:**
No further documentation available for this predicate.

**get_doc_field/3:**
No further documentation available for this predicate.

**get_doc_field_dict/3:**
No further documentation available for this predicate.

**bind_dict_varnames/1:**
Usage: `bind_dict_varnames(Dict)`
- *Description:* Binds the variables in `Dict` to the corresponding names (i.e., the names that appeared in the source during read).

**get_mod_doc/3:**
No further documentation available for this predicate.

**modtype/1:**
- `modtype(part).`
- `modtype(application).`
- `modtype(documentation).`
- `modtype(module).`
- `modtype(user).`
- `modtype(include).`
- `modtype(package).`

Usage:
- *Description:* Represents the type of file being documented.
docst_modtype/2:  
No further documentation available for this predicate.

get_first_loc_for_pred/3:  
No further documentation available for this predicate.
17 Documentation Abstract Syntax Tree

Author(s): Manuel Hermenegildo (original version), Jose F. Morales.
This module defines the intermediate tree representation doctree/1 for documentation and its related operations.

Note: This part needs better documentation. – JFMC

17.1 Usage and interface (autodoc_doctree)

- Library usage:
  ```prolog```
  ```:- use_module(library(autodoc_doctree)).```  
- Exports:
  - Predicates:
    cmd_type/1, doctree_is_empty/1, is_nonempty_doctree/1, empty_doctree/1,
    doctree_insert_end/3, doctree_insert_before_section/3, doctree_concat/3,
    doclink_at/2, doclink_is_local/1, section_prop/2, section_select_prop/3,
    doctree_save/2, doctree_restore/2, doctree_simplify/2, doctree_putvars/5,
    doctree_scan_and_save.refs/2, doctree_prepare_docst_translate_and_write/3,
    doctree_to_rawtext/3, escape_string/4, is_version/1, version_patch/2,
    version_date/2, version_numstr/2, version_string/2, insert_show_toc/3.
  - Regular Types:
    doctree/1, doclink/1, doclabel/1, doctokens/1.
  - Multifiles:
    autodoc_rw_command_hook/4, autodoc_escape_string_hook/5.
- Other modules used:
  - Application modules:
    lpdosrc(src(autodoc_state)), lpdosrc(src(autodoc_refsd)), lpdosrc(src(autodoc_index)),
    lpdosrc(src(autodoc_structure)), lpdosrc(src(autodoc_settings)), lpdosrc(src(comments)),
    lpdosrc(src(autodoc_texinfo)), lpdosrc(src(autodoc_man)), lpdosrc(src(autodoc_html)).
  - System library modules:
    write, operators, format, lists, make/system_extra, read, terms, make/make_rt.
  - Internal (engine) modules:
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.

17.2 Documentation on exports (autodoc_doctree)

cmd_type/1:

PREDICATE

No further documentation available for this predicate.
**doctree/1:**

*Usage:*

- *Description:* Intermediate tree representation for documentation

---

**doctree_is_empty/1:**

*Usage: doctree_is_empty(+R)*

- *Description:* Emptiness test

---

**is_nonempty_doctree/1:**

*Usage: is_nonempty_doctree(+R)*

- *Description:* Not empty test

---

**empty_doctree/1:**

*Usage: empty_doctree(-R)*

- *Description:* Empty

---

**doctree_insert_end/3:**

*Usage: doctree_insert_end(A0,Elem,A)*

- *Description:* Insert *Elem* in *A0* at the end, obtaining *A*

---

**doctree_insert_before_section/3:**

*Usage: doctree_insert_before_section(A0,Elem,A)*

- *Description:* Insert *Elem* in *A0* before the first section, obtaining *A*

---

**doctree_concat/3:**

No further documentation available for this predicate.
**doclink/1:**
Usage:

- Description: A link to a document label

**doclabel/1:**
Usage:

- Description: An internal label

**doclink_at/2:**
No further documentation available for this predicate.

**doclink_is_local/1:**
No further documentation available for this predicate.

**doctokens/1:**
Usage:

- Description: Primitive doctree subset (ready for output, not further reducible)

**section_prop/2:**
No further documentation available for this predicate.

**section_select_prop/3:**
No further documentation available for this predicate.

**doctree_save/2:**

**doctree_restore/2:**

**doctree_simplify/2:**
No further documentation available for this predicate.

**doctree_putvars/5:**
Usage: doctree_putvars(R0, DocSt, PDict, VarDict, R)

- Description: Traverse R0 and replace each var(Name) doctree item with a fresh variable B. For each replacement, the term B=Var is introduced in VarDict, where Var is the associated value to Name in the dictionary PDict.
The following properties should hold at call time:
Intermediate tree representation for documentation (doctree/1)
docstate(DocSt) (docstate/1)

The following properties should hold upon exit:
Intermediate tree representation for documentation (doctree/1)

doctree_scan_and_save_refs/2:
Usage: doctree_scan_and_save_refs(R,DocSt)
Description: Scan and save the references of the doctree

The following properties should hold at call time:
Intermediate tree representation for documentation (doctree/1)
docstate(DocSt) (docstate/1)


doctree_prepare_docst_translate_and_write/3:
No further documentation available for this predicate.


doctree_to_rawtext/3:
Usage: doctree_to_rawtext(X,DocSt,Y)
Description: Y is a simplified raw text representation of the X
Call and exit should be compatible with:
Intermediate tree representation for documentation (doctree/1)
docstate(DocSt) (docstate/1)
Y is a string (a list of character codes). (string/1)


doctree_translate_and_write/3:
No further documentation available for this predicate.

escape_string/4:
Usage:
Description: Escapes needed characters in input string as needed for the target format.
The following properties should hold upon exit:
Arg1 is currently instantiated to an atom. (atom/1)
Arg2 is a string (a list of character codes). (string/1)
docstate(Arg3) (docstate/1)
Arg4 is a string (a list of character codes). (string/1)

is_version/1:
No further documentation available for this predicate.
version_patch/2:  
No further documentation available for this predicate.

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

version_numstr/2:  
Usage: version_numstr(Version,Str)  
— Description: Obtain the string \texttt{Str} representation of version \texttt{Version} (except date)

version_string/2:  
Usage: version_string(Version,Str)  
— Description: Obtain the string \texttt{Str} representation of version \texttt{Version} (including date)

insert_show_toc/3:  
Usage: insert_show_toc(RO,DocSt,R)  
— Description: Insert the command to show the table of contents in a given doctree/1.  
The right place may be different depending on the chosen backend.

17.3 Documentation on multifiles (autodoc_doctree)

autodoc_rw_command_hook/4:  
No further documentation available for this predicate.  
The predicate is \textit{multifile}.

autodoc_escape_string_hook/5:  
No further documentation available for this predicate.  
The predicate is \textit{multifile}.
18 Handling the Document Structure

Author(s): Jose F. Morales.

18.1 Usage and interface (autodoc_structure)

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

- Exports:
  - Predicates:
    - `docstr_node/4`, `clean_docstr/0`, `parse_structure/0`, `standalone_docstr/1`, `get_mainmod/1`, `get_mainmod_spec/1`, `all_component_specs/1`.

- Other modules used:
  - Application modules:
    - `lpdocsrc(src(autodoc_settings))`.
  - System library modules:
    - `filenames`, `aggregates`, `terms`, `make/make_rt`.
  - Internal (engine) modules:
    - `term_basic`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_compare`, `term_typing`, `hiord_rt`, `debugger_support`.

18.2 Documentation on exports (autodoc_structure)

- `docstr_node/4`:
  No further documentation available for this predicate. The predicate is of type `data`.

- `clean_docstr/0`:
  No further documentation available for this predicate.

- `parse_structure/0`:
  No further documentation available for this predicate.

- `standalone_docstr/1`:
  No further documentation available for this predicate.

- `get_mainmod/1`:
  No further documentation available for this predicate.
get_mainmod_spec/1:                                  PREDICATE
   No further documentation available for this predicate.

all_component_specs/1:                                PREDICATE
   No further documentation available for this predicate.
19 Access to Default Settings

Author(s): Jose F. Morales.
This module defines the setting values with some default values.

Note: This part needs better documentation. – JFMC

19.1 Usage and interface (autodoc_settings)

- Library usage:
  ```prolog```
  ```:- use_module(library(autodoc_settings)).```  
- Exports:
  - Predicates:
    ```prolog```  
    lpdoc_option/1, verify_settings/0, check_setting/1, setting_value_or_default/2, setting_value_or_default/3, setting_value/2, all_setting_values/2, get_command_option/1, requested_file_formats/1, load_vpaths/0, viewer/3, xdvi/1, xdivsize/1, pdfview/1, psview/1, htmlview/1, bibtext/1, tex/1, texindex/1, dvips/1, ps2pdf/1, makeinfo/1, makertf/1, rtftohlp/1, convertc/1.
  - Other modules used:
    - System library modules:
      ```prolog```  
      make/system_extra, make/make_rt, aggregates, distutils/distutils.
    - Internal (engine) modules:
      ```prolog```  
      term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

19.2 Documentation on exports (autodoc_settings)

lpdoc_option/1:  
  Defines the global options of lpdoc.
  The predicate is of type data.

verify_settings/0:  
  No further documentation available for this predicate.

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

setting_value_or_default/2:  
  Usage: setting_value_or_default(Var,Value)
— Description: Returns in Value the value of the variable Var. In case this variable does not exists, it returns a default value. If there is no default value for the variable Var it fails.

**setting_value_or_default/3:**
- No further documentation available for this predicate.

**setting_value/2:**
- No further documentation available for this predicate.

**all_setting_values/2:**
- No further documentation available for this predicate.

**get_command_option/1:**
- No further documentation available for this predicate.

**requested_file_formats/1:**
- Usage: requested_file_formats(F)
  - Description: F is a requested file format

**load_vpaths/0:**
- No further documentation available for this predicate.

**viewer/3:**
- No further documentation available for this predicate.

**xdvi/1:**
- No further documentation available for this predicate.

**xdvisize/1:**
- No further documentation available for this predicate.

**pdfview/1:**
- No further documentation available for this predicate.
psview/1:
   No further documentation available for this predicate.

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

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

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

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

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

ps2pdf/1:
   No further documentation available for this predicate.

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

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

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

convertc/1:
   No further documentation available for this predicate.
LPdoc Backends
20 Texinfo Backend

Author(s): Manuel Hermenegildo, Jose F. Morales.

20.1 Usage and interface (autodoc_texinfo)

- Library usage:
  ```prolog
  :- use_module(library(autodoc_texinfo)).
  ```
- Exports:
  - Predicates:
    ```prolog
    infodir_base/2.
    ```
  - Multifiles:
    ```prolog
    autodoc_escape_string_hook/5, autodoc_rw_command_hook/4, autodoc_finish_hook/1, autodoc_gen_alternative_hook/2.
    ```
- Other modules used:
  - Application modules:
    ```prolog
    lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_filesystem)),
    lpdocsrc(src(autodoc_structure)), lpdocsrc(src(autodoc_index)),
    lpdocsrc(src(autodoc_doctree)), lpdocsrc(src(autodoc_images)),
    lpdocsrc(src(autodoc_settings)), fastformat, lpdocsrc(src(comments)),
    claodesrc(makedir(ConfigValues)), lpdocsrc(src(autodoc_aux)).
    ```
  - System library modules:
    ```prolog
    lists, terms, format, messages, system, make/make_rt, file_utils, make/system_extra.
    ```
  - Internal (engine) modules:
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic Props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

20.2 Documentation on exports (autodoc_texinfo)

infodir_base/2: PREDICATE

No further documentation available for this predicate.

20.3 Documentation on multifiles (autodoc_texinfo)

autodoc_escape_string_hook/5: PREDICATE

No further documentation available for this predicate.

The predicate is multifile.

autodoc_rw_command_hook/4: PREDICATE

The predicate is multifile.

Usage: autodoc_rw_command_hook(Backend,DocSt,Command,NewCommand)
The following properties should hold at call time:

- Backend is a supported backend.
- docstate(DocSt)
- Intermediate tree representation for documentation
- Intermediate tree representation for documentation

autodoc_finish_hook/1:
No further documentation available for this predicate.
The predicate is multifile.

autodoc_gen_alternative_hook/2:
No further documentation available for this predicate.
The predicate is multifile.
21 HTML Backend

Author(s): Jose F. Morales.

21.1 Usage and interface (autodoc_html)

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

- Other modules used:
  
  - Application modules:
    
    ```prolog
    lpdocsr(src(autodoc_state)), lpdocsr(src(autodoc_structure)),
    lpdocsr(src(autodoc_filesystem)), lpdocsr(src(autodoc_doctree)),
    lpdocsr(src(autodoc_index)), lpdocsr(src(autodoc_refsdb)),
    lpdocsr(src(autodoc_images)), lpdocsr(src(autodoc_settings)),
    lpdocsr(src(comments)), fastformat, lpdocsr(src(autodoc_html_template)),
    lpdocsr(src(distpkg_download)), lpdocsr(src(autodoc_html_resources)).
    ```
  
  - System library modules:
    
    ```prolog
    lists, dict, system, file_utils.
    ```

  - Internal (engine) modules:
    
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

21.2 Documentation on multifiles (autodoc_html)

`autodoc_escape_string_hook/5`: PREDICATE

No further documentation available for this predicate.

The predicate is multifile.

`autodoc_rw_command_hook/4`: PREDICATE

The predicate is multifile.

Usage: `autodoc_rw_command_hook(Backend,DocSt,Command,NewCommand)`

- The following properties should hold at call time:
  
    ```prolog
    Backend is a supported backend. ( backend_id/l)
    docstate(DocSt) ( docstate/l)
    Intermediate tree representation for documentation ( doctree/1)
    Intermediate tree representation for documentation ( doctree/1)
    ```

`autodoc_finish_hook/1`: PREDICATE

No further documentation available for this predicate.

The predicate is multifile.
autodoc_gen_alternative_hook/2:
   No further documentation available for this predicate.
The predicate is *multifile*.
22 Resource Handling for the HTML Backend

Author(s): Jose F. Morales.

22.1 Usage and interface (autodoc_html_resources)

- **Library usage:**
  
  ```prolog
  :- use_module(library(autodoc_html_resources)).
  ```

- **Exports:**
  
  - **Predicates:**
    
    prepare_web_skel/1, prepare_mathjax/0, using_mathjax/1.

- **Other modules used:**
  
  - **Application modules:**
    
    lpdocsrc(src(autodoc)), lpdocsrc(src(autodoc_settings)),
    lpdocsrc(src(autodoc_filesystem)).
  
  - **System library modules:**
    
    messages, file_utils, make/system_extra, distutils/dirutils, terms.
  
  - **Internal (engine) modules:**
    
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.

22.2 Documentation on exports (autodoc_html_resources)

**prepare_web_skel/1:**

No further documentation available for this predicate.

**prepare_mathjax/0:**

No further documentation available for this predicate.

**using_mathjax/1:**

No further documentation available for this predicate.
23 Template Support for the HTML Backend

Author(s): Jose F. Morales.

23.1 Usage and interface (autodoc_html_template)

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

- Exports:
  - Predicates:
    - `img_url/2`, `fmt_html_template/3`.

- Other modules used:
  - Application modules:
    - `lpdocsrc(src(autodoc_settings))`.
  - System library modules:
    - `component_registry/component_registry`, `messages`, `aggregates`, `system`, `file_utils`, `make/system_extra`, `distutils/dirutils`, `lists`, `terms`, `make/make_rt`, `pillow/html`.
  - Internal (engine) modules:
    - `term_basic`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_compare`, `term_typing`, `hiord_rt`, `debugger_support`.

23.2 Documentation on exports (autodoc_html_template)

`img_url/2`:

**Usage:** `img_url(Name,Url)`

- **Description:** Obtain the URL where image `Name` is or will be found.
- **Call and exit should be compatible with:**
  - `Name` is an atom.  
  - `Url` is a string (a list of character codes).

`fmt_html_template/3`:

No further documentation available for this predicate.
24 Man Pages (man) Backend

Author(s): Jose F. Morales, Manuel Hermenegildo.

24.1 Usage and interface (autodoc_man)

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

- Other modules used:
  - Application modules:
    lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_doctree)), lpdocsrc(src(autodoc_images)), lpdocsrc(src(autodoc_aux)), lpdocsrc(src(comments)), fastformat.
  - System library modules:
    lists.
  - Internal (engine) modules:
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

24.2 Documentation on multifiles (autodoc_man)

autodoc_rw_command_hook/4: PREDICATE

The predicate is multifile.
Usage: autodoc_rw_command_hook(Backend,DocSt,Command,NewCommand)

- The following properties should hold at call time:
  Backend is a supported backend.
  docstate(DocSt)
  Intermediate tree representation for documentation
  Intermediate tree representation for documentation

autodoc_finish_hook/1: PREDICATE

No further documentation available for this predicate.
The predicate is multifile.

autodoc_gen_alternative_hook/2: PREDICATE

No further documentation available for this predicate.
The predicate is multifile.
25 Filesystem Abstraction

Author(s): Jose F. Morales.

This module provides definitions to assign unique file-system paths and names for each of the intermediate and final results of documentation generation.

25.1 Usage and interface (autodoc_filesystem)

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

- Exports:
  - Predicates:
    ```prolog
    file_format_name/2, supported_file_format/1, file_format_provided_by_backend/3, clean_fs_db/0, get_output_dir/2, get_cache_dir/2, ensure_output_dir/1, ensure_cache_dir/1, main_absfile_in_format/2, main_absfile_for_subtarget/3, absfile_for_aux/3, absfile_for_subtarget/4, main_output_name/2, get_subbase/3, absfile_to_relfile/3, clean_all/0, clean_docs_no_texi/0, clean_all_temporal/0, clean_intermediate/0, clean_tex_intermediate/0.
    ```
  - Regular Types:
    ```prolog
    filename/1, basename/1, subtarget/1.
    ```

- Other modules used:
  - Application modules:
    ```prolog
    lpdocsrc(src(autodoc_settings)), lpdocsrc(src(autodoc_structure)), lpdocsrc(src(autodoc_state)), lpdocsrc(src(component_versions)).
    ```
  - System library modules:
    ```prolog
    aggregates, make/system_extra, terms, distutils/dirutils, system, component, registry/component.registry.
    ```
  - Internal (engine) modules:
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

25.2 Documentation on exports (autodoc_filesystem)

**filename/1:**

Usage: `filename(X)`

- Description: `X` is the name of a file.

**basename/1:**

Usage: `basename(X)`

- Description: `X` is the base name of a file (without extension).
subtarget/1:
Usage:
   - Description: The kind of intermediate/final results for a single documentation processing unit (module).

file_format_name/2:
Usage: file_format_name(Ext)  
   Description: No further documentation available for this predicate.

supported_file_format/1:
Usage: supported_file_format(Ext)  
   Description: No further documentation available for this predicate.

file_format_provided_by_backend/3:
Usage: file_format_provided_by_backend(Ext, Backend, Subtarget)  
   Description: Backend is the backend that generates files with format Ext
   Call and exit should be compatible with:
      Ext is an atom.  \((\text{atm/1})\)
      Backend is a supported backend.  \((\text{backend_id/1})\)
      Subtarget is an atom.  \((\text{atm/1})\)

clean_fs_db/0:
Usage:
   - Description: Clean the cached information for the filesystem mapping of the documentation generation.

get_output_dir/2:
Usage: get_output_dir(Backend, Dir)  
   Description: Obtain the Dir directory where final documentation files will be stored

get_cache_dir/2:
Usage: get_cache_dir(Backend, Dir)  
   Description: Obtain the Dir directory where final documentation files will be stored

ensure_output_dir/1:
Usage: ensure_output_dir()  
   Description: No further documentation available for this predicate.

ensure_cache_dir/1:
Usage: ensure_cache_dir()  
   Description: No further documentation available for this predicate.
**main_absfile_in_format/2:**

Usage: `main_absfile_in_format(Ext,File)`

- Description: File is the absolute file name for the documentation in Ext format of the main module

**main_absfile_for_subtarget/3:**

No further documentation available for this predicate.

**absfile_for_aux/3:**

Usage: `absfile_for_aux(AuxName,Backend,AbsFile)`

- Description: Absolute file for an auxiliary output file (e.g. CSS, images, etc.)

**absfile_for_subtarget/4:**

No further documentation available for this predicate.

**main_output_name/2:**

No further documentation available for this predicate.

**get_subbase/3:**

Usage: `get_subbase(Base,Sub,SubBase)`

- Description: SubBase is the name for the sub-file (Sub) associated with Base
- The following properties should hold upon exit:
  - Base is the base name of a file (without extension).
  - Sub is an atom.
  - SubBase is the base name of a file (without extension).

**absfile_to_relfile/3:**

Usage: `absfile_to_relfile(A,Backend,B)`

- Description: Obtain the relative path, w.r.t. the output directory, of an absolute file. This is useful, e.g., for URLs.

**clean_all/0:**

No further documentation available for this predicate.

**clean_docs_no_texi/0:**

No further documentation available for this predicate.
**clean_all_temporal/0:**
No further documentation available for this predicate.

**clean_intermediate/0:**
No further documentation available for this predicate.

**clean_tex_intermediate/0:**
No further documentation available for this predicate.
26 Indexing Commands (Definition and Formatting)

Author(s): Jose F. Morales.
This module defines index commands and formatting.

Note: This part needs better documentation. – JFMC

26.1 Usage and interface (autodoc_index)

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

- Exports:
  
  - Predicates:
    
    - get_idxsub/2, get_idxbase/3, typeindex/5, idx_get_indices/3, is_index_cmd/1, codetype/1, normalize_index_cmd/3, fmt_idx_env/7, fmt_index/3.
  
- Other modules used:
  
    - Application modules:
      
      - lpdocsr(src(autodoc_state)), lpdocsr(src(autodoc_filesystem)), lpdocsr(src(autodoc_doctree)), lpdocsr(src(autodoc_structure)), lpdocsr(src(autodoc_refsdb)).
    
    - System library modules:
      
      - dict, lists, aggregates.
    
    - Internal (engine) modules:
      
      - term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

26.2 Documentation on exports (autodoc_index)

get_idxsub/2:

No further documentation available for this predicate.

get_idxbase/3:

No further documentation available for this predicate.

typeindex/5:

Usage: typeindex(Type, Index, IType, Name, Comment)

- Description: Index is the (info) index name in which objects of type Type go. Name is the title of the index in the documentation. IType is the type of index; an empty string means normal. codeComment is a comment to include before the index.
The following properties should hold upon exit:
Type is currently instantiated to an atom. (atom/1)
Index is an atom. (atm/1)
IType is a string (a list of character codes). (string/1)
Name is a string (a list of character codes). (string/1)
Intermediate tree representation for documentation (doctree/1)

idx_get_indices/3:
No further documentation available for this predicate.

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

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

normalize_index_cmd/3:
No further documentation available for this predicate.

fmt_idx_env/7:
No further documentation available for this predicate.

fmt_index/3:
No further documentation available for this predicate.
27 Database of Documentation References

Author(s): Jose F. Morales.

This module stores and manages all the documentation references (indices, sections, bibliography, etc.). It includes the generation of the table of contents.

27.1 Usage and interface (autodoc_refsdb)

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

- Exports:
  - Predicates:
    - compute_refs_and_biblio/1, prepare_current_refs/1, clean_current_refs/1, secttree_resolve/3.
  - Regular Types:
    - secttree/1.

- Other modules used:
  - Application modules:
    - lpdocsdisp(src(autodoc_state)), lpdocsdisp(src(autodoc_doctree)),
    - lpdocsdisp(src(autodoc_structure)), lpdocsdisp(src(autodoc_filesystem)),
    - lpdocsdisp(src(autodoc_bibrefs)), (autodoc_structure).
  - System library modules:
    - aggregates, lists.
  - Internal (engine) modules:
    - term_basic, arithmetic, atomic_basic, attributes, basic_prop, basiccontrol,
    - data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    - system_info, term_compare, term_typing, hiord_rt, debugger_support.

27.2 Documentation on exports (autodoc_refsdb)

compute_refs_and_biblio/1: PREDICATE

No further documentation available for this predicate.

prepare_current_refs/1: PREDICATE

Usage:
- Description: Prepare references for the translation of the current file.
- Call and exit should be compatible with:
  ```prolog
docstate(Arg1)
  ```

clean_current_refs/1: PREDICATE

Usage:
- **Description**: Clean the data stored by `prepare_current_refs/1`.
- **Call and exit should be compatible with**: `docstate(Arg1)` (docstate/1)

**secttree/1:**

Usage:

- **Description**: A tree of sections

**secttree_resolve/3:**

Usage: `secttree_resolve(LabelName,Tree,Link)`

- **Description**: Locate in the section tree `Tree` the section with label name `LabelName` and return the `Link` to the section.
- **Call and exit should be compatible with**:
  - `LabelName` is a string (a list of character codes). (string/1)
  - Intermediate tree representation for documentation (doctree/1)
  - A link to a document label (doclink/1)
28 Error Messages

Author(s): Manuel Hermenegildo.

28.1 Usage and interface (autodoc_errors)

- Library usage:
  ```prolog
  :- use_module(library(autodoc_errors)).
  ```
- Exports:
  - Predicates:
    ```prolog
    error_text/3.
    ```
- Other modules used:
  - Internal (engine) modules:
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

28.2 Documentation on exports (autodoc_errors)

**error_text/3:**

No further documentation available for this predicate.
29 Resolution of Bibliographical References

Author(s): Manuel Hermenegildo (original version), Jose F. Morales.
This module provides a predicate to resolve the bibliographical references found during the generation of documentation.

29.1 Usage and interface (autodoc_bibrefs)

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

- Exports:
  - Predicates:
    ```prolog
    resolve_bibliography/1.
    ```

- Other modules used:
  - Application modules:
    ```prolog
    lpdosrc(src(autodoc_state)), lpdosrc(src(autodoc_doctree)), lpdosrc(src(autodoc_refsdb)), lpdosrc(src(autodoc_aux)), lpdosrc(src(autodoc_settings)), lpdosrc(src(autodoc_parse)).
    ```
  - System library modules:
    ```prolog
    dict, aggregates, terms, file_utils, lists, format, make/make Rut, make/system extra.
    ```
  - Internal (engine) modules:
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

29.2 Documentation on exports (autodoc_bibrefs)

resolve_bibliography/1: PREDICATE

Usage: resolve_bibliography(DocSt)

- Description: This predicate resolves bibliographical references. The algorithm is as follows:
  - Write all the bibliographical references to a .aux file.
  - Invoke BibTeX with a customized .bst file that generates a pseudo-docstring.
  - Load the docstring and fix its syntax.
  - Parse the docstring as a doctree.
  - Extract (Label,Ref) pairs from 

Both the docstring and label/reference pairs are kept in the DocSt, and used later to map symbolic references to textual labels.

- The following properties should hold at call time:
  ```prolog
  docstate(DocSt) (docstate/1)
  ```
30 Auxiliary Definitions

Author(s): Manuel Hermenegildo, Jose F. Morales.

30.1 Usage and interface (autodoc_aux)

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

- Exports:
  - Predicates:
    ```prolog
    read_file/2, ascii_blank_lines/2, sh_exec/2.
    ```

- Other modules used:
  - Application modules:
    ```prolog
    lpdocsr(src(autodoc_settings)).
    ```
  - System library modules:
    ```prolog
    messages, system, make/system_extra, lists.
    ```
  - Internal (engine) modules:
    ```prolog
    term_basic, arithmetic, atomic_basic, attributes, basic_props, basiccontrol,
    data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic,
    system_info, term_compare, term_typing, hiord_rt, debugger_support.
    ```

30.2 Documentation on exports (autodoc_aux)

read_file/2:

No further documentation available for this predicate.

ascii_blank_lines/2:

No further documentation available for this predicate.

sh_exec/2:

No further documentation available for this predicate.
Chapter 31: Image Handling

31 Image Handling

Author(s): Jose F. Morales.

This module defines the handling of image commands. It defines predicates to locate and convert images in the different formats required for documentation.

Note: This part needs better documentation. – JFMC

31.1 Usage and interface (autodoc_images)

- **Library usage:**
  ```prolog
  :- use_module(library(autodoc_images)).
  ```

- **Exports:**
  - **Predicates:**
    - `locate_and_convert_image/4`, `clean_image_cache/0`.
  - **Other modules used:**
    - **Application modules:**
      - `lpdocrsrc(src(autodoc_state))`, `lpdocrsrc(src(autodoc_filesystem))`, `lpdocrsrc(src(autodoc_settings))`, `lpdocrsrc(src(autodoc_aux))`.
    - **System library modules:**
      - `terms`, `make/make_rt`, `make/system_extra`, `system`, `errhandle`, `messages`, `format`.
    - **Internal (engine) modules:**
      - `term_basic`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_compare`, `term_typing`, `hiord_rt`, `debugger_support`.

31.2 Documentation on exports (autodoc_images)

**locate_and_convert_image/4:**

Usage:

`locate_and_convert_image(SrcSpecS, AcceptedFormats, DocSt, TargetFileS)`

- **Description:** The image at `SrcSpecS` is located (as one of the known formats `known_format/1`) and converted to one of the `AcceptedFormats`. The target file is called `TargetFileS`
- **Call and exit should be compatible with:**
  - `SrcSpecS` is a string (a list of character codes).
  - `AcceptedFormats` is a list of atoms.
  - `docstate(DocSt)` (docstate/2)
  - `TargetFileS` is a string (a list of character codes).

**clean_image_cache/0:**

Usage:

- **Description:** Clean the cache for image copy/conversions.
References

[Bue95] F. Bueno.


Prolog: The Standard.
Springer-Verlag, 1996.

Cost Analysis of Logic Programs.

Non-Failure Analysis for Logic Programs.

Lower Bound Cost Estimation for Logic Programs.

[Her99] M. Hermenegildo.
A Documentation Generator for Logic Programming Systems.

[Her00] M. Hermenegildo.
A Documentation Generator for (C)LP Systems.

Compilation of Logic Programs for Restricted And-Parallelism.

Constraint Logic Programming: A Survey.

[Knu84] D. Knuth.
Literate programming.

A Methodology for Granularity Based Control of Parallelism in Logic Programs.

Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
An Assertion Language for Debugging of Constraint Logic Programs.


An Assertion Language for Constraint Logic Programs.
Library/Module Index

A
assertions ......................................... 41
assertions_props ................................. 51
autodoc ............................................. 111
autodoc_aux ....................................... 159
autodoc_bibrefs ................................. 157
autodoc_doctree ................................. 123
autodoc_errors ................................... 155
autodoc_filesystem .............................. 147
autodoc_html ..................................... 139
autodoc_html_resources ........................ 141
autodoc_html_template .......................... 143
autodoc_images .................................. 161
autodoc_index .................................... 151
autodoc_man ................................. 145
autodoc_refsdh .................................. 153
autodoc_settings ............................... 131
autodoc_state ................................... 115
autodoc_structure ............................... 129
autodoc_teiinfo ................................. 137
B
basic_props ....................................... 61
C
comments .......................................... 25
E
example_module .................................. 95
G
Generating ...................................... 15
L
lpdoc_examples .................................. 89
lpdoc_install .................................... 107
M
meta_props ....................................... 87
N
native_props ..................................... 77
R
regtypes ......................................... 57
rtchecks ......................................... 103
U
unittest .......................................... 105
# Predicate/Method Index

## A

<table>
<thead>
<tr>
<th>Predicate/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>absfile_for_aux/3</td>
<td>149</td>
</tr>
<tr>
<td>absfile_for_subtarget/4</td>
<td>149</td>
</tr>
<tr>
<td>absfile_to_refile/3</td>
<td>149</td>
</tr>
<tr>
<td>all_component_specs/1</td>
<td>130</td>
</tr>
<tr>
<td>all_indices/2</td>
<td>120</td>
</tr>
<tr>
<td>all_setting_values/2</td>
<td>132</td>
</tr>
<tr>
<td>ascii_blank_lines/2</td>
<td>159</td>
</tr>
<tr>
<td>autodoc_compute_grefs/3</td>
<td>113</td>
</tr>
<tr>
<td>autodoc_escape_string_hook/5</td>
<td>127, 137, 139</td>
</tr>
<tr>
<td>autodoc_finish/1</td>
<td>114</td>
</tr>
<tr>
<td>autodoc_finish_hook/1</td>
<td>114, 138, 139, 145</td>
</tr>
<tr>
<td>autodoc_gen_alternative/2</td>
<td>114</td>
</tr>
<tr>
<td>autodoc_gen_alternative_hook/2</td>
<td>114, 138, 140, 145</td>
</tr>
<tr>
<td>autodoc_gen_doctree/5</td>
<td>113</td>
</tr>
<tr>
<td>autodoc_rw_command_hook/4</td>
<td>127, 137, 139, 145</td>
</tr>
<tr>
<td>autodoc_translate_doctree/3</td>
<td>113</td>
</tr>
</tbody>
</table>

## B

<table>
<thead>
<tr>
<th>Predicate/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>backend_alt_format/2</td>
<td>117</td>
</tr>
<tr>
<td>backend_ignores_components/1</td>
<td>116</td>
</tr>
<tr>
<td>bibtex/1</td>
<td>133</td>
</tr>
<tr>
<td>bind_dict_varnames/1</td>
<td>120</td>
</tr>
</tbody>
</table>

## C

<table>
<thead>
<tr>
<th>Predicate/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>callme/2</td>
<td>88</td>
</tr>
<tr>
<td>check/1</td>
<td>48</td>
</tr>
<tr>
<td>check_setting/1</td>
<td>131</td>
</tr>
<tr>
<td>clean_all/0</td>
<td>149</td>
</tr>
<tr>
<td>clean_all_temporal/0</td>
<td>149</td>
</tr>
<tr>
<td>clean_current.refs/1</td>
<td>153</td>
</tr>
<tr>
<td>clean_docs_no_texi/0</td>
<td>149</td>
</tr>
<tr>
<td>clean_docstr/0</td>
<td>129</td>
</tr>
<tr>
<td>clean_fs_db/0</td>
<td>148</td>
</tr>
<tr>
<td>clean_image.cache/0</td>
<td>161</td>
</tr>
<tr>
<td>clean_intermediate/0</td>
<td>150</td>
</tr>
<tr>
<td>clean_tex_intermediate/0</td>
<td>150</td>
</tr>
<tr>
<td>cmd_type/1</td>
<td>123</td>
</tr>
<tr>
<td>codetype/1</td>
<td>152</td>
</tr>
<tr>
<td>compute_refs_and_biblio/1</td>
<td>153</td>
</tr>
<tr>
<td>convertc/1</td>
<td>133</td>
</tr>
</tbody>
</table>

## D

<table>
<thead>
<tr>
<th>Predicate/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_id_type/3</td>
<td>32</td>
</tr>
<tr>
<td>doclink_at/2</td>
<td>125</td>
</tr>
<tr>
<td>doclink_is_local/1</td>
<td>125</td>
</tr>
<tr>
<td>docst_backend/2</td>
<td>117</td>
</tr>
<tr>
<td>docst_curmod/2</td>
<td>117</td>
</tr>
<tr>
<td>docst_curmod_is_main/1</td>
<td>118</td>
</tr>
<tr>
<td>docst_gdata/3</td>
<td>119</td>
</tr>
<tr>
<td>docst_gdata_clean/1</td>
<td>119</td>
</tr>
<tr>
<td>docst_gdata_query/2</td>
<td>119</td>
</tr>
<tr>
<td>docst_gdata_query/3</td>
<td>119</td>
</tr>
<tr>
<td>docst_gdata_restore/1</td>
<td>119</td>
</tr>
<tr>
<td>docst_gvar_restore/2</td>
<td>120</td>
</tr>
<tr>
<td>docst_gvar_save/2</td>
<td>120</td>
</tr>
<tr>
<td>docst_has_index/2</td>
<td>120</td>
</tr>
<tr>
<td>docst_inputfile/2</td>
<td>117</td>
</tr>
<tr>
<td>docst_mdata_assertz/2</td>
<td>119</td>
</tr>
<tr>
<td>docst_mdata_clean/1</td>
<td>119</td>
</tr>
<tr>
<td>docst_mdata_save/1</td>
<td>119</td>
</tr>
<tr>
<td>docst_modname/2</td>
<td>119</td>
</tr>
<tr>
<td>docst_modtype/2</td>
<td>121</td>
</tr>
<tr>
<td>docst_mvar_get/3</td>
<td>119</td>
</tr>
<tr>
<td>docst_mvar_lookup/3</td>
<td>119</td>
</tr>
<tr>
<td>docst_mvar_replace/4</td>
<td>119</td>
</tr>
<tr>
<td>docst_new_no_src/4</td>
<td>117</td>
</tr>
<tr>
<td>docst_new_sub/3</td>
<td>118</td>
</tr>
<tr>
<td>docst_new_with_src/6</td>
<td>118</td>
</tr>
<tr>
<td>docst_no_components/1</td>
<td>118</td>
</tr>
<tr>
<td>docst_opt/2</td>
<td>118</td>
</tr>
<tr>
<td>docst_opts/2</td>
<td>117</td>
</tr>
<tr>
<td>docst_set_curmod/3</td>
<td>117</td>
</tr>
<tr>
<td>docst_set_opts/3</td>
<td>117</td>
</tr>
<tr>
<td>docstr_node/4</td>
<td>129</td>
</tr>
<tr>
<td>doctree_concat/3</td>
<td>124</td>
</tr>
<tr>
<td>doctree_insert_before_section/3</td>
<td>124</td>
</tr>
<tr>
<td>doctree_insert_end/3</td>
<td>124</td>
</tr>
<tr>
<td>doctree_is_empty/1</td>
<td>124</td>
</tr>
<tr>
<td>doctree_prepare_docst_translate_and_write/3</td>
<td>126</td>
</tr>
<tr>
<td>doctree_putvars/5</td>
<td>125</td>
</tr>
<tr>
<td>doctree_restore/2</td>
<td>125</td>
</tr>
<tr>
<td>doctree_save/2</td>
<td>125</td>
</tr>
<tr>
<td>doctree_scan_and_save_refs/2</td>
<td>126</td>
</tr>
<tr>
<td>doctree_simplify/2</td>
<td>125</td>
</tr>
<tr>
<td>doctree_to_rawtext/3</td>
<td>126</td>
</tr>
<tr>
<td>doctree_translate_and_write/3</td>
<td>126</td>
</tr>
<tr>
<td>dvips/1</td>
<td>133</td>
</tr>
</tbody>
</table>
rtftohlp/1 ........................................... 133

S
s/1 ..................................................... 98
section_prop/2 ...................................... 125
section_select_prop/3 .............................. 125
secttree_resolve/3 ................................. 154
setting_value/2 .................................... 132
setting_value_or_default/2 ................. 131
setting_value_or_default/3 ...........
sh_exec/2 ........................................... 159
standalone_docstr/1 .......................... 129
supported_file_format/1 ..................... 148

T
t/5 ..................................................... 98
tex/1 .................................................. 133
texindex/1 .......................................... 133
top_suffix/2 ....................................... 117
true/1 .............................................. 49
trust/1 .............................................. 48
typeindex/5 ........................................ 151

U
u/3 .................................................... 97
using_mathjax/1 .................................... 141

V
verify_settings/0 ................................. 131
version_date/2 ..................................... 127
version_numstr/2 ................................. 127
version_patch/2 ................................. 127
version_string/2 ................................. 127
viewer/3 ............................................. 132

W
w/1 ..................................................... 98

X
xdvi/1 .................................................. 132
xdvisize/1 ........................................... 132
## Property Index

**B**
- bind_ins/1 ........................................ 75

**C**
- call/2 ........................................... 87
- clique/1 ........................................ 77
- clique_1/1 ....................................... 78
- compat/2 ........................................ 72
- constraint/1 ..................................... 78
- covered/1 ........................................ 78
- covered/2 ........................................ 78

**D**
- deprecated/1 .................................... 73
- docstring/1 ...................................... 25, 56

**E**
- equiv/2 ........................................... 75
- error_free/1 ..................................... 75
- eval/1 ............................................ 75
- exception/1 ..................................... 78
- exception/2 ..................................... 78

**F**
- fails/1 ............................................ 79
- filter/2 .......................................... 75
- finite_solutions/1 ................................ 79

**H**
- have_choicepoints/1 ............................. 70
- head_pattern/1 ................................... 52

**I**
- indep/1 .......................................... 79
- indep/2 .......................................... 79
- inst/2 ........................................... 72
- instance/2 ....................................... 86
- is_det/1 .......................................... 79
- iso/1 ............................................. 73

**L**
- linear/1 .......................................... 80
- long/1 ........................................... 97

**M**
- member/2 ........................................... 69
- memo/1 ........................................... 75
- mshare/1 ......................................... 80
- mut_exclusive/1 .................................. 80

**N**
- nabody/1 ......................................... 54
- native/1 ......................................... 74
- native/2 ......................................... 74
- no_choicepoints/1 ................................ 80
- no_exception/1 .................................... 81
- no_exception/2 .................................... 81
- no_rtcheck/1 ..................................... 74
- no_signal/1 ....................................... 81
- no_signal/2 ....................................... 81
- non_det/1 ......................................... 81
- nonground/1 ....................................... 81
- not_covered/1 ..................................... 81
- not_fails/1 ......................................... 81
- not_further_inst/2 ................................ 73
- not_mut_exclusive/1 ............................... 82
- num_solutions/2 ................................... 82

**P**
- pe_type/1 ......................................... 76
- possibly_fails/1 .................................. 82
- possibly_nondet/1 ................................ 83
- prop/2 ............................................ 87
- prop_abs/1 ........................................ 88

**R**
- regtype/1 ......................................... 74
- regtype/2 ......................................... 88
- relations/2 ....................................... 83
S
sideff/2................................. 74
sideff_hard/1............................. 83
sideff_pure/1............................. 83
sideff_soft/1............................. 83
signal/1................................. 83
signal/2................................. 83
signals/2............................... 84
size/2.................................. 84
size/3.................................. 84
size_lb/2................................ 84
size_metric/3........................... 84
size_metric/4........................... 84
size_o/2............................... 84
size_ub/2............................... 84
solutions/2............................. 82

steps/2.................................. 85
steps_lb/2................................ 85
steps_o/2................................ 85
steps_ub/2................................ 85
stringcommand/1.......................... 26
supported_option/1........................ 115

T
tau/1................................... 85
terminates/1............................. 86
test_type/2............................... 86
throws/2................................ 86

U
user_output/2............................ 86
# Regular Type Index

## A
- aorb/1 ........................................ 95
- assert_body/1 .................................. 51
- assert_status/1 ................................ 56
- assert_type/1 .................................. 56
- atm/1 ........................................... 64
- atm_or_atm_list/1 ................................. 72

## B
- backend_id/1 .................................... 116
- bar/1 ........................................... 95
- basename/1 ...................................... 147
- baz/1 ........................................... 95

## C
- c_assert_body/1 .................................. 54
- callable/1 ...................................... 66
- character_code/1 ................................ 70
- complex_arg_property/1 .......................... 52
- complex_goal_property/1 .......................... 53
- constant/1 ...................................... 66

## D
- dictionary/1 ..................................... 54
- doclabel/1 ...................................... 125
- doclink/1 ...................................... 125
- docstate/1 ...................................... 117
- doctokens/1 ..................................... 125
- doctree/1 ....................................... 124

## F
- filename/1 ...................................... 147
- filetype/1 ....................................... 32
- flag_values/1 ................................... 75
- flt/1 ........................................... 63

## G
- g_assert_body/1 .................................. 55
- gnd/1 ........................................... 65
- gndstr/1 ........................................ 65

## I
- int/1 ............................................ 62

## L
- list/1 ........................................... 67, 99
- list/2 ........................................... 68, 101
- list_or_aorb/2 .................................. 96

## M
- modtype/1 ...................................... 120

## N
- nlist/2 .......................................... 68
- negint/1 ........................................ 62
- num/1 ........................................... 63
- num_code/1 ....................................... 71

## O
- operator_specifier/1 .............................. 67

## P
- predfunctor/1 ..................................... 56
- predname/1 ....................................... 71
- property_conjunction/1 .......................... 53
- property_startterm/1 .............................. 53
- propfunctor/1 ..................................... 56

## S
- s_assert_body/1 .................................. 54
- secttree/1 ...................................... 154
- sequence/2 ...................................... 69
- sequence_or_list/2 ................................. 70
- string/1 ........................................ 71
- struct/1 ........................................ 64
- subtarget/1 ...................................... 147

## T
- term/1 ........................................... 61
- time_struct/1 ................................... 39
- tree_of/2 ....................................... 96

## V
- version_descriptor/1 ............................... 32
- version_maintenance_type/1 ....................... 39
- version_number/1 ................................ 38
# Declaration Index

## C
- calls/1 .............................................. 43
- calls/2 .............................................. 43
- comment/2 ......................................... 48
- comp/1 .............................................. 45
- comp/2 .............................................. 45

## D
- decl/1 .............................................. 47
- decl/2 .............................................. 47
- doc/2 .............................................. 32, 47

## E
- entry/1 .............................................. 46
- exit/1 .............................................. 46
- exit/2 .............................................. 47

## M
- modef/1 .............................................. 47

## P
- pred/1 .............................................. 42
- pred/2 .............................................. 43
- prop/1 .............................................. 45
- prop/2 .............................................. 46

## R
- regtype/1 ........................................... 60
- regtype/2 ........................................... 60

## S
- success/1 .......................................... 44
- success/2 .......................................... 44

## T
- test/1 .............................................. 44
- test/2 .............................................. 44
- texec/1 ............................................ 43
- texec/2 ............................................ 43
Concept Index

.bib files ........................................ 16, 29

@  
@! command ......................................... 31
@' command ......................................... 31
@. command ......................................... 31
@.. command ......................................... 31
@= command ......................................... 31
@? command ......................................... 31
@' command ......................................... 31
@~ command ......................................... 31
@" command ......................................... 31
@~ command ......................................... 31
@aa command ........................................ 31
@AA command ........................................ 31
@ae command ........................................ 31
@AE command ........................................ 31
@apl command ....................................... 28
©author command .................................... 29
@b command ......................................... 31
@begin{alert} command .............................. 27
@begin{cartouche} command ........................ 27
@begin{description} command ........................ 26
@begin{displaymath} command ........................ 30
@begin{enumerate} command ......................... 26
@begin{itemize} command ......................... 26
@begin{verbatim} command ........................ 27
@bf command ........................................ 27
©bullet command .................................... 31
@c command ......................................... 31
@cindex command .................................... 28
@cite command .................................... 29
©comment command .................................. 26
©concept command .................................. 28
@copyright command ................................ 31
@d command ......................................... 31
@decl command ...................................... 28
@defmathcmd/2 command ......................... 30
@defmathcmd/3 command ......................... 30
@en command ...................................... 27
@email command ................................... 29
@end{alert} command ................................ 27
@end{cartouche} command ........................ 27
@end{description} command ....................... 27
@end{displaymath} command ...................... 30
@end{enumerate} command ......................... 26
@end{itemize} command ............................ 26
@end{verbatim} command .......................... 27
@end{enumerate} command ......................... 26
@end{itemize} command ............................ 26
@file command ..................................... 29
@footnote command ................................ 27
@H command ........................................ 31
@highlight command ................................ 27
@i command ......................................... 31
@key command ....................................... 27
@l command .......................................... 31
@L command ......................................... 31
@lib command ....................................... 28
@math command ...................................... 29
@noindent command ................................ 27
@o command ......................................... 31
@O command ......................................... 31
@oe command ........................................ 31
@OE command ........................................ 31
@op command ........................................ 28
@p command ......................................... 27
@pred command ...................................... 28
@ref command ....................................... 29
@result command ................................... 31
@section command .................................. 27
@sp command ........................................ 27
@ss command ........................................ 31
@subsection command ................................ 27
@t command ......................................... 31
@today command ..................................... 29
@tt command ......................................... 27
@u command ......................................... 31
@uref command ...................................... 29
@v command ......................................... 31
@var command ....................................... 29
@version command .................................. 29
two-sided ................................. 16
typewriter-like font ...................... 27

U

unit tests ................................. 105
Universal Resource Locator ............ 29
URL ........................................ 29
urls ......................................... 29
usage of a command ...................... 30
usage of the application ................. 16

V

verbatim text .............................. 27
version ..................................... 29
version maintenance mode for packages .. 37
version number ............................ 36

W

WWW address .............................. 29
Author Index

A
Anonymous Author 1 .................................. 95
Anonymous Author 2 .................................. 95

D
Daniel Cabeza ......................................... 61

E
Edison Mera ........................................... 77, 103, 105

F
Francisco Bueno ....................................... 41, 57, 77, 87

G
German Puebla ......................................... 41

J
Jose F. Morales .... 111, 115, 123, 129, 131, 137, 139, 141, 143, 145, 147, 151, 153, 157, 159, 161

M
Manuel Hermeñegildo .. 15, 25, 41, 51, 57, 61, 77, 89, 107, 111, 115, 123, 137, 145, 155, 157, 159

P
Pedro Lopez ............................................ 57, 77
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.

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ......
<table>
<thead>
<tr>
<th>Command Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>@lib command</td>
<td>28</td>
</tr>
<tr>
<td>@math command</td>
<td>29</td>
</tr>
<tr>
<td>@noindent command</td>
<td>27</td>
</tr>
<tr>
<td>@c command</td>
<td>31</td>
</tr>
<tr>
<td>@U command</td>
<td>31</td>
</tr>
<tr>
<td>@oe command</td>
<td>31</td>
</tr>
<tr>
<td>@OE command</td>
<td>31</td>
</tr>
<tr>
<td>@op command</td>
<td>28</td>
</tr>
<tr>
<td>@p command</td>
<td>27</td>
</tr>
<tr>
<td>@pred command</td>
<td>27</td>
</tr>
<tr>
<td>@ref command</td>
<td>29</td>
</tr>
<tr>
<td>@result command</td>
<td>31</td>
</tr>
<tr>
<td>@section command</td>
<td>27</td>
</tr>
<tr>
<td>@sp command</td>
<td>27</td>
</tr>
<tr>
<td>@ss command</td>
<td>31</td>
</tr>
<tr>
<td>@subcommand</td>
<td>27</td>
</tr>
<tr>
<td>@t command</td>
<td>31</td>
</tr>
<tr>
<td>@today command</td>
<td>29</td>
</tr>
<tr>
<td>@tt command</td>
<td>27</td>
</tr>
<tr>
<td>@u command</td>
<td>31</td>
</tr>
<tr>
<td>@uref command</td>
<td>29</td>
</tr>
<tr>
<td>@v command</td>
<td>31</td>
</tr>
<tr>
<td>@var command</td>
<td>29</td>
</tr>
<tr>
<td>@version command</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>absf ile_for_aux/3</td>
<td>147, 149</td>
</tr>
<tr>
<td>absf ile_for_subtarget/4</td>
<td>147, 149</td>
</tr>
<tr>
<td>absfile_to_relfile/3</td>
<td>147, 149</td>
</tr>
<tr>
<td>abstract</td>
<td>34</td>
</tr>
<tr>
<td>accents</td>
<td>30</td>
</tr>
<tr>
<td>acceptable modes</td>
<td>52</td>
</tr>
<tr>
<td>acknowledgements</td>
<td>34</td>
</tr>
<tr>
<td>author</td>
<td>33</td>
</tr>
<tr>
<td>author indexing</td>
<td>33</td>
</tr>
<tr>
<td>autodoc</td>
<td>3, 16, 111</td>
</tr>
<tr>
<td>autodoc_aux</td>
<td>159</td>
</tr>
<tr>
<td>autodoc_bibrefs</td>
<td>157</td>
</tr>
<tr>
<td>autodoc_compute_grefs/3</td>
<td>112, 113</td>
</tr>
<tr>
<td>autodoc_doctree</td>
<td>123</td>
</tr>
<tr>
<td>autodoc_errors</td>
<td>155</td>
</tr>
<tr>
<td>autodoc_escape_string_hook/5</td>
<td>123, 127, 137, 139</td>
</tr>
<tr>
<td>autodoc_filesystem</td>
<td>147</td>
</tr>
<tr>
<td>autodoc_finish/1</td>
<td>112, 114</td>
</tr>
<tr>
<td>autodoc_finish_hook/1</td>
<td>112, 114, 137, 138, 139, 145</td>
</tr>
<tr>
<td>autodoc_gen_alternative/2</td>
<td>112, 114</td>
</tr>
<tr>
<td>autodoc_gen_alternative_hook/2</td>
<td>112, 114, 137, 138, 140, 145</td>
</tr>
<tr>
<td>autodoc_gen_doctree/5</td>
<td>112, 113</td>
</tr>
<tr>
<td>autodoc_html/4</td>
<td>111, 139</td>
</tr>
<tr>
<td>autodoc_html_resources</td>
<td>141</td>
</tr>
<tr>
<td>autodoc_html_template</td>
<td>143</td>
</tr>
</tbody>
</table>

**A**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A4 paper</td>
<td>17</td>
</tr>
<tr>
<td>aggregates</td>
<td>112, 115, 129, 131, 143, 147, 151, 153, 157</td>
</tr>
<tr>
<td>all_componentspecs/1</td>
<td>129, 130</td>
</tr>
<tr>
<td>all_indices/2</td>
<td>115, 120</td>
</tr>
<tr>
<td>all_setting_values/2</td>
<td>131, 132</td>
</tr>
<tr>
<td>analyzer output</td>
<td>49</td>
</tr>
<tr>
<td>Anonymous Author 1</td>
<td>95</td>
</tr>
<tr>
<td>Anonymous Author 2</td>
<td>95</td>
</tr>
<tr>
<td>assert</td>
<td>34</td>
</tr>
<tr>
<td>assertion status</td>
<td>43, 44, 45, 47</td>
</tr>
<tr>
<td>assertions</td>
<td>3, 22, 23, 25, 41, 42, 51, 111</td>
</tr>
<tr>
<td>assertions/assrt_lib</td>
<td>112, 115</td>
</tr>
<tr>
<td>assertions/native_props</td>
<td>61, 95</td>
</tr>
<tr>
<td>assertions/props</td>
<td>51</td>
</tr>
<tr>
<td>assrt_body/1</td>
<td>42, 43, 46, 47, 51, 60</td>
</tr>
<tr>
<td>assrt_status/1</td>
<td>43, 44, 45, 46, 47, 51, 56, 60</td>
</tr>
<tr>
<td>atm/1</td>
<td>38, 61, 64, 113, 143, 148, 149, 152</td>
</tr>
<tr>
<td>atm_or_atm_list/1</td>
<td>61, 72</td>
</tr>
<tr>
<td>atomic_basic</td>
<td>112, 126, 152</td>
</tr>
<tr>
<td>author</td>
<td>33</td>
</tr>
<tr>
<td>author indexing</td>
<td>33</td>
</tr>
<tr>
<td>autodoc</td>
<td>3, 16, 111</td>
</tr>
<tr>
<td>autodoc_aux</td>
<td>159</td>
</tr>
<tr>
<td>autodoc_bibrefs</td>
<td>157</td>
</tr>
<tr>
<td>autodoc_compute_grefs/3</td>
<td>112, 113</td>
</tr>
<tr>
<td>autodoc_doctree</td>
<td>123</td>
</tr>
<tr>
<td>autodoc_errors</td>
<td>155</td>
</tr>
<tr>
<td>autodoc_escape_string_hook/5</td>
<td>123, 127, 137, 139</td>
</tr>
<tr>
<td>autodoc_filesystem</td>
<td>147</td>
</tr>
<tr>
<td>autodoc_finish/1</td>
<td>112, 114</td>
</tr>
<tr>
<td>autodoc_finish_hook/1</td>
<td>112, 114, 137, 138, 139, 145</td>
</tr>
<tr>
<td>autodoc_gen_alternative/2</td>
<td>112, 114</td>
</tr>
<tr>
<td>autodoc_gen_alternative_hook/2</td>
<td>112, 114, 137, 138, 140, 145</td>
</tr>
<tr>
<td>autodoc_gen_doctree/5</td>
<td>112, 113</td>
</tr>
<tr>
<td>autodoc_html/4</td>
<td>111, 139</td>
</tr>
<tr>
<td>autodoc_html_resources</td>
<td>141</td>
</tr>
<tr>
<td>autodoc_html_template</td>
<td>143</td>
</tr>
<tr>
<td>Function/Variable</td>
<td>Pages</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>autodoc_images</td>
<td>161</td>
</tr>
<tr>
<td>autodoc_index</td>
<td>151</td>
</tr>
<tr>
<td>autodoc_man</td>
<td>111, 145</td>
</tr>
<tr>
<td>autodoc_refsd</td>
<td>153</td>
</tr>
<tr>
<td>autodoc_rw_command_hook/4</td>
<td>123, 127, 137, 139, 145</td>
</tr>
<tr>
<td>autodoc_settings</td>
<td>131</td>
</tr>
<tr>
<td>autodoc_state</td>
<td>115</td>
</tr>
<tr>
<td>autodoc_structure</td>
<td>129</td>
</tr>
<tr>
<td>autodoc_texinfo</td>
<td>111, 137</td>
</tr>
<tr>
<td>autodoc_translate_doctree/3</td>
<td>112, 113</td>
</tr>
<tr>
<td>autodocformats</td>
<td>3</td>
</tr>
<tr>
<td>automatic documentation</td>
<td>111</td>
</tr>
<tr>
<td>automatic documentation library</td>
<td>3, 111</td>
</tr>
<tr>
<td>avoiding indentation</td>
<td>27</td>
</tr>
<tr>
<td>backend_alt_format/2</td>
<td>115, 117</td>
</tr>
<tr>
<td>backend_id/1</td>
<td>113, 115, 116, 138, 139, 145, 148</td>
</tr>
<tr>
<td>backend_ignores_components/1</td>
<td>115, 116</td>
</tr>
<tr>
<td>bar/1</td>
<td>95, 100</td>
</tr>
<tr>
<td>basename/1</td>
<td>113, 147, 149</td>
</tr>
<tr>
<td>bash</td>
<td>18</td>
</tr>
<tr>
<td>basic_props</td>
<td>3, 25, 42, 51, 61, 77, 87, 95, 104, 106, 112, 115, 123, 129, 131, 137, 139, 141, 143, 145, 147, 151, 153, 155, 157, 159, 161</td>
</tr>
<tr>
<td>basic_props.pl</td>
<td>103</td>
</tr>
<tr>
<td>basic_props:regtype/1</td>
<td>57</td>
</tr>
<tr>
<td>basiccontrold</td>
<td>25, 42, 51, 61, 77, 87, 95, 104, 106, 112, 115, 123, 129, 131, 137, 139, 141, 143, 145, 147, 151, 153, 155, 157, 159, 161</td>
</tr>
<tr>
<td>baz(Arg3)</td>
<td>100</td>
</tr>
<tr>
<td>baz/1</td>
<td>95, 100</td>
</tr>
<tr>
<td>bibliographic citations</td>
<td>29</td>
</tr>
<tr>
<td>bibliographic entries</td>
<td>16</td>
</tr>
<tr>
<td>bibliographic entry</td>
<td>29</td>
</tr>
<tr>
<td>bibtex</td>
<td>11, 16, 29, 108</td>
</tr>
<tr>
<td>bibtex/1</td>
<td>131, 133</td>
</tr>
<tr>
<td>bind_dict_varnames/1</td>
<td>115, 120</td>
</tr>
<tr>
<td>bind_ins/1</td>
<td>61, 63, 69, 75</td>
</tr>
<tr>
<td>blank lines</td>
<td>27</td>
</tr>
<tr>
<td>bold face</td>
<td>27</td>
</tr>
<tr>
<td>bug</td>
<td>36</td>
</tr>
</tbody>
</table>

C

<table>
<thead>
<tr>
<th>Function/Variable</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_assert_body/1</td>
<td>43, 44, 46, 51, 54</td>
</tr>
<tr>
<td>c_itf</td>
<td>9</td>
</tr>
<tr>
<td>call(T,A)</td>
<td>102</td>
</tr>
<tr>
<td>call/1</td>
<td>54</td>
</tr>
<tr>
<td>call/2</td>
<td>87</td>
</tr>
<tr>
<td>callable/1</td>
<td>61, 66, 82, 87, 88</td>
</tr>
<tr>
<td>callme/2</td>
<td>87, 88</td>
</tr>
<tr>
<td>calls assertion</td>
<td>43, 44</td>
</tr>
<tr>
<td>calls/1</td>
<td>42, 43, 44, 46</td>
</tr>
<tr>
<td>calls/2</td>
<td>42, 43</td>
</tr>
<tr>
<td>character string</td>
<td>41</td>
</tr>
<tr>
<td>character_code/1</td>
<td>61, 70, 71</td>
</tr>
<tr>
<td>check assertion</td>
<td>48</td>
</tr>
<tr>
<td>Check(X)</td>
<td>82</td>
</tr>
<tr>
<td>check/1</td>
<td>42, 48, 49</td>
</tr>
<tr>
<td>check_setting/1</td>
<td>131</td>
</tr>
<tr>
<td>Ciao</td>
<td>3, 15, 19, 20, 89, 107</td>
</tr>
<tr>
<td>Ciao Emacs mode</td>
<td>15</td>
</tr>
<tr>
<td>cioc</td>
<td>8, 9</td>
</tr>
<tr>
<td>ciodesrc(makedirs(ConfigValues))</td>
<td>137</td>
</tr>
<tr>
<td>ciopp</td>
<td>77</td>
</tr>
<tr>
<td>cjpeg</td>
<td>108</td>
</tr>
<tr>
<td>clean_all/0</td>
<td>147, 149</td>
</tr>
<tr>
<td>clean_all_temporal/0</td>
<td>147, 149</td>
</tr>
<tr>
<td>clean_current_ref/1</td>
<td>153</td>
</tr>
<tr>
<td>clean_docs_no_texi/0</td>
<td>147, 149</td>
</tr>
<tr>
<td>clean_docstr/0</td>
<td>129</td>
</tr>
<tr>
<td>clean_fs_db/0</td>
<td>147, 148</td>
</tr>
<tr>
<td>clean_image_cache/0</td>
<td>161</td>
</tr>
<tr>
<td>clean_intermediate/0</td>
<td>147, 150</td>
</tr>
<tr>
<td>clean_tex_intermediate/0</td>
<td>147, 150</td>
</tr>
<tr>
<td>clique/1</td>
<td>77</td>
</tr>
<tr>
<td>clique_l/1</td>
<td>77, 78</td>
</tr>
<tr>
<td>CLP</td>
<td>3</td>
</tr>
<tr>
<td>cmd_type/1</td>
<td>123</td>
</tr>
<tr>
<td>codetype/1</td>
<td>151, 152</td>
</tr>
<tr>
<td>comment</td>
<td>26</td>
</tr>
<tr>
<td>comment assertion</td>
<td>48</td>
</tr>
<tr>
<td>comment string</td>
<td>52, 54, 55</td>
</tr>
<tr>
<td>comment/2</td>
<td>10, 42, 48</td>
</tr>
<tr>
<td>comments</td>
<td>3, 23, 25</td>
</tr>
<tr>
<td>comments, machine readable</td>
<td>41</td>
</tr>
<tr>
<td>CommentType=ack</td>
<td>34</td>
</tr>
<tr>
<td>CommentType=address</td>
<td>33</td>
</tr>
<tr>
<td>CommentType=appendix</td>
<td>35</td>
</tr>
<tr>
<td>CommentType=author</td>
<td>33</td>
</tr>
</tbody>
</table>
Global Index

- docst_new_with_src/6 115, 118
- docst_no_components/1 115, 118
- docst_opt/2 115, 118
- docst_opts/2 115, 117
- docst_set_currmod/3 115, 117
- docst_set_opts/3 115, 117
- docstate(Arg1) 153, 154
- docstate(Arg3) 126
- docstate(DocSt) 113, 126, 138, 139, 145, 157, 161
- docstate/1 113, 115, 117, 126, 138, 139, 145, 153, 154, 157, 161
- docstr_node/4 129
- docstring/1 25, 32, 33, 34, 35, 36, 41, 48, 51, 52, 54, 55, 56
- doctokens/1 123, 125
- doctree/1 123, 124, 126, 127, 138, 139, 145, 152, 154
- doctree_concat/3 123, 124
- doctree_insert_before_section/3 123, 124
- doctree_insert_end/3 123, 124
- doctree_is_empty/l 123, 124
- doctree_prepare_docst_translate_and_write/3 123, 126
- doctree_putvars/5 123, 125
- doctree_restore/2 123, 125
- doctree_save/2 123, 125
- doctree_scan_and_save_refs/2 123, 126
- doctree_simplify/2 123, 125
- doctree_to_rawtext/3 123, 126
- doctree_translate_and_write/3 123, 126
- document structure 16
- documentation format 111
- documentation strings 32
- DOTcshrc 107
- dvips 7, 108
- dvips/1 131, 133
- dynamic/1 19
- email address 29
- email addresses 11, 29
- emphasis face 27
- empty_doctree/1 123, 124
- encapsulated postscript 30
- engine/basic_props 95
- ensure_cache_dir/1 147, 148
- ensure_loaded/1 10, 22, 38
- ensure_output_dir/1 147, 148
- ensure_output_dir_prepared/2 112
- entry assertion 46
- entry/1 42, 46, 54
- enumerated list 26
- environment variables 107
- equiv/2 61, 75
- errhandle 161
- error(a,b) 105
- error_free/1 61, 75
- error_text/3 155
- escape sequences 28
- escape_string/4 123, 126
- eval/1 61, 62, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 75, 99, 101
- example of lpdoc use 89
- example_module 95
- exception(error(a,b)) 105
- exception/1 77, 78
- exception/2 77, 78
- exceptions 25, 42, 51, 61, 77, 87, 95, 104, 106, 112
- export/1 42, 46, 47
- export/2 42, 47
- exported predicates 111
- exports 103
- fails/l 77, 79
- false assertion 49
- false/1 42, 49
- fastformat 137, 139, 145
- file_format_name/2 147, 148
- file_format_provided_by_backend/3 147, 148
- file_utils 77, 137, 139, 141, 143, 157
- filename/1 147
- filenames 112, 115, 129

E

Edison Mera 77, 103, 105
emacs 11, 36, 39, 107, 108
Emacs 15, 18, 19, 20
emac Ciao mode 36
Emacs, accessing info files 19
Emacs, generating manuals from 15
Emacs, Lpdoc mode 15
emacs-library.el 108
Global Index

inserting images .................................. 11
inst/2 ........................................ 61, 72
installation ...................................... 3
installation, of manuals ........................... 18
INSTALLATION.lpdoc ............................... 23
instance/2 ....................................... 86
instantiation properties .......................... 57
int/1 ........................................... 61, 62, 82, 96, 97, 98, 100, 101
integer/1 ....................................... 53
internals ......................................... 77
internals manual ................................... 3, 22
introduction ....................................... 34
io_aux ........................................... 25, 42, 51, 61, 77, 87, 95, 104, 106, 112,
115, 123, 129, 131, 137, 139, 141, 143, 145, 147,
151, 153, 155, 157, 159, 161
io_basic .......................................... 25, 42, 51, 61, 77, 87, 95, 104, 106, 112,
115, 123, 129, 131, 137, 139, 141, 143, 145, 147,
151, 153, 155, 157, 159, 161
is/2 ............................................... 102
is_det/l .......................................... 62, 63, 64, 65, 66, 67, 68, 77, 79, 99
is_index_cmd/l .................................. 123, 124
is_nonempty_doctree/l ........................... 123, 126
iso/1 ............................................. 10, 61, 73, 100
italics face ...................................... 27
item in an itemized list .......................... 26
itemized list ..................................... 26

J
Jose F. Morales ................................. 111, 115, 123, 129, 131, 137, 139,
141, 143, 145, 147, 151, 153, 157, 159, 161

K
keyboard key ...................................... 27
known_format/1 .................................. 161

L
labgen_clean/l ................................... 115, 118
labgen_get/2 .................................... 115, 118
labgen_init/l .................................... 115, 118
LaTeX ............................................. 26
LaTeX notation .................................... 29
letter size paper .................................. 17
library ........................................... 3
library(basicmodes) ................................ 52
library(isomodes) .................................. 52
linear/1 ......................................... 77, 80
Linux ............................................. 107, 108
list/1 .................................. 61, 67, 68, 69, 72, 82, 96, 97, 98, 99, 100,
101
list/2 ........................................... 37, 38, 53, 61, 68, 97, 98, 101, 113, 161
list_or_aorb/2 .................................. 95, 96
lists .............................................. 42, 45, 77, 95, 112, 115, 123, 137, 143,
145, 151, 153, 157, 159
literate programming ................................ 19
load_vpaths/0 ................................... 131, 132
locate_and_convert_image/l ....................... 161
log of changes .................................... 36
long ............................................... 104
long/1 ........................................... 95, 97, 98
lpdoc ........................................... 1, 3, 4, 5, 8, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 29, 37, 41, 48, 52, 56, 89, 107, 108
lpdoc .help ...................................... 16
lpdoc all ........................................ 16, 17, 19
lpdoc_examples ................................... 89
lpdoc_install .................................... 107
lpdoc_option/l ................................... 131
lpdocsrc(src(autodoc)) ......................... 141
lpdocsrc(src(autodoc_aux)) ...................... 137, 145, 157, 161
lpdocsrc(src(autodoc_bibrefs)) .................. 153
lpdocsrc(src(autodoc_doctree)) ................. 112, 115, 137, 139, 145, 151, 153, 157
lpdocsrc(src(autodoc_filesytem)) .............. 112, 115, 123, 137, 141, 151, 153, 161
lpdocsrc(src(autodoc_html)) .................... 123
lpdocsrc(src(autodoc_html_resources)) .......... 112, 139
lpdocsrc(src(autodoc_html_template)) .......... 139
lpdocsrc(src(autodoc_images)) .................. 137, 139, 145
lpdocsrc(src(autodoc_index)) ................... 112, 115, 137, 139
lpdocsrc(src(autodoc_man)) .................... 123
lpdocsrc(src(autodoc_parse)) ........................ 112, 115, 157
lpdocsrc(src(autodoc_refsdb)) ................... 112, 123, 139, 151, 157
lpdocsrc(src(autodoc_settings)) .................. 112, 139
lpdocsrc(src(autodoc_state)) ................... 112, 115, 123, 137, 141, 143, 147, 151,
153, 157, 159
lpdocsrc(src(autodoc_structure)) ............... 112, 115, 137, 139, 145, 151, 153, 157

189