

The Ipdoc Documentation Generator

An Automatic Documentation Generator for (C)LP Systems

REFERENCE MANUAL

The Ciao Documentation Series

<http://ciao-lang.org/>

Generated/Printed on: 27 March 2013

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

Copyright © 1996-2011 Manuel Hermenegildo and José Francisco Morales.

This document may be freely read, stored, reproduced, disseminated, translated or quoted by any means and on any medium provided the following conditions are met:

1. Every reader or user of this document acknowledges that is aware that no guarantee is given regarding its contents, on any account, and specifically concerning veracity, accuracy and fitness for any purpose.
2. No modification is made other than cosmetic, change of representation format, translation, correction of obvious syntactic errors, or as permitted by the clauses below.
3. Comments and other additions may be inserted, provided they clearly appear as such; translations or fragments must clearly refer to an original complete version, preferably one that is easily accessed whenever possible.
4. Translations, comments and other additions or modifications must be dated and their author(s) must be identifiable (possibly via an alias).
5. This licence is preserved and applies to the whole document with modifications and additions (except for brief quotes), independently of the representation format.
6. Any reference to the "official version", "original version" or "how to obtain original versions" of the document is preserved verbatim. Any copyright notice in the document is preserved verbatim. Also, the title and author(s) of the original document should be clearly mentioned as such.
7. In the case of translations, verbatim sentences mentioned in (6.) are preserved in the language of the original document accompanied by verbatim translations to the language of the translated document. All translations state clearly that the author is not responsible for the translated work. This license is included, at least in the language in which it is referenced in the original version.
8. Whatever the mode of storage, reproduction or dissemination, anyone able to access a digitized version of this document must be able to make a digitized copy in a format directly usable, and if possible editable, according to accepted, and publicly documented, public standards.
9. Redistributing this document to a third party requires simultaneous redistribution of this licence, without modification, and in particular without any further condition or restriction, expressed or implied, related or not to this redistribution. In particular, in case of inclusion in a database or collection, the owner or the manager of the database or the collection renounces any right related to this inclusion and concerning the possible uses of the document after extraction from the database or the collection, whether alone or in relation with other documents.

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

Table of Contents

Summary	1
1 Introduction.....	3
1.1 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 Enhancing the documentation being generated	18
2.6 Accessing on-line manuals	19
2.6.1 Accessing html manuals	19
2.6.2 Accessing info manuals	19
2.6.3 Accessing man manuals	20
2.6.4 Putting it all together	20
2.7 Some usage tips	21
2.7.1 Ensuring Compatibility with All Supported Target Formats	21
2.7.2 Writing comments which document version/patch changes	21
2.7.3 Documenting Libraries and/or Applications	21
2.7.4 Documenting files which are not modules	22
2.7.5 Splitting large documents into parts	22
2.7.6 Documenting reexported predicates	22
2.7.7 Separating the documentation from the source file	23
2.7.8 Generating auxiliary files (e.g. READMEs)	23
2.8 Troubleshooting	23
3 Documentation Mark-up Language and Declarations.....	25
3.1 Usage and interface (<code>comments</code>)	25
3.2 Documentation on exports (<code>comments</code>)	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 (<code>comments</code>)	32
doc/2 (decl)	32
version_number/1 (regtype)	38
ymd_date/1 (regtype)	39
time_struct/1 (regtype)	39
version_maintenance_type/1 (regtype)	39

4	The Ciao assertion package	41
4.1	More info	41
4.2	Some attention points	41
4.3	Usage and interface (<code>assertions_doc</code>)	42
4.4	Documentation on new declarations (<code>assertions_doc</code>)	42
	(pred)/1 (decl)	42
	(pred)/2 (decl)	43
	(texec)/1 (decl)	43
	(texec)/2 (decl)	43
	(calls)/1 (decl)	43
	(calls)/2 (decl)	44
	(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 (<code>assertions_doc</code>)	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 (<code>assertions_props</code>)	51
5.2	Documentation on exports (<code>assertions_props</code>)	51
	assrt_body/1 (regtype)	51
	head_pattern/1 (prop)	52
	complex_arg_property/1 (regtype)	53
	property_conjunction/1 (regtype)	53
	property_starterm/1 (regtype)	53
	complex_goal_property/1 (regtype)	53
	nabody/1 (prop)	54
	dictionary/1 (regtype)	54
	c_assrt_body/1 (regtype)	54
	s_assrt_body/1 (regtype)	54
	g_assrt_body/1 (regtype)	55
	assrt_status/1 (regtype)	55
	assrt_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 (<code>regtypes_doc</code>)	60
6.3	Documentation on new declarations (<code>regtypes_doc</code>)	60
	(<code>regtype</code>)/1 (decl)	60
	(<code>regtype</code>)/2 (decl)	61
7	Basic data types and properties	63
7.1	Usage and interface (<code>basic_props</code>)	63
7.2	Documentation on exports (<code>basic_props</code>)	63
	<code>term</code> /1 (<code>regtype</code>)	63
	<code>int</code> /1 (<code>regtype</code>)	64
	<code>nnegint</code> /1 (<code>regtype</code>)	64
	<code>ft</code> /1 (<code>regtype</code>)	65
	<code>num</code> /1 (<code>regtype</code>)	66
	<code>atm</code> /1 (<code>regtype</code>)	66
	<code>struct</code> /1 (<code>regtype</code>)	67
	<code>gnd</code> /1 (<code>regtype</code>)	67
	<code>gndstr</code> /1 (<code>regtype</code>)	68
	<code>constant</code> /1 (<code>regtype</code>)	68
	<code>callable</code> /1 (<code>regtype</code>)	69
	<code>operator_specifier</code> /1 (<code>regtype</code>)	69
	<code>list</code> /1 (<code>regtype</code>)	70
	<code>list</code> /2 (<code>regtype</code>)	70
	<code>nlist</code> /2 (<code>regtype</code>)	71
	<code>member</code> /2 (prop)	71
	<code>sequence</code> /2 (<code>regtype</code>)	72
	<code>sequence_or_list</code> /2 (<code>regtype</code>)	72
	<code>character_code</code> /1 (<code>regtype</code>)	73
	<code>string</code> /1 (<code>regtype</code>)	73
	<code>num_code</code> /1 (<code>regtype</code>)	74
	<code>predname</code> /1 (<code>regtype</code>)	74
	<code>atm_or_atm_list</code> /1 (<code>regtype</code>)	74
	<code>compat</code> /2 (prop)	75
	<code>inst</code> /2 (prop)	75
	<code>iso</code> /1 (prop)	75
	<code>deprecated</code> /1 (prop)	76
	<code>not_further_inst</code> /2 (prop)	76
	<code>sideff</code> /2 (prop)	76
	(<code>regtype</code>)/1 (prop)	77
	<code>native</code> /1 (prop)	77
	<code>native</code> /2 (prop)	77
	<code>rtcheck</code> /1 (prop)	77
	<code>rtcheck</code> /2 (prop)	78
	<code>no_rtcheck</code> /1 (prop)	78
	<code>eval</code> /1 (prop)	79
	<code>equiv</code> /2 (prop)	79
	<code>bind_ins</code> /1 (prop)	79
	<code>error_free</code> /1 (prop)	79
	<code>memo</code> /1 (prop)	79
	<code>filter</code> /2 (prop)	79
	<code>flag_values</code> /1 (<code>regtype</code>)	79
	<code>pe_type</code> /1 (prop)	79

8 Properties which are native to analyzers 81

8.1	Usage and interface (<code>native_props</code>)	81
8.2	Documentation on exports (<code>native_props</code>)	81
	<code>clique/1</code> (prop)	81
	<code>clique_1/1</code> (prop)	82
	<code>compat/1</code> (prop)	82
	<code>constraint/1</code> (prop)	82
	<code>covered/1</code> (prop)	82
	<code>covered/2</code> (prop)	83
	<code>exception/1</code> (prop)	83
	<code>exception/2</code> (prop)	83
	<code>fails/1</code> (prop)	83
	<code>finite_solutions/1</code> (prop)	83
	<code>have_choicepoints/1</code> (prop)	83
	<code>indep/1</code> (prop)	84
	<code>indep/2</code> (prop)	84
	<code>instance/1</code> (prop)	84
	<code>is_det/1</code> (prop)	84
	<code>linear/1</code> (prop)	84
	<code>mshare/1</code> (prop)	85
	<code>mut_exclusive/1</code> (prop)	85
	<code>no_choicepoints/1</code> (prop)	85
	<code>no_exception/1</code> (prop)	86
	<code>no_exception/2</code> (prop)	86
	<code>no_signal/1</code> (prop)	86
	<code>no_signal/2</code> (prop)	86
	<code>non_det/1</code> (prop)	86
	<code>nonground/1</code> (prop)	86
	<code>not_covered/1</code> (prop)	86
	<code>not_fails/1</code> (prop)	87
	<code>not_mut_exclusive/1</code> (prop)	87
	<code>num_solutions/2</code> (prop)	87
	<code>solutions/2</code> (prop)	87
	<code>possibly_fails/1</code> (prop)	87
	<code>possibly_nondet/1</code> (prop)	88
	<code>relations/2</code> (prop)	88
	<code>sideff_hard/1</code> (prop)	88
	<code>sideff_pure/1</code> (prop)	88
	<code>sideff_soft/1</code> (prop)	89
	<code>signal/1</code> (prop)	89
	<code>signal/2</code> (prop)	89
	<code>signals/2</code> (prop)	89
	<code>size/2</code> (prop)	89
	<code>size/3</code> (prop)	90
	<code>size_lb/2</code> (prop)	90
	<code>size_o/2</code> (prop)	90
	<code>size_ub/2</code> (prop)	90
	<code>size_metric/3</code> (prop)	90
	<code>size_metric/4</code> (prop)	91
	<code>succeeds/1</code> (prop)	91
	<code>steps/2</code> (prop)	91
	<code>steps_lb/2</code> (prop)	91
	<code>steps_o/2</code> (prop)	91
	<code>steps_ub/2</code> (prop)	92
	<code>tau/1</code> (prop)	92
	<code>terminates/1</code> (prop)	92

	test_type/2 (prop)	92
	throws/2 (prop)	93
	user_output/2 (prop)	93
	instance/2 (prop)	93
9	Meta-properties	95
9.1	Usage and interface (meta_props)	95
9.2	Documentation on exports (meta_props)	95
	call/2 (prop)	95
	(prop)/2 (prop)	96
	(regtype)/2 (prop)	96
9.3	Documentation on multifiles (meta_props)	96
	callme/2 (pred)	96
9.4	Documentation on internals (meta_props)	96
	prop_abs/1 (prop)	96
10	An Example - Documenting a Library Module	
	97
11	Auto Documenter Output for the Example	
	Module	103
11.1	Usage and interface (example_module)	103
11.2	Documentation on exports (example_module)	103
	bar/1 (regtype)	103
	baz/1 (regtype)	103
	aorb/1 (regtype)	104
	tree_of/2 (regtype)	104
	list_or_aorb/2 (regtype)	104
	q/2 (pred)	104
	r/1 (pred)	104
	p/1 (pred)	105
	p/5 (pred)	105
	u/3 (pred)	105
	long/1 (prop)	105
	w/1 (pred)	106
	mytype/1 (pred)	106
	t/5 (pred)	106
	s/1 (pred)	106
	q/1 (pred)	107
	list/1 (regtype)	107
11.3	Documentation on multifiles (example_module)	108
	p/3 (pred)	108
11.4	Documentation on internals (example_module)	109
	list/2 (regtype)	109
	og/2 (modedef)	110
	is/2 (pred)	110
12	Run-time checking of assertions	111
12.1	Usage and interface (rtchecks_doc)	112
13	Unit Testing Library	113
13.1	Additional notes	113
13.2	Usage and interface (unittest_doc)	114

14	Installing lpdoc	115
14.1	Other software packages required (lpdoc)	115
PART II - LPdoc Internals Manual		117
15	Documentation Generation Library	119
15.1	Usage and interface (<code>autodoc</code>)	120
15.2	Documentation on exports (<code>autodoc</code>)	120
	<code>index_comment/2</code> (pred)	120
	<code>reset_output_dir_db/0</code> (pred)	120
	<code>ensure_output_dir_prepared/2</code> (pred)	120
	<code>get_autodoc_opts/3</code> (pred)	121
	<code>autodoc_gen_doctree/5</code> (pred)	121
	<code>fmt_infodir_entry/3</code> (pred)	121
	<code>autodoc_compute_grefs/3</code> (pred)	121
	<code>autodoc_translate_doctree/3</code> (pred)	121
	<code>autodoc_finish/1</code> (pred)	122
	<code>autodoc_gen_alternative/2</code> (pred)	122
15.3	Documentation on multifiles (<code>autodoc</code>)	122
	<code>autodoc_finish_hook/1</code> (pred)	122
	<code>autodoc_gen_alternative_hook/2</code> (pred)	122
16	Internal State for Documentation Generation	123
16.1	Usage and interface (<code>autodoc_state</code>)	123
16.2	Documentation on exports (<code>autodoc_state</code>)	124
	<code>supported_option/1</code> (prop)	124
	<code>option_comment/2</code> (pred)	124
	<code>backend_id/1</code> (regtype)	124
	<code>backend_ignores_components/1</code> (pred)	124
	<code>backend_alt_format/2</code> (pred)	125
	<code>top_suffix/2</code> (pred)	125
	<code>docstate/1</code> (regtype)	125
	<code>docst_backend/2</code> (pred)	125
	<code>docst_currmod/2</code> (pred)	125
	<code>docst_set_currmod/3</code> (pred)	125
	<code>docst_opts/2</code> (pred)	125
	<code>docst_set_opts/3</code> (pred)	125
	<code>docst_inputfile/2</code> (pred)	125
	<code>docst_new_no_src/4</code> (pred)	126
	<code>docst_new_with_src/6</code> (pred)	126
	<code>docst_new_sub/3</code> (pred)	126
	<code>docst_message/2</code> (pred)	126
	<code>docst_message/3</code> (pred)	126
	<code>docst_opt/2</code> (pred)	126
	<code>docst_currmod_is_main/1</code> (pred)	126
	<code>docst_no_components/1</code> (pred)	126
	<code>docst_modname/2</code> (pred)	126
	<code>labgen_init/1</code> (pred)	126
	<code>labgen_clean/1</code> (pred)	126
	<code>labgen_get/2</code> (pred)	127
	<code>docst_mvar_lookup/3</code> (pred)	127
	<code>docst_mvar_replace/4</code> (pred)	127
	<code>docst_mvar_get/3</code> (pred)	127

docst_mdata_clean/1 (pred)	127
docst_mdata_assertz/2 (pred)	127
docst_mdata_save/1 (pred)	127
docst_gdata/3 (pred)	127
docst_gdata_query/2 (pred)	127
docst_gdata_query/3 (pred)	127
docst_gdata_restore/1 (pred)	127
docst_gdata_clean/1 (pred)	128
docst_gvar_save/2 (pred)	128
docst_gvar_restore/2 (pred)	128
docst_has_index/2 (pred)	128
all_indices/2 (pred)	128
get_doc/4 (pred)	128
get_doc_changes/3 (pred)	128
get_doc_pred_varnames/2 (pred)	128
doc_assertion_read/9 (pred)	128
bind_dict_varnames/1 (pred)	128
get_mod_doc/3 (pred)	128
pred_has_docprop/2 (pred)	128
modtype/1 (regtype)	129
docst_modtype/2 (pred)	129
get_first_loc_for_pred/3 (pred)	129

17 Documentation Abstract Syntax Tree..... 131

17.1 Usage and interface (autodoc_doctree)	131
17.2 Documentation on exports (autodoc_doctree)	131
cmd_type/1 (pred)	131
doctree/1 (regtype)	132
doctree_is_empty/1 (pred)	132
is_nonempty_doctree/1 (pred)	132
empty_doctree/1 (pred)	132
doctree_insert_end/3 (pred)	132
doctree_insert_before_section/3 (pred)	132
doctree_concat/3 (pred)	133
doclink/1 (regtype)	133
doclabel/1 (regtype)	133
doclink_at/2 (pred)	133
doclink_is_local/1 (pred)	133
doctokens/1 (regtype)	133
section_prop/2 (pred)	133
section_select_prop/3 (pred)	133
doctree_save/2 (pred)	133
doctree_restore/2 (pred)	134
doctree_simplify/2 (pred)	134
doctree_putvars/5 (pred)	134
doctree_scan_and_save_refs/2 (pred)	134
doctree_prepare_docst_translate_and_write/3 (pred) ..	134
doctree_to_rawtext/3 (pred)	135
doctree_translate_and_write/3 (pred)	135
escape_string/4 (pred)	135
is_version/1 (pred)	135
version_patch/2 (pred)	135
version_date/2 (pred)	135
version_numstr/2 (pred)	135
version_string/2 (pred)	135
insert_show_toc/3 (pred)	135

17.3	Documentation on multifiles (<code>autodoc_doctree</code>).....	136
	<code>autodoc_rw_command_hook/4</code> (pred).....	136
	<code>autodoc_escape_string_hook/5</code> (pred).....	136
18	Handling the Document Structure.....	137
18.1	Usage and interface (<code>autodoc_structure</code>).....	137
18.2	Documentation on exports (<code>autodoc_structure</code>).....	137
	<code>docstr_node/4</code> (pred).....	137
	<code>clean_docstr/0</code> (pred).....	137
	<code>parse_structure/0</code> (pred).....	137
	<code>standalone_docstr/1</code> (pred).....	137
	<code>get_mainmod/1</code> (pred).....	137
	<code>get_mainmod_spec/1</code> (pred).....	138
	<code>all_component_specs/1</code> (pred).....	138
19	Access to Default Settings.....	139
19.1	Usage and interface (<code>autodoc_settings</code>).....	139
19.2	Documentation on exports (<code>autodoc_settings</code>).....	139
	<code>lpdoc_option/1</code> (pred).....	139
	<code>verify_settings/0</code> (pred).....	139
	<code>check_setting/1</code> (pred).....	139
	<code>setting_value_or_default/2</code> (pred).....	139
	<code>setting_value_or_default/3</code> (pred).....	140
	<code>setting_value/2</code> (pred).....	140
	<code>all_setting_values/2</code> (pred).....	140
	<code>get_command_option/1</code> (pred).....	140
	<code>requested_file_formats/1</code> (pred).....	140
	<code>load_vpaths/0</code> (pred).....	140
	<code>viewer/4</code> (pred).....	140
	<code>xdvi/1</code> (pred).....	140
	<code>xdvisize/1</code> (pred).....	140
	<code>bibtex/1</code> (pred).....	140
	<code>tex/1</code> (pred).....	140
	<code>texindex/1</code> (pred).....	141
	<code>dvips/1</code> (pred).....	141
	<code>ps2pdf/1</code> (pred).....	141
	<code>makeinfo/1</code> (pred).....	141
	<code>makertf/1</code> (pred).....	141
	<code>rtftohlp/1</code> (pred).....	141
	<code>convertc/1</code> (pred).....	141
	LPdoc Backends.....	143
20	Texinfo Backend.....	145
20.1	Usage and interface (<code>autodoc_texinfo</code>).....	145
20.2	Documentation on exports (<code>autodoc_texinfo</code>).....	145
	<code>infodir_base/2</code> (pred).....	145
20.3	Documentation on multifiles (<code>autodoc_texinfo</code>).....	145
	<code>autodoc_escape_string_hook/5</code> (pred).....	145
	<code>autodoc_rw_command_hook/4</code> (pred).....	145
	<code>autodoc_finish_hook/1</code> (pred).....	146
	<code>autodoc_gen_alternative_hook/2</code> (pred).....	146

21	HTML Backend	147
21.1	Usage and interface (<code>autodoc_html</code>)	147
21.2	Documentation on multifiles (<code>autodoc_html</code>)	147
	<code>autodoc_escape_string_hook/5</code> (pred)	147
	<code>autodoc_rw_command_hook/4</code> (pred)	147
	<code>autodoc_finish_hook/1</code> (pred)	147
	<code>autodoc_gen_alternative_hook/2</code> (pred)	148
22	Resource Handling for the HTML Backend ..	149
22.1	Usage and interface (<code>autodoc_html_resources</code>)	149
22.2	Documentation on exports (<code>autodoc_html_resources</code>)	149
	<code>prepare_web_skel/1</code> (pred)	149
	<code>prepare_mathjax/0</code> (pred)	149
	<code>using_mathjax/1</code> (pred)	149
23	Template Support for the HTML Backend ..	151
23.1	Usage and interface (<code>autodoc_html_template</code>)	151
23.2	Documentation on exports (<code>autodoc_html_template</code>)	151
	<code>img_url/2</code> (pred)	151
	<code>fmt_html_template/3</code> (pred)	151
24	Man Pages (man) Backend	153
24.1	Usage and interface (<code>autodoc_man</code>)	153
24.2	Documentation on multifiles (<code>autodoc_man</code>)	153
	<code>autodoc_rw_command_hook/4</code> (pred)	153
	<code>autodoc_finish_hook/1</code> (pred)	153
	<code>autodoc_gen_alternative_hook/2</code> (pred)	153
25	Filesystem Abstraction	155
25.1	Usage and interface (<code>autodoc_filesystem</code>)	155
25.2	Documentation on exports (<code>autodoc_filesystem</code>)	155
	<code>filename/1</code> (regtype)	155
	<code>basename/1</code> (regtype)	155
	<code>subtarget/1</code> (regtype)	156
	<code>file_format_name/2</code> (pred)	156
	<code>supported_file_format/1</code> (pred)	156
	<code>file_format_provided_by_backend/3</code> (pred)	156
	<code>clean_fs_db/0</code> (pred)	156
	<code>get_output_dir/2</code> (pred)	156
	<code>get_cache_dir/2</code> (pred)	156
	<code>ensure_output_dir/1</code> (pred)	156
	<code>ensure_cache_dir/1</code> (pred)	156
	<code>main_absfile_in_format/2</code> (pred)	156
	<code>main_absfile_for_subtarget/3</code> (pred)	157
	<code>absfile_for_aux/3</code> (pred)	157
	<code>absfile_for_subtarget/4</code> (pred)	157
	<code>main_output_name/2</code> (pred)	157
	<code>get_subbase/3</code> (pred)	157
	<code>absfile_to_relfile/3</code> (pred)	157
	<code>clean_all/0</code> (pred)	157
	<code>clean_docs_no_texi/0</code> (pred)	157
	<code>clean_all_temporal/0</code> (pred)	157
	<code>clean_intermediate/0</code> (pred)	158
	<code>clean_tex_intermediate/0</code> (pred)	158

26	Indexing Commands (Definition and Formatting)	159
26.1	Usage and interface (<code>autodoc_index</code>)	159
26.2	Documentation on exports (<code>autodoc_index</code>)	159
	<code>get_idxsub/2</code> (pred)	159
	<code>get_idxbase/3</code> (pred)	159
	<code>typeindex/5</code> (pred)	159
	<code>idx_get_indices/3</code> (pred)	160
	<code>is_index_cmd/1</code> (pred)	160
	<code>codetype/1</code> (pred)	160
	<code>normalize_index_cmd/3</code> (pred)	160
	<code>fmt_idx_env/7</code> (pred)	160
	<code>fmt_index/3</code> (pred)	160
27	Database of Documentation References	161
27.1	Usage and interface (<code>autodoc_refsdb</code>)	161
27.2	Documentation on exports (<code>autodoc_refsdb</code>)	161
	<code>compute_refs_and_biblio/1</code> (pred)	161
	<code>prepare_current_refs/1</code> (pred)	161
	<code>clean_current_refs/1</code> (pred)	161
	<code>sectree/1</code> (regtype)	162
	<code>sectree_resolve/3</code> (pred)	162
28	Error Messages	163
28.1	Usage and interface (<code>autodoc_errors</code>)	163
28.2	Documentation on exports (<code>autodoc_errors</code>)	163
	<code>error_text/3</code> (pred)	163
29	Resolution of Bibliographical References	165
29.1	Usage and interface (<code>autodoc_bibrefs</code>)	165
29.2	Documentation on exports (<code>autodoc_bibrefs</code>)	165
	<code>resolve_bibliography/1</code> (pred)	165
	<code>parse_commands/3</code> (pred)	165
30	Auxiliary Definitions	167
30.1	Usage and interface (<code>autodoc_aux</code>)	167
30.2	Documentation on exports (<code>autodoc_aux</code>)	167
	<code>all_vars/1</code> (pred)	167
	<code>read_file/2</code> (pred)	167
	<code>ascii_blank_lines/2</code> (pred)	167
	<code>sh_exec/2</code> (pred)	167
31	Image Handling	169
31.1	Usage and interface (<code>autodoc_images</code>)	169
31.2	Documentation on exports (<code>autodoc_images</code>)	169
	<code>locate_and_convert_image/4</code> (pred)	169
	<code>clean_image_cache/0</code> (pred)	169
	References	171
	Library/Module Index	173

Predicate/Method Index	175
Property Index	177
Regular Type Index	179
Declaration Index	181
Concept Index	183
Author Index	185
Global Index	187

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 [DEDC96], 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 115.

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 2.5 [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 2.7 [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 2.5 [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., `man1` 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 (undefined) [Installing a generated manual in a public area], page (undefined)).

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).

1.3 lpdoc usage

The following provides the different command line options available when invoking `lpdoc`. This description is intended only for advanced users which might like to use `lpdoc` in custom applications. Note that the normal way to use `lpdoc` 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/1` 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 `@version{}` command (expands to the version of the software to be documented). (Jose Morales)
- Shorter `SETTINGS.pl` files (with some rudimentary, assertion-based checking of options) (Jose Morales)
- Bug fix: `'@@ include'` and `'@@ includeverbatim'` are no longer a problem (space can be omitted) (Jose Morales)
- Added and documented a new `documentation` filetype (for some parts of the manual that contains only documentation). That avoids the old trick of declaring a fake `main/0` predicate. (Jose Morales)
- Style for subtitle added automatically (in texinfo, it is *emph*; in HTML it is normal text with smaller font). The entries in `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 `.pl` and `.lpdoc` files. (Jose Morales)
- No copyright section if no copyright comment. (Jose Morales)
- Auxiliary documentation files ending in `'_doc'` displayed incorrect names for the module (ending in `'_doc'`). E.g., `use_package(foo_doc)` was displayed instead of `use_package(foo_doc)`. Fixed. (Jose Morales)
- In `verbatim` environments, new-line characters are removed from the beginning. (Jose Morales)
- Fix wrong use of `erases/1` for clauses (which resulted in segmentation fault when documentation generation failed) (Jose Morales)
- Fixed image generation (now uses `.png` files for HTML) (Jose Morales)
- New code for text escape fixed some problems, like `'@/1'` operator not being displayed correctly in Info. (Jose Morales)
- Colors for Prolog variables (in HTML). (Jose Morales)
- Added `@begin{alert}` environment for alert messages (like *cartouche*, but in red). (Jose Morales)
- Supporting `'@''` command for umlaut, in addition to `'@.'` (Jose Morales)
- Double quotes correctly translated to HTML (Jose Morales)
- `@author` command to reference authors (changed command referring to people by `@author`, in all the documentation) (Jose Morales)
- Simplification of documentation setting files (see the documentation for further details) (Jose Morales)
- Using `open` for `lpdoc htmlview` command in MacOS X (Jose Morales)
- Adding `html` and `pdf` formats as options for emacs customization of LPdoc (`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 `-cv`, `-comment-version`, that tells `lpdoc` if the file has version comment. Formatting of `lpdoc` version comments completed. (Edison Mera)
- Improved handling of option values. Added `-d` option to `lpdoc`, that allows defining additional values in the argument. Added options `-l` and `-m` that are similar to the corresponding `lpmake` options. (Edison Mera)
- Support for in-code sections (experimental):
 - Latex-like font-lock highlight of sectioning documentation comments (`:-doc(C, "...")`, with `C` one of `title`, `section`, and `subsection`). Currently the `section` and `subsection` comments are still ignored by `LPdoc`. (Jose Morales)
- Support for mathematical notation (experimental):
 - new `@math{...}` and `@begin{displaymath}...@end{displaymath}` environments are supported (see the documentation for more details) (Jose Morales)
 - In documentation strings, single `\` must be escaped (e.g. `'@math{\\lambda}'`) (Jose Morales)
 - Supported in both the `texinfo` and `HTML` (using `MathJax`) backends. (Jose Morales)
 - Added `@defmathcmd{Cmd}{N}{Def}` and `@defmathcmd{Cmd}{Def}`, both for `texinfo` and `HTML` backends. Those `LPdoc` commands define new mathematical environments (equivalent to `\newcommand`). (Jose Morales)

Version 2.1 (2004/10/28, 16:38:17 CEST)

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

- New functionality:
 - Use of `:-doc` declarations (as a shorthand for `comment`) now allowed. (Manuel Hermenegildo)
 - Made `xdvi` viewer, `ps` viewer, and `xdvi` zoom size be parameters (the latter since new versions of `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 `dvips`. (Manuel Hermenegildo)
 - Added option `-c Target` in `lpdoc`, that treats `Target` as a separate component. (Edison Mera)
 - Added option `-f ConfigFile` in `lpdoc`, that uses the file `ConfigFile` instead the default `LPSETTINGS.pl`. (Edison Mera)
 - Added option `ascii` that generates documentation in `ascii` plain format. (Edison Mera)
 - Added `-help` option. Is equal to `-h`. (Edison Mera)
 - Added option `testsettings` to check that the settings file is correctly specified. (Edison Mera)
 - Changed `generate_html_pointer/5` by `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: lpd doc 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 PAPER TYPE can be set in the SETTINGS file which controls the format of printed output. (Manuel Hermenegildo)
 - Default pdf viewer is now `ghostview`, since recent versions handle pdf well. (Manuel Hermenegildo)
 - Changed default style sheet in order to show `<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 `.refs` file is not empty. (Manuel Hermenegildo)
 - A *copy* of the html style sheet is now included in *distributions*. Also *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 `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 `lpdoc`.

- New commands:
 - `@begin{cartouche}` and `@end{cartouche}` commands now supported.
 - `@foote` command now supported.
 - New `gmake htmlview` command (makes a running `netscape` visit the generated html manual). Suggested by Per Cederberg.
 - New `gmake distclean` command, intended for software distributions. Leaves the generated documents and eliminates *all* intermediate files (including `.texic/.texi` files).
 - Adobe `pdf` format now supported as a valid target. Unfortunately, embedded `.eps` figures are not supported at this time in pdf output.
 - The second argument of `:- comment(hide,...)` and `:- comment(doinclude,...)` declarations can now be a list of predicate names.
 - A `-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 `-u` option in `ciaoc`).
 - Now typing just `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)

Version 1.8 (1999/3/24, 21:15:33 MET)

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 `ensure_loaded/1`).
- Separated generation of the manuals from their installation. I.e., `gmake install` now does not force a `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 `-v` option allows using quieter by default operation when not debugging.
- New option `-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 `-noisoline` makes the textual explanation of the `iso/1` property not appear in the description of the usage (but the `◻ISO◻` symbol does appear)
- Two new options, `-nosysmods` and `-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 `@noindent` and `@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 `@` 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 `.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 MEST)

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

(Manuel Hermenegildo)

Version 1.5 (1998/8/23, 20:30:32 EST)

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)

Version 1.4 (1998/8/4, 19:10:35 MET DST)

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 multifile 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)

Version 1.2 (1998/6/4, 9:12:19 MET DST)

Major overall improvements... (Manuel Hermenegildo)

Version 1.1 (1998/3/31)

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 `includefact/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.

PART I - LPdoc Reference Manual



2 Generating Installing and Accessing Manuals

Author(s): Manuel Hermenegildo.

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

This section describes how to generate a manual (semi-)automatically from a set of source files using `lpdoc`, 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 `lpdoc` generate quickly the documentation of a single file it suffices to execute the command `lpdoc -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`, `pfd`, `ps`, etc. – see later). `lpdoc` 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 `lpdoc` 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 `lpdoc lpsettings` in the directory where the documentation is to be created (e.g., `doc` in the previous point). `lpdoc` 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 by default 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 select 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 docsclean` 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 14.1 [Other software packages required (`lpdoc`)], page 115), 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.

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 `bash` 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.

2.5 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 *does not fail*), standard compiler directives (such as `dynamic/1`, `op/3`, `meta_predicate/1...`), etc. When documenting a module, `lpdoc` 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 `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).

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 *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.

`lpdoc` requires these assertions and comments to be written using the `Ciao` system *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 `compatibility` directory in the `lpdoc` library). Using this library, such assertions and comments are simply ignored by the compiler. This compatibility library also allows compiling `lpdoc` itself under (C)LP systems other than the `Ciao` system under which it is developed.

2.6 Accessing on-line manuals

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

2.6.1 Accessing html manuals

Once generated, the `.html` files can be viewed using any standard WWW browser, e.g., Firefox (a command `lpdoc 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 `html` format). To make these files publicly readable on the WWW, they should be copied into a directory visible by browsers running in other machines, such as `/home/clip/public_html/lpdoc_docs`, `/usr/home/httpd/htmldocs/lpdoc_docs`, etc. As mentioned before, this is easily done by setting the `docdir` variable in the `SETTINGS.pl` file to this directory and typing `lpdoc install`.

2.6.2 Accessing info manuals

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

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

- **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 `infoindex` 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 `infoindex` 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 `cs`h 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`, 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`).

2.6.3 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`.

2.6.4 Putting it all together

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

```
setenv DOCDIR    /home/clip/public_html/lpdoc_docs
setenv INFOPATH /usr/local/info:${DOCDIR}
setenv MANPATH  ${DOCDIR}:${MANPATH}
```

Example files for inclusion in user's or common shell initialization files are included in the `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 `docformats` and `docdir` variables and performing `lpdoc install` for each installation format/directory.

2.7 Some usage tips

This section contains additional suggestions on the use of `lpdoc`.

2.7.1 Ensuring Compatibility with All Supported Target Formats

One of the nice things about `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.
- `texinfo` supports only a limited number of indices. Thus, if you select too many indices in the `SETTINGS.pl` file you may exceed `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 `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 `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 `texinfo` do not always handle it correctly.

2.7.2 Writing comments which document version/patch changes

When writing version comments (`:- 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 `pred` assertions" than "modifying file so `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 `lpdoc` will not read). An alternative and quite useful solution is to put such internal comments in *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 *version* changes (e.g., 1.1#2 to 1.2#0). Selecting the appropriate options in `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 *internals manual*).

2.7.3 Documenting Libraries and/or Applications

As mentioned before, for each a `.pl` file, `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 `lpdoc` manual. Some typical combinations are:

- *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.
- *Main file is an application, no components*: A manual of a simple application.

- *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.
- *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.
- *Main file is a (pseudo-)application, components are libraries:* A manual of a complex library made up of smaller libraries (for example, the `Prolog` library). The (pseudo-)application file contains the introductory material (title, version, etc.). Each chapter describes a particular library.
- *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.

2.7.4 Documenting files which are not modules

Sometimes it is difficult for `lpdoc` to distinguish include files and Ciao packages from normal *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 `include/1` or `use_package/1` declarations instead of `ensure_loaded/1`) and effect (since they are included, they 'export' operators, declarations, etc.), and should typically be documented differently. There is a special `doc/2` declaration (`:- 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 `ensure_loaded/1`). Adding this declaration will, for example, avoid spurious documentation of the declarations in the `assertions` package themselves when this package is included in a user file.

2.7.5 Splitting large documents into parts

As mentioned before, in `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 second argument of the `:- doc filetype, ...` declaration mentioned above designed for this purpose. The special *filetype* value `part` can be used to flag that the file in which it appears should be documented as the start of one of the major *parts in a large document*. In order to introduce such a part, a `.pl` file with a declaration `:- doc filetype, part` should be inserted in the sequence of files that make up the `components` variable of the `SETTINGS.pl` file at each point in which a major part starts. The `:- doc title, "..."` declaration of this file will be used as the part title, and the `:- doc module, "..."` declaration text will be used as the introduction to the part.

2.7.6 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.

2.7.7 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 `m1.pl`, this can be done by calling the module containing the documentation `m1_doc.pl`. This `m1_doc.pl` file is the one that will be included in the `lpdoc SETTINGS.pl` file, instead of `m1.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 `m1`.

2.7.8 Generating auxiliary files (e.g. READMEs)

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

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}
...
").
```

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:

```
:- 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`.

2.8 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 `.pl` file, this means (in `make`'s language) that it *cannot find the corresponding .pl 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.

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 on-line 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:

```
:- module(...,[assertions]).
:- use_package(assertions).
```

- **Exports:**

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

- **Imports:**

- *System library modules:*
`strings`.
- *Internal (engine) modules:*
`term_basic`, `arithmetic`, `atomic_basic`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_compare`, `term_typing`, `hiord_rt`, `debugger_support`.
- *Packages:*
`prelude`, `nonpure`, `dgc`, `assertions`, `regtypes`, `fsyntax`.

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)`

`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.

- `@item{itemtext}`
marks the beginning of a *new item in description list*, and contains the header for the item.
- `@end{description}`
marks the end of a description list.
- `@begin{verbatim}`
marks the beginning of *fixed format text*, such as a program example. A fixed-width, typewriter-like font is used.
- `@end{verbatim}`
marks the end of formatted text.
- `@begin{cartouche}`
marks the beginning of a section of text in a *framed box*, with round corners.
- `@end{cartouche}`
marks the end of a section of text in a framed box.
- `@begin{alert}`
marks the beginning of a section of text in a *framed box*, for alert messages.
- `@end{alert}`
marks the end of the alert message.
- `@section{text}`
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.
- `@subsection{text}`
starts a *subsection* whose title is *text*.
- `@subsubsection{text}`
starts a *subsubsection* whose title is *text*.
- `@footnote{text}`
places *text* in a *footnote*.
- `@hfill` introduces horizontal filling space (may be ignored in certain formats).
- `@bf{text}` *text* will be formatted in *bold face* or any other *strong face*.
- `@em{text}` *text* will be formatted in *italics face* or any other *emphasis face*.
- `@tt{text}` *text* will be formatted in a *fixed-width font* (i.e., *typewriter-like font*).
- `@key{key}`
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.
- `@sp{N}` generates *N blank lines* of space. Forces also a paragraph break.
- `@p` forces a *paragraph break*, in the same way as leaving one or more blank lines.
- `@noindent`
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.

`@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.

`@href{URL}`

introduces at point a reference to the *Universal Resource Locator* (i.e., a *WWW address* 'URL').

`@href{URL}{text}`

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}`

marks the end of the (long) formula.

`@defmathcmd{cmd}{n}{def}`

defines the math command *cmd*, taking *n* arguments, which is expanded as *def*. Arguments are denotated as #1, ..., #*n* inside *def*.

`@defmathcmd{cmd}{def}`

defines the math command *cmd*, which is expanded as *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.

`@include{filename}`

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.

`@includeverbatim{filename}`

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 @'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 `@includeverbatim{filename}` with `@begin{verbatim}` and `@end{verbatim}`.

`@includefact{factname}`

it is assumed that the file being documented contains a fact of the predicate *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.).

`@includedef{predname}`

it is assumed that the file being documented contains a definition for the predicate *predname*. The clauses defining this predicate will be included in-line, in verbatim mode, as if they were part of the documentation string.

`@image{epsfile}`

including an image at point, contained in file *epsfile*. The *image file* should be in *encapsulated postscript* format.

`@image{epsfile}{width}{height}` same as above, but *width* and *height* should be integers which provide a size (in points) to which the image will be scaled.

- **Accents and special characters:**

The following commands can be used to insert *accents* and *special characters*.

@‘{o}	⇒	ò
@’{o}	⇒	ó
@~{o}	⇒	ô
@..{o}	⇒	ö
@"{o}	⇒	ö
@~{o}	⇒	õ
@={o}	⇒	ō
@.{o}	⇒	ò
@u{o}	⇒	ů
@v{o}	⇒	ǎ
@H{o}	⇒	Ǿ
@t{oo}	⇒	öö
@c{o}	⇒	ç
@d{o}	⇒	đ
@b{o}	⇒	ḃ
@oe	⇒	œ
@OE	⇒	Œ
@ae	⇒	æ
@AE	⇒	Æ
@aa	⇒	å
@AA	⇒	Å
@o	⇒	ø
@O	⇒	Ø
@l	⇒	ł
@L	⇒	Ł
@ss	⇒	ß
@?	⇒	¿
@!	⇒	¡
@i	⇒	ı
@j	⇒	Ј
@copyright	⇒	©
@iso	⇒	◻•ISO•◻
@bullet	⇒	•
@result	⇒	⇒

Usage: stringcommand(CO)

CO is a structure denoting a command that is admissible in strings inside assertions.

version_descriptor/1:

REGTYPE

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)`

`Descriptor` is a complete version descriptor.

filetype/1:

REGTYPE

Intended uses of a file:

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

Usage: `filetype(Type)`

`Type` describes the intended use of a file.

doc_id_type/3:

PREDICATE

No further documentation available for this predicate.

3.3 Documentation on internals (comments)

doc/2:

DECLARATION

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).`

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
```

```
( = /2)
```

```
TitleText is a documentation string.
```

```
( docstring/1)
```

Usage 2: `:- doc(CommentType,SubtitleText).`

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

Example:

```
:- 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 ( = /2)
SubtitleText is a documentation string. ( docstring/1)
```

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

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:

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

– *The following properties should hold upon exit:*

```
CommentType=subtitle_extra ( = /2)
SubtitleText is a documentation string. ( docstring/1)
```

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

The name of the logo image for the manual.

– *The following properties should hold upon exit:*

```
CommentType=logo ( = /2)
SubtitleText is any term. ( term/1)
```

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

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:

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

– *The following properties should hold upon exit:*

```
CommentType=author ( = /2)
AuthorText is a documentation string. ( docstring/1)
```

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

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

Example:

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

– *The following properties should hold upon exit:*

```
CommentType=address ( = /2)
Text is a documentation string. ( docstring/1)
```

Usage 7: `:- doc(CommentType,AckText).`

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                                     (= /2)
  AckText is a documentation string.                 ( docstring/1)
```

Usage 8: `:- doc(CommentType,CopyrightText).`

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                               (= /2)
  CopyrightText is a documentation string.           ( docstring/1)
```

Usage 9: `:- doc(CommentType,SummaryText).`

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                                 (= /2)
  SummaryText is a documentation string.             ( docstring/1)
```

Usage 10: `:- doc(CommentType,CommentText).`

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

Usage 11: `:- doc(CommentType,CommentText).`

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.

Example:

```
:- 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).`

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:

```
:- 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).`

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:

```
:- 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).`

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

Example:

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

- *The following properties should hold upon exit:*

```

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

```

Usage 15: `:- doc(CommentType,SubSubSection).`

Insert a *program subsection* with name `SubSubSection` (see *program section* command for more details).

Example:

```
:- doc(subsubsection,"Auxiliary Definitions").
```

- *The following properties should hold upon exit:*

```

CommentType=subsubsection                            ( = /2)
SubSubSection is a documentation string.           ( docstring/1)

```

Usage 16: `:- doc(PredName,CommentText).`

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:


```
:- 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).
```

(`predname/1`)

`CommentText` is a *documentation string*.

(`docstring/1`)

Usage 17: `:- doc(CommentType,CommentText).`

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`

(`= /2`)

`CommentText` is a *documentation string*.

(`docstring/1`)

Usage 18: `:- doc(Version,CommentText).`

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(1*1+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.

(`version_descriptor/1`)

`CommentText` is a *documentation string*.

(`docstring/1`)

Usage 19: `:- doc(CommentType,VersionMaintenanceType).`

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 `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 *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.

- *The following properties should hold upon exit:*

```

CommentType=version_maintenance           (= /2)
VersionMaintenanceType a type of version maintenance for a file.      (
version_maintenance_type/1)

```

Usage 20: `:- doc(CommentType,PredName).`

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 `:- doc(doinclude,PredName).` forces documentation for predicate (or type, property, function, ...) `PredName` to be included even if `PredName` is not exported. Also, if `PredName` is reexported from another module, a declaration `:- doc(doinclude,PredName).` will force the documentation for `PredName` to appear directly in this module.

Example:

```

:- doc(doinclude,p/3).

```

- *The following properties should hold upon exit:*

```

CommentType=doinclude                     (= /2)
PredName is a Name/Arity structure denoting a predicate name:
    predname(P/A) :-
        atm(P),
        nnegint(A).
                                           ( predname/1)

```

Usage 21: `:- doc(CommentType,PredName).`

A different usage which allows the second argument of `:- doc(doinclude,...)` to be a list of predicate names.

- *The following properties should hold upon exit:*

```

CommentType=doinclude                     (= /2)
PredName is a list of prednames.          ( list/2)

```

Usage 22: `:- doc(CommentType,PredName).`

This is similar to the previous usage but has the opposite effect: it signals that an exported predicate should *not* be included in the documentation.

Example:

```

:- doc(hide,p/3).

```

- *The following properties should hold upon exit:*

```

CommentType=hide                          (= /2)
PredName is a Name/Arity structure denoting a predicate name:

```

```

predname(P/A) :-
    atm(P),
    nnegint(A).

```

(predname/1)

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

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.

```

(= /2)
(list/2)

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

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:

```

:- 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 2.7.5 [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.

```

(= /2)
(filetype/1)

Usage 25: :- doc(CommentType,FileName).

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:

```

:- doc(nodoc,assertions).

```

- *The following properties should hold upon exit:*

```

CommentType=nodoc
FileName is an atom.

```

(= /2)
(atm/1)

version_number/1:

REGTYPE

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

```

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

```

Usage: `version_number(Version)`

`Version` is a complete version number

ymd_date/1:

REGTYPE

A Year/Month/Day structure denoting a date:

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

Usage: `ymd_date(Date)`

`Date` is a Year/Month/Day structure denoting a date.

time_struct/1:

REGTYPE

A structure containing time information:

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

Usage: `time_struct(Time)`

`Time` contains time information.

version_maintenance_type/1:

REGTYPE

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)`

`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:

```
:- 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], pred/2 [1150,xfx], prop/1 [1150,fx], prop/2 [1150,xfx], modedef/1 [1150,fx], calls/1 [1150,fx], calls/2 [1150,xfx], success/1 [1150,fx], success/2 [1150,xfx], 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.
```

- **Imports:**

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

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 `assrt_body/1`.

More than one of these assertions may appear per predicate, in which case each one represents a possible “mode” of use (usage) of the predicate. The exact scope of the usage is defined by the properties given for calls in the body of each assertion (which should thus distinguish the different usages intended). All of them together cover all possible modes of usage.

For example, the following assertions describe (all the and the only) modes of usage of predicate `length/2` (see lists):

```
:- 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. (`assrt_body/1`)

pred/2: DECLARATION

This assertion is similar to a `pred/1` assertion but it is explicitly qualified. Non-qualified `pred/1` assertions are assumed the qualifier `check`.

Usage: :- AssertionStatus pred 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`)

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_assrt_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. (`assrt_status/1`)

AssertionBody is a call assertion body. (`c_assrt_body/1`)

calls/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it only provides information about the calls to a predicate. If one or several calls assertions are given they are understood to describe all possible calls to the predicate.

For example, the following assertion describes all possible calls to predicate `is/2` (see `arithmetic`):

```
:- calls is(term,arithexpression).
```

Usage: :- calls AssertionBody.

- *The following properties should hold at call time:*

AssertionBody is a call assertion body. (`c_assrt_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. (`assrt_status/1`)

`AssertionBody` is a call assertion body. (`c_assrt_body/1`)

success/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it only provides information about the answers to a predicate. The described answers might be conditioned to a particular way of calling the predicate.

For example, the following assertion specifies the answers of the `length/2` predicate *if* it is called as in the first mode of usage above (note that the previous `pred` assertion already conveys such information, however it also compelled the predicate calls, while the `success` assertion does not):

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

Usage: `:- success AssertionBody.`

- *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

success/2: DECLARATION

`success` assertion This assertion is similar to a `success/1` assertion but it is explicitly qualified with an assertion status. The status of non-qualified `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. (`assrt_status/1`)

`AssertionBody` is a predicate assertion body. (`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:

```
:- test length(L,N) : ( L = [1,2,5,2] ) => ( N = 4 ).
```

Usage: `:- test AssertionBody.`

- *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

test/2: DECLARATION

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: `:- AssertionStatus test AssertionBody.`

- *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

comp/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it only provides information about the global execution properties of a predicate (note that such kind of information is also conveyed by `pred` assertions). The described properties might be conditioned to a particular way of calling the predicate.

For example, the following assertion specifies that the computation of `append/3` (see `lists`) will not fail *if* it is called as described (but does not compel the predicate to be called that way):

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

Usage: `:- comp AssertionBody.`

- *The following properties should hold at call time:*

`AssertionBody` is a `comp` assertion body. (`g_assrt_body/1`)

comp/2: DECLARATION

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: `:- AssertionStatus comp AssertionBody.`

- *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

`AssertionBody` is a `comp` assertion body. (`g_assrt_body/1`)

prop/1: DECLARATION

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. (`assrt_body/1`)

prop/2:

DECLARATION

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. (`assrt_status/1`)

AssertionBody is an assertion body. (`assrt_body/1`)

entry/1:

DECLARATION

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. (`c_assrt_body/1`)

exit/1:

DECLARATION

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.:

```
:- 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. (`s_assrt_body/1`)

exit/2: DECLARATION

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/1`)

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

modedef/1: DECLARATION

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/1`)

decl/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it is used for declarations instead than for predicates.

Usage: `:- decl AssertionBody.`

- *The following properties should hold at call time:*

`AssertionBody` is an assertion body. (`assrt_body/1`)

decl/2: DECLARATION

This assertion is similar to a `decl/1` assertion but it is explicitly qualified. Non-qualified `decl/1` assertions are assumed the qualifier `check`.

Usage: `:- AssertionStatus decl 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`)

doc/2: DECLARATION**Usage:** `:- doc(Pred, Comment).`Documentation . This assertion provides a text `Comment` for a given predicate `Pred`.

- *The following properties should hold at call time:*

`Pred` is a head pattern. (`head_pattern/1`)`Comment` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments. (`docstring/1`)**comment/2:** DECLARATION**Usage:** `:- comment(Pred, Comment).`An alias for `doc/2` (deprecated, for compatibility with older versions).

- *The following properties should hold at call time:*

`Pred` is a head pattern. (`head_pattern/1`)`Comment` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments. (`docstring/1`)

4.5 Documentation on exports (`assertions_doc`)

check/1: PREDICATE**Usage:** `check(PropertyConjunction)`This assertion provides information on a clause program point (position in the body of a clause). Calls to a `check/1` assertion can appear in the body of a clause in any place where a literal can normally appear. The property defined by `PropertyConjunction` should hold in all the run-time stores corresponding to that program point. See also Chapter 12 [Run-time checking of assertions], page 111.

- *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. (`property_conjunction/1`)**trust/1:** PREDICATE**Usage:** `trust(PropertyConjunction)`This assertion also provides information on a clause program point. It is identical syntactically to a `check/1` assertion. However, the properties stated are not taken as something to be checked but are instead *trusted* by the compiler. While the compiler may in some cases detect an inconsistency between a `trust/1` assertion and the program, in all other cases the information given in the assertion will be taken to be true. As a result, if these assertions are erroneous they can introduce bugs in programs. Thus, `trust/1` assertions should be written with care.

An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program (either because the information is not

present or because the analyzer being used is not precise enough). In particular, providing information on external predicates which may not be accessible at the time of compiling the module can greatly improve the precision of the analyzer. This can be easily done with trust assertion.

- *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

true/1:

PREDICATE

Usage: `true(PropertyConjunction)`

This assertion is identical syntactically to a `check/1` assertion. However, the properties stated have been proved to hold by the analyzer. Thus, these assertions often represent the analyzer output.

- *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

false/1:

PREDICATE

Usage: `false(PropertyConjunction)`

This assertion is identical syntactically to a `check/1` assertion. However, the properties stated have been proved not to hold by the analyzer. Thus, these assertions often represent the analyzer output.

- *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

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:**
`:- use_module(library(assertions_props)).`
- **Exports:**
 - *Properties:*
`head_pattern/1, nobody/1, docstring/1.`
 - *Regular Types:*
`assrt_body/1, complex_arg_property/1, property_conjunction/1, property_starterm/1, complex_goal_property/1, dictionary/1, c_assrt_body/1, s_assrt_body/1, g_assrt_body/1, assrt_status/1, assrt_type/1, predfunctor/1, propfunctor/1.`
- **Imports:**
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, dcg, assertions, regtypes.`

5.2 Documentation on exports (`assertions_props`)

assrt_body/1:

REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `pred/1`, `decl/1`, etc. assertions. Such a body is of the form:

$$\text{Pr } [:: \text{DP}] \text{ } [:\text{ CP}] \text{ } [=> \text{AP}] \text{ } [+ \text{GP}] \text{ } [\# \text{CO}]$$

where (fields between [...] are optional):

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `DP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which expresses properties which are compatible with the predicate, i.e., instantiations made by the predicate are *compatible* with the properties in the sense that applying the property at any point 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: `assrt_body(X)`

X is an assertion body.

head_pattern/1:

PROPERTY

A head pattern can be a predicate name (functor/arity) (`predname/1`) or a term. Thus, both `p/3` and `p(A,B,C)` are valid head patterns. In the case in which the head pattern is a term, each argument of such a term can be:

- A variable. This is useful in order to be able to refer to the corresponding argument positions by name within properties and in comments. Thus, `p(Input,Parameter,Output)` is a valid head pattern.
- A variable, as above, but preceded by a “ mode.” This mode determines in a compact way certain call or answer properties. For example, the head pattern `p(Input,+Parameter,Output)` is valid, as long as `+/1` is declared as a mode.

Acceptable modes are documented in `library(basicmodes)` and `library(isomodes)`. User defined modes are documented in `modedef/1`.

- Any term. In this case this term determines the instantiation state of the corresponding argument position of the predicate calls to which the assertion applies.
- A ground term preceded by a “ mode.” The ground term determines a property of the corresponding argument. The mode determines if it applies to the calls and/or the successes. The actual property referred to is that given by the term but with one more argument added at the beginning, which is a new variable which, in a rewriting of the head pattern, appears at the argument position occupied by the term. For example, the head pattern `p(Input,+list(int),Output)` is valid for mode `+/1` defined in `library(isomodes)`, and equivalent in this case to having the head pattern `p(Input,A,Output)` and stating that the property `list(A,int)` holds for the calls of the predicate.
- Any term preceded by a “ mode.” In this case, only one variable is admitted, it has to be the first argument of the mode, and it represents the argument position. I.e., it plays the role of the new variable mentioned above. Thus, no rewriting of the head pattern is performed in this case. For example, the head pattern `p(Input,+(Parameter,list(int)),Output)` is valid for mode `+/2` defined in `library(isomodes)`, and equivalent in this case to having the head pattern `p(Input,Parameter,Output)` and stating that the property `list(Parameter,int)` holds for the calls of the predicate.

Usage: `head_pattern(Pr)`

Pr is a head pattern.

complex_arg_property/1: REGTYPE

`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)`

`Props` is a (possibly empty) complex argument property

property_conjunction/1: REGTYPE

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)`

`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: REGTYPE

This type defines a second, compact format in which properties can be expressed in the bodies of assertions. A `property_starterm/1` is a term whose main functor is `*/2` and, when it appears in an assertion, the number of terms joined by `*/2` is exactly the arity of the predicate it refers to. A similar series of properties as in `property_conjunction/1` appears, but the arity of each property is one less: the argument position to which they refer (first argument) is left out and determined by the position of the property in the `property_starterm/1`. The idea is that each element of the `*/2` term corresponds to a head argument position. Several properties can be assigned to each argument position by grouping them in curly brackets. The following is an example of a complex property in this format:

- `integer * list(integer)`: the first argument of the procedure (or function, or ...) has the property `integer/1` and the second one has the property `list/2`, with second argument `integer`.
- `{integer,var} * list(integer)`: the first argument of the procedure (or function, or ...) has the properties `integer/1` and `var/1` and the second one has the property `list/2`, with second argument `integer`.

Usage: `property_starterm(Props)`

`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: REGTYPE

`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

```
:- comp var(A) + not_further_inst(A).
```

has property `not_further_inst/1` as goal property, and establishes that in all executions of `var(A)` it should hold that `not_further_inst(var(A),A)`.

Usage: `complex_goal_property(Props)`

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: PROPERTY

Usage: `nabody(ABody)`

`ABody` is a normalized assertion body.

dictionary/1: REGTYPE

Usage: `dictionary(D)`

`D` is a dictionary of variable names.

c_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `call/1`, `entry/1`, etc. assertions. The following are admissible:

```
Pr : CP [# CO]
```

where (fields between [...] are optional):

- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `c_assrt_body(X)`

`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:

```

Pr : CP => AP # CO
Pr : CP => AP
Pr => AP # CO
Pr => AP

```

where:

- Pr is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- 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.
- 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)`

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:

```

Pr : CP + GP # CO
Pr : CP + GP
Pr + GP # CO
Pr + GP

```

where:

- Pr is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- CP is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- 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.
- 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)`

X is a comp assertion body.

assrt_status/1: REGTYPE

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)`

X is an acceptable status for an assertion.

assrt_type/1: REGTYPE

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)`

X is an admissible kind of assertion.

predfunctor/1: REGTYPE

Usage: `predfunctor(X)`

X is a type of assertion which defines a predicate.

propfunctor/1: REGTYPE

Usage: `propfunctor(X)`

X is a type of assertion which defines a *property*.

docstring/1: PROPERTY

Usage: `docstring(String)`

`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:

```
:- regtype tree(X) # "X is a tree."
```

instead of the more cumbersome:

```
:- 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 assertions related to computations (i.e., `comp/1`) are called *properties of computations*. Different considerations apply when writing a property of the former or of the latter kind.

Consider a definition of the predicate `string_concat/3` which concatenates two character strings (represented as lists of ASCII codes):

```
string_concat([],L,L).
string_concat([X|Xs],L,[X|NL]):- string_concat(Xs,L,NL).
```

Assume that we would like to state in an assertion that each argument “is a list of integers.” However, we must decide which one of the following two possibilities we mean exactly: “the argument is *instantiated* to a list of integers” (let us call this property `instantiated_to_intlist/1`), or “if any part of the argument is instantiated, this instantiation must be compatible with the argument being a list of integers” (we will call this property `compatible_with_intlist/1`). For example, `instantiated_to_intlist/1` should be true for the terms `[]` and `[1,2]`, but should not for `X`, `[a,2]`, and `[X,2]`. In turn, `compatible_with_intlist/1` should be true for `[]`, `X`, `[1,2]`, and `[X,2]`, but should not be for `[X|1]`, `[a,2]`, and `1`. We refer to properties such as `instantiated_to_intlist/1` above as *instantiation properties* and to those such as `compatible_with_intlist/1` as *compatibility properties* (corresponding to the traditional notions of “instantiation types” and “compatibility types”).

It turns out that both of these notions are quite useful in practice. In the example above, we probably would like to use `compatible_with_intlist/1` to state that on success of `string_concat/3` all three argument must be compatible with lists of integers in an assertion like:

```
:- success string_concat(A,B,C) => ( compatible_with_intlist(A),
                                   compatible_with_intlist(B),
                                   compatible_with_intlist(C) ).
```

With this assertion, no error will be flagged for a call to `string_concat/3` such as `string_concat([20],L,R)`, which on success produces the resulting atom `string_concat([20],L,[20|L])`, but a call `string_concat([],a,R)` would indeed flag an error.

On the other hand, and assuming that we are running on a Prolog system, we would probably like to use `instantiated_to_intlist/1` for `sumlist/2` as follows:

```
:- 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 111) 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:

```
:- 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 run-time 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.`
- **Imports:**
 - *System library modules:*
`assertions/assertions_props.`
 - *Internal (engine) modules:*
`term_basic.`
 - *Packages:*
`prelude, assertions, pure.`

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:

$$p(x, v_1, \dots, v_n) \quad :- \quad \text{body}_1, \dots, \text{body}_k.$$

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, $p(f(X, Y)) \quad :- \dots$ is valid, but $p(f(X, X)) \quad :- \dots$ 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, \dots, 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. $q(y, t_1, \dots, t_m)$ where $m \geq 0$, q/m is a *parametric type functor*, not in the set of functors $=/2, \wedge/2, ./3$.
 t_1, \dots, 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 63.

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`)

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, rtcheck/1, rtcheck/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.

- **Imports:**

- *System library modules:*

assertions/native_props, terms_check.

- *Internal (engine) modules:*

term_basic, arithmetic, atomic_basic, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

- *Packages:*

prelude, nonpure, assertions, nortchecks, nativeprops.

7.2 Documentation on exports (basic_props)

term/1:

REGTYPE

The most general type (includes all possible terms).

(True) Usage: term(X)

X is any term.

- *The following properties hold globally:*

This predicate is understood natively by CiaoPP.

(native/1)

General properties:

True: term(X)

- *The following properties hold globally:*

term(X) is side-effect free.

(sideff/2)

True: `term(X)`

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

True: `term(X)`

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

int/1: REGTYPE

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

(True) Usage: `int(T)`

T is an integer.

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

General properties:

True: `int(T)`

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

True: `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`)

Trust: `int(T)`

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

Trust:

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

nnegint/1: REGTYPE

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

(True) Usage: `nnegint(T)`

T is a non-negative integer.

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

General properties:

True: `nnegint(T)`

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

True: `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`)

Trust: `nnegint(T)`

- *The following properties hold upon exit:*
`T` is a non-negative integer. (`nnegint/1`)

Trust:

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

flt/1:

REGTYPE

The type of floating-point numbers. The range of floats is the one provided by the C `double` type, typically $[4.9\text{e-}324, 1.8\text{e+}308]$ (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.Nan`

(True) Usage: `flt(T)`

`T` is a float.

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

General properties:

True: `flt(T)`

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

True: `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`)

Trust: `flt(T)`

- *The following properties hold upon exit:*
`T` is a float. (`flt/1`)

Trust:

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

num/1:	REGTYPE
The type of numbers, that is, integer or floating-point.	
(True) Usage: num(T)	
T is a number.	
– <i>The following properties hold globally:</i>	
This predicate is understood natively by CiaoPP. (native/1)	
General properties:	
True: num(T)	
– <i>The following properties hold globally:</i>	
num(T) is side-effect free.	(sideeff/2)
num(T) is binding insensitive.	(bind_ins/1)
True: num(T)	
– <i>If the following properties hold at call time:</i>	
T is currently a term which is not a free variable.	(nonvar/1)
<i>then the following properties hold globally:</i>	
num(T) is evaluable at compile-time.	(eval/1)
All calls of the form num(T) are deterministic.	(is_det/1)
Trust: num(T)	
– <i>The following properties hold upon exit:</i>	
T is a number.	(num/1)
Trust:	
– <i>The following properties hold globally:</i>	
Indicates the type of test that a predicate performs. Required by the nonfailure analysis. (test_type/2)	
atm/1:	REGTYPE
The type of atoms, or non-numeric constants. The size of atoms is unbound.	
(True) Usage: atm(T)	
T is an atom.	
– <i>The following properties hold globally:</i>	
This predicate is understood natively by CiaoPP. (native/1)	
General properties:	
True: atm(T)	
– <i>The following properties hold globally:</i>	
atm(T) is side-effect free.	(sideeff/2)
True: atm(T)	
– <i>If the following properties hold at call time:</i>	
T is currently a term which is not a free variable.	(nonvar/1)
<i>then the following properties hold globally:</i>	
atm(T) is evaluable at compile-time.	(eval/1)
All calls of the form atm(T) are deterministic.	(is_det/1)
Trust: atm(T)	

- *The following properties hold upon exit:*

T is an atom. (atm/1)

Trust:

- *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analysis. (test_type/2)

struct/1:

REGTYPE

The type of compound terms, or terms with non-zeroary functors. By now there is a limit of 255 arguments.

(True) Usage: struct(T)

T is a compound term.

- *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (native/1)

General properties:

True: struct(T)

- *The following properties hold globally:*

struct(T) is side-effect free. (sideeff/2)

True: struct(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:

struct(T) is evaluable at compile-time. (eval/1)

Trust: struct(T)

- *The following properties hold upon exit:*

T is a compound term. (struct/1)

gnd/1:

REGTYPE

The type of all terms without variables.

(True) Usage: gnd(T)

T is ground.

- *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (native/1)

General properties:

True: gnd(T)

- *The following properties hold globally:*

gnd(T) is side-effect free. (sideeff/2)

True: gnd(T)

- *If the following properties hold at call time:*

T is currently ground (it contains no variables). (ground/1)

then the following properties hold globally:

gnd(T) is evaluable at compile-time. (eval/1)

All calls of the form gnd(T) are deterministic. (is_det/1)

Trust: `gnd(T)`

- *The following properties hold upon exit:*
T is ground. (`gnd/1`)

Trust:

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

gndstr/1: REGTYPE

(**True**) **Usage:** `gndstr(T)`

T is a ground compound term.

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

General properties:

True: `gndstr(T)`

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

True: `gndstr(T)`

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

Trust: `gndstr(T)`

- *The following properties hold upon exit:*
T is a ground compound term. (`gndstr/1`)

constant/1: REGTYPE

(**True**) **Usage:** `constant(T)`

T is an atomic term (an atom or a number).

General properties:

True: `constant(T)`

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

True: `constant(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:
`constant(T)` is evaluable at compile-time. (`eval/1`)
All calls of the form `constant(T)` are deterministic. (`is_det/1`)

Trust: `constant(T)`

- *The following properties hold upon exit:*
T is an atomic term (an atom or a number). (`constant/1`)

callable/1: REGTYPE

(True) Usage: callable(T)

T is a term which represents a goal, i.e., an atom or a structure.

General properties:

True: callable(T)

– *The following properties hold globally:*

callable(T) is side-effect free. (sideff/2)

True: callable(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:

callable(T) is evaluable at compile-time. (eval/1)

All calls of the form callable(T) are deterministic. (is_det/1)

Trust: callable(T)

– *The following properties hold upon exit:*

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

operator_specifier/1: REGTYPE

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

xfx Infix, non-associative: it is a requirement that both of the two subexpressions which are the arguments of the operator must be of *lower* precedence than the operator itself.

xfy Infix, right-associative: only the first (left-hand) subexpression must be of lower precedence; the right-hand subexpression can be of the *same* precedence as the main operator.

yfx Infix, left-associative: same as above, but the other way around.

fx Prefix, non-associative: the subexpression must be of *lower* precedence than the operator.

fy Prefix, associative: the subexpression can be of the *same* precedence as the operator.

xf Postfix, non-associative: the subexpression must be of *lower* precedence than the operator.

yf Postfix, associative: the subexpression can be of the *same* precedence as the operator.

(True) Usage: operator_specifier(X)

X specifies the type and associativity of an operator.

General properties:

True: operator_specifier(X)

– *The following properties hold globally:*

operator_specifier(X) is side-effect free. (sideff/2)

True: 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)

Trust: `operator_specifier(T)`

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

list/1: REGTYPE

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

```
list([]).
list([_1|L]) :-
    list(L).
```

(True) Usage: `list(L)`

L is a list.

General properties:

True: `list(L)`

- *The following properties hold globally:*
`list(L)` is side-effect free. (sideeff/2)

True: `list(L)`

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

Trust: `list(T)`

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

list/2: REGTYPE

`list(L,T)`

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

(True) Usage: `list(L,T)`

L is a list of T s.

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

General properties:

True: `list(L,T)`

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

True: `list(L,T)`

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

Trust: `list(X,T)`

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

nlist/2:

REGTYPE

(True) Usage: `nlist(L,T)`

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 `nlist/1` type

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

General properties:

True: `nlist(L,T)`

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

True: `nlist(L,T)`

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

Trust: `nlist(X,T)`

- *The following properties hold upon exit:*
 - X is any term. (`term/1`)

member/2:

PROPERTY

(True) Usage: `member(X,L)`

X is an element of L.

General properties:

True: `member(X,L)`

- *The following properties hold globally:*
 - `member(X,L)` is side-effect free. (`sideff/2`)
 - `member(X,L)` is binding insensitive. (`bind_ins/1`)

True: `member(X,L)`

- *If the following properties hold at call time:*
 - L is a list. (`list/1`)
- then the following properties hold globally:*
 - `member(X,L)` is evaluable at compile-time. (`eval/1`)

Trust: `member(_X,L)`

- *The following properties hold upon exit:*

L is a list. (`list/1`)

Trust: `member(X,L)`

- *If the following properties hold at call time:*

L is currently ground (it contains no variables). (`ground/1`)

then the following properties hold upon exit:

X is currently ground (it contains no variables). (`ground/1`)

sequence/2:

REGTYPE

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

(True) Usage: `sequence(S,T)`

S is a sequence of Ts.

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

General properties:

True: `sequence(S,T)`

- *The following properties hold globally:*

`sequence(S,T)` is side-effect free. (`sideff/2`)

True: `sequence(S,T)`

- *If the following properties hold at call time:*

S is currently ground (it contains no variables). (`ground/1`)

T is currently ground (it contains no variables). (`ground/1`)

then the following properties hold globally:

`sequence(S,T)` is evaluable at compile-time. (`eval/1`)

Trust: `sequence(E,T)`

- *The following properties hold upon exit:*

E is currently a term which is not a free variable. (`nonvar/1`)

T is currently ground (it contains no variables). (`ground/1`)

sequence_or_list/2:

REGTYPE

(True) Usage: `sequence_or_list(S,T)`

S is a sequence or list of Ts.

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

General properties:

True: `sequence_or_list(S,T)`

- *The following properties hold globally:*

`sequence_or_list(S,T)` is side-effect free. (`sideff/2`)

True: `sequence_or_list(S,T)`

- *If the following properties hold at call time:*
 - S is currently ground (it contains no variables). (ground/1)
 - T is currently ground (it contains no variables). (ground/1)
 - then the following properties hold globally:*
 - sequence_or_list(S,T) is evaluable at compile-time. (eval/1)
- Trust:** sequence_or_list(E,T)
- *The following properties hold upon exit:*
 - E is currently a term which is not a free variable. (nonvar/1)
 - T is currently ground (it contains no variables). (ground/1)

character_code/1: REGTYPE

(True) Usage: character_code(T)

T is an integer which is a character code.

General properties:

True: character_code(T)

- *The following properties hold globally:*
- character_code(T) is side-effect free. (sideeff/2)

True: character_code(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:*
- character_code(T) is evaluable at compile-time. (eval/1)

Trust: character_code(I)

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

string/1: REGTYPE

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

(True) Usage: string(T)

T is a string (a list of character codes).

General properties:

True: string(T)

- *The following properties hold globally:*
- string(T) is side-effect free. (sideeff/2)

True: string(T)

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

Trust: string(T)

- *The following properties hold upon exit:*

T is a string (a list of character codes). (string/1)

num_code/1: REGTYPE

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: REGTYPE

(**True**) **Usage:** predname(P)

P is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
    atm(P),
    nnegint(A).
```

General properties:

True: predname(P)

- *The following properties hold globally:*

predname(P) is side-effect free. (sideeff/2)

True: predname(P)

- *If the following properties hold at call time:*

P is currently ground (it contains no variables). (ground/1)

then the following properties hold globally:

predname(P) is evaluable at compile-time. (eval/1)

Trust: predname(P)

- *The following properties hold upon exit:*

P is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
    atm(P),
    nnegint(A).
```

(predname/1)

atm_or_atm_list/1: REGTYPE

(**True**) **Usage:** atm_or_atm_list(T)

T is an atom or a list of atoms.

General properties:

True: atm_or_atm_list(T)

- *The following properties hold globally:*

atm_or_atm_list(T) is side-effect free. (sideeff/2)

True: atm_or_atm_list(T)

- *If the following properties hold at call time:*
T is currently ground (it contains no variables). (`ground/1`)
then the following properties hold globally:
`atm_or_atm_list(T)` is evaluable at compile-time. (`eval/1`)
- Trust:** `atm_or_atm_list(T)`
- *The following properties hold upon exit:*
T is an atom or a list of atoms. (`atm_or_atm_list/1`)

compat/2:

PROPERTY

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.

(True) Usage: `compat(Term,Prop)`

Term is *compatible* with Prop

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

General properties:

True: `compat(Term,Prop)`

- *If the following properties hold at call time:*
Term is currently ground (it contains no variables). (`ground/1`)
Prop is currently ground (it contains no variables). (`ground/1`)
then the following properties hold globally:
`compat(Term,Prop)` is evaluable at compile-time. (`eval/1`)

inst/2:

PROPERTY

(True) Usage: `inst(Term,Prop)`

Term is instantiated enough to satisfy Prop.

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

General properties:

True: `inst(Term,Prop)`

- *The following properties hold globally:*
`inst(Term,Prop)` is side-effect free. (`sideff/2`)

True: `inst(Term,Prop)`

- *If the following properties hold at call time:*
Term is currently ground (it contains no variables). (`ground/1`)
Prop is currently ground (it contains no variables). (`ground/1`)
then the following properties hold globally:
`inst(Term,Prop)` is evaluable at compile-time. (`eval/1`)

- iso/1:** PROPERTY
(True) Usage: `iso(G)`
Complies with the ISO-Prolog standard.
Meta-predicate with arguments: iso(goal).
General properties:
True: `iso(G)`
 – *The following properties hold globally:*
 `iso(G)` is side-effect free. (`sideff/2`)
- deprecated/1:** PROPERTY
 Specifies that the predicate marked with this global property has been deprecated, i.e., its use is not recommended any more since it will be deleted at a future date. Typically this is done because its functionality has been superseded by another predicate.
(True) Usage: `deprecated(G)`
DEPRECATED.
Meta-predicate with arguments: deprecated(goal).
General properties:
True: `deprecated(G)`
 – *The following properties hold globally:*
 `deprecated(G)` is side-effect free. (`sideff/2`)
- not_further_inst/2:** PROPERTY
(True) Usage: `not_further_inst(G,V)`
 V is not further instantiated.
Meta-predicate with arguments: not_further_inst(goal,?).
General properties:
True: `not_further_inst(G,V)`
 – *The following properties hold globally:*
 `not_further_inst(G,V)` is side-effect free. (`sideff/2`)
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
- sideff/2:** PROPERTY
`sideff(G,X)`
 Declares that G is side-effect free (if its execution has no observable result other than its success, its failure, or its abortion), soft (if its execution may have other observable results which, however, do not affect subsequent execution, e.g., input/output), or hard (e.g., assert/retract).
(True) Usage: `sideff(G,X)`
 G is side-effect X.
 – *If the following properties hold at call time:*
 G is a term which represents a goal, i.e., an atom or a structure. (`callable/1`)
 X is an element of [`free,soft,hard`]. (`member/2`)

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

General properties:

True: `sideff(G,X)`

- *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (`native/1`)

`sideff(G,X)` is side-effect free. (`sideff/2`)

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

regtype/1:

PROPERTY

(True) Usage: `regtype G`

Defines a regular type.

Meta-predicate with arguments: `regtype goal`.

General properties:

True: `regtype G`

- *The following properties hold globally:*

`regtype G` is side-effect free. (`sideff/2`)

native/1:

PROPERTY

(True) Usage: `native(Pred)`

This predicate is understood natively by CiaoPP.

Meta-predicate with arguments: `native(goal)`.

General properties:

True: `native(P)`

- *The following properties hold globally:*

`native(P)` is side-effect free. (`sideff/2`)

native/2:

PROPERTY

(True) Usage: `native(Pred,Key)`

This predicate is understood natively by CiaoPP as `Key`.

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

General properties:

True: `native(P,K)`

- *The following properties hold globally:*

`native(P,K)` is side-effect free. (`sideff/2`)

rtcheck/1:

PROPERTY

(True) Usage: `rtcheck(G)`

Equivalent to `rtcheck(G, complete)`.

- *If the following properties hold at call time:*
`G` is a term which represents a goal, i.e., an atom or a structure. (`callable/1`)

Meta-predicate with arguments: `rtcheck(goal)`.

General properties:

True: `rtcheck(G)`

- *The following properties hold globally:*
`rtcheck(G)` is side-effect free. (`sideff/2`)

rtcheck/2: PROPERTY

(True) Usage: `rtcheck(G,Status)`

The runtime check of the property have the status `Status`.

- *If the following properties hold at call time:*
`G` is a term which represents a goal, i.e., an atom or a structure. (`callable/1`)

`Status` of the runtime-check implementation for a given property. Valid values are:

- `unimplemented`: No run-time checker has been implemented for the property. Although it can be implemented further.
- `incomplete`: The current run-time checker is incomplete, which means, under certain circumstances, no error is reported if the property is violated.
- `unknown`: We do not know if current implementation of run-time checker is complete or not.
- `complete`: The opposite of incomplete, error is reported always that the property is violated. Default.
- `impossible`: The property must not be run-time checked (for theoretical or practical reasons).

(`rtc_status/1`)

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

General properties:

True: `rtcheck(G,Status)`

- *The following properties hold globally:*
`rtcheck(G,Status)` is side-effect free. (`sideff/2`)

no_rtcheck/1: PROPERTY

(True) Usage: `no_rtcheck(G)`

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`.

- *If the following properties hold at call time:*
`G` is a term which represents a goal, i.e., an atom or a structure. (`callable/1`)

Meta-predicate with arguments: `no_rtcheck(goal)`.

General properties:

True: `no_rtcheck(G)`

- *The following properties hold globally:*
`no_rtcheck(G)` is side-effect free. (`sideff/2`)

eval/1: (True) Usage: <code>eval(Goal)</code> Goal is evaluable at compile-time. <i>Meta-predicate</i> with arguments: <code>eval(goal)</code> .	PROPERTY
equiv/2: (True) Usage: <code>equiv(Goal1,Goal2)</code> Goal1 is equivalent to Goal2. <i>Meta-predicate</i> with arguments: <code>equiv(goal,goal)</code> .	PROPERTY
bind_ins/1: (True) Usage: <code>bind_ins(Goal)</code> Goal is binding insensitive. <i>Meta-predicate</i> with arguments: <code>bind_ins(goal)</code> .	PROPERTY
error_free/1: (True) Usage: <code>error_free(Goal)</code> Goal is error free. <i>Meta-predicate</i> with arguments: <code>error_free(goal)</code> .	PROPERTY
memo/1: (True) Usage: <code>memo(Goal)</code> Goal should be memoized (not unfolded). <i>Meta-predicate</i> with arguments: <code>memo(goal)</code> .	PROPERTY
filter/2: (True) Usage: <code>filter(Vars,Goal)</code> Vars should be filtered during global control).	PROPERTY
flag_values/1: (True) Usage: <code>flag_values(X)</code> Define the valid flag values	REGTYPE
pe_type/1: (True) Usage: <code>pe_type(Goal)</code> Goal will be filtered in partial evaluation time according to the PE types defined in the assertion. <i>Meta-predicate</i> with arguments: <code>pe_type(goal)</code> .	PROPERTY

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:**

```
:- 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, compat/1, constraint/1, covered/1, covered/2,
exception/1, exception/2, fails/1, finite_solutions/1, have_choicepoints/1,
indep/1, indep/2, instance/1, 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,
sideff_hard/1, sideff_pure/1, sideff_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,
succeeds/1, steps/2, steps_lb/2, steps_o/2, steps_ub/2, tau/1, terminates/1,
test_type/2, throws/2, user_output/2.
```

- **Imports:**

- *System library modules:*

```
terms_check, terms_vars, hiordlib, sort, lists, streams, file_utils, system,
odd, rtchecks/rtchecks_send.
```

- *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, 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.
```

- *Packages:*

```
prelude, nonpure, assertions, hiord.
```

8.2 Documentation on exports (`native_props`)

`clique/1:`

```
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)`

The clique pattern is X.

PROPERTY

- *The following properties should hold globally:*

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

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

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)`

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`)

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

compat/1:

PROPERTY

Usage: `compat(Prop)`

Use `Prop` as a compatibility property.

- *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

Meta-predicate with arguments: `compat(goal)`.

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.

(True) Usage: `constraint(C)`

`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)`

All the calls of the form `X` are covered.

- *The following properties should hold globally:*

The runtime check of the property have the status `unimplemented`. (`rtcheck/2`)

<p>covered/2: <code>covered(X,Y)</code> All variables occurring in <code>X</code> occur also in <code>Y</code>. (True) Usage: <code>covered(X,Y)</code> <code>X</code> is covered by <code>Y</code>. – <i>The following properties hold globally:</i> This predicate is understood natively by CiaoPP.</p>	<p>PROPERTY (native/1)</p>
<p>exception/1: Usage: <code>exception(Goal)</code> Calls of the form <code>Goal</code> throw an exception. <i>Meta-predicate</i> with arguments: <code>exception(goal)</code>.</p>	<p>PROPERTY</p>
<p>exception/2: Usage: <code>exception(Goal,E)</code> Calls of the form <code>Goal</code> throw an exception that unifies with <code>E</code>. <i>Meta-predicate</i> with arguments: <code>exception(goal,?)</code>.</p>	<p>PROPERTY</p>
<p>fails/1: <code>fails(X)</code> Calls of the form <code>X</code> fail. (True) Usage: <code>fails(X)</code> Calls of the form <code>X</code> fail. – <i>The following properties hold globally:</i> This predicate is understood natively by CiaoPP. <i>Meta-predicate</i> with arguments: <code>fails(goal)</code>.</p>	<p>PROPERTY (native/1)</p>
<p>finite_solutions/1: <code>finite_solutions(X)</code> Calls of the form <code>X</code> produce a finite number of solutions [DLGH97]. Usage: <code>finite_solutions(X)</code> All the calls of the form <code>X</code> have a finite number of solutions. – <i>The following properties should hold globally:</i> Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to <code>rtcheck(G, impossible)</code>. <i>Meta-predicate</i> with arguments: <code>finite_solutions(goal)</code>.</p>	<p>PROPERTY (no_rtcheck/1)</p>

- have_choicepoints/1:** PROPERTY
Usage: `have_choicepoints(X)`
 A call to `X` creates choicepoints.
Meta-predicate with arguments: `have_choicepoints(goal)`.
- indep/1:** PROPERTY
(True) Usage: `indep(X)`
 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
(True) Usage: `indep(X,Y)`
`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`)
- instance/1:** PROPERTY
Usage: `instance(Prop)`
 Use `Prop` as an instantiation property. Verify that execution of `Prop` does not produce bindings for the argument variables.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
Meta-predicate with arguments: `instance(goal)`.
- is_det/1:** PROPERTY
`is_det(X)`
 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.
Usage: `is_det(X)`
 All calls of the form `X` are deterministic.
Meta-predicate with arguments: `is_det(goal)`.
- linear/1:** PROPERTY
`linear(X)`
`X` is bound to a term which is linear, i.e., if it contains any variables, such variables appear only once in the term. For example, `[1,2,3]` and `f(A,B)` are linear terms, while `f(A,A)` is not.
(True) Usage: `linear(X)`
`X` is instantiated to a linear term.

- *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (`native/1`)

mshare/1: PROPERTY

`mshare(X)`

`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)`

The sharing pattern is `X`.

- *The following properties should hold globally:*

This predicate is understood natively by CiaoPP as `sharing(X)`. (`native/2`)

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

General properties:

Test: `mshare(L)`

- *If the following properties should hold at call time:*

`L=[[A],[p(A)]]` (`= /2`)

then the following properties should hold globally:

Calls of the form `mshare(L)` fail. (`fails/1`)

Test: `mshare(L)`

- *If the following properties should hold at call time:*

`L=[[A],[p(B)]]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `mshare(L)` do not fail. (`not_fails/1`)

mut_exclusive/1: PROPERTY

`mut_exclusive(X)`

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

Usage: `mut_exclusive(X)`

For any call of the form `X` at most one clause succeeds.

- *The following properties should hold globally:*

The runtime check of the property have the status `unimplemented`. (`rtcheck/2`)

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

no_choicepoints/1: PROPERTY

Usage: `no_choicepoints(X)`

A call to `X` does not create choicepoints.

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

<p>no_exception/1: Usage: no_exception(Goal) Calls of the form <code>Goal</code> do not throw any exception. <i>Meta-predicate</i> with arguments: no_exception(goal).</p>	PROPERTY
<p>no_exception/2: Usage: no_exception(Goal,E) Calls of the form <code>Goal</code> do not throw exception <code>E</code>. <i>Meta-predicate</i> with arguments: no_exception(goal,?).</p>	PROPERTY
<p>no_signal/1: Usage: no_signal(Goal) Calls of the form <code>Goal</code> do not send any signal. <i>Meta-predicate</i> with arguments: no_signal(goal).</p>	PROPERTY
<p>no_signal/2: Usage: no_signal(Goal,E) Calls of the form <code>Goal</code> do not send the signal <code>E</code>. <i>Meta-predicate</i> with arguments: no_signal(goal,?).</p>	PROPERTY
<p>non_det/1: non_det(X) All calls of the form <code>X</code> are non-deterministic, i.e., produce several solutions. Usage: non_det(X) All calls of the form <code>X</code> are non-deterministic. <i>Meta-predicate</i> with arguments: non_det(goal).</p>	PROPERTY
<p>nonground/1: Usage: nonground(X) <code>X</code> is not ground. – <i>The following properties should hold globally:</i> This predicate is understood natively by CiaoPP as not_ground(X). (native/2)</p>	PROPERTY
<p>not_covered/1: not_covered(X) There is some call of the form <code>X</code> for which there is no clause whose test succeeds [DLGH97]. Usage: not_covered(X) Not all of the calls of the form <code>X</code> are covered. – <i>The following properties should hold globally:</i> The runtime check of the property have the status unimplemented. (rtcheck/2)</p>	PROPERTY

- not_fails/1:** PROPERTY
`not_fails(X)`
 Calls of the form `X` produce at least one solution, or do not terminate [DLGH97].
(True) Usage: `not_fails(X)`
 All the calls of the form `X` do not fail.
 – *The following properties hold globally:*
 This predicate is understood natively by CiaoPP. (native/1)
Meta-predicate with arguments: `not_fails(goal)`.
- 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.
Usage: `not_mut_exclusive(X)`
 For some calls of the form `X` more than one clause may succeed.
 – *The following properties should hold globally:*
 The runtime check of the property have the status `unimplemented`. (rtcheck/2)
Meta-predicate with arguments: `not_mut_exclusive(goal)`.
- num_solutions/2:** PROPERTY
Usage 1: `num_solutions(X,N)`
 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)`
 For a call to `Goal`, `Check(X)` succeeds, where `X` is the number of solutions.
 – *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)`
 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 can be ensured about non-failure nor termination of such calls.

Usage: `possibly_fails(X)`

Non-failure is not ensured for calls of the form `X`.

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

Meta-predicate with arguments: `possibly_fails(goal)`.

possibly_nondet/1: PROPERTY`possibly_nondet(X)`

Non-determinism is not ensured for all calls of the form `X`. In other words, nothing can be ensured about determinacy nor termination of such calls.

Usage: `possibly_nondet(X)`

Non-determinism is not ensured for calls of the form `X`.

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

relations/2: PROPERTY`relations(X,N)`

The goal `X` produces `N` solutions. In other words, `N` is the cardinality of the solution set of `X`.

Usage: `relations(X,N)`

Goal `X` produces `N` solutions.

– *The following properties should hold globally:*

The runtime check of the property have the status `unimplemented`. (`rtcheck/2`)

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

sideff_hard/1: PROPERTY**Usage:** `sideff_hard(X)`

`X` has *hard side-effects*, i.e., those that might affect program execution (e.g., `assert/retract`).

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

Meta-predicate with arguments: `sideff_hard(goal)`.

- sideff_pure/1:** PROPERTY
Usage: `sideff_pure(X)`
 X is pure, i.e., has no side-effects.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
Meta-predicate with arguments: `sideff_pure(goal)`.
- sideff_soft/1:** PROPERTY
Usage: `sideff_soft(X)`
 X has *soft side-effects*, i.e., those not affecting program execution (e.g., input/output).
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
Meta-predicate with arguments: `sideff_soft(goal)`.
- signal/1:** PROPERTY
Usage: `signal(Goal)`
 Calls of the form `Goal` throw a signal.
Meta-predicate with arguments: `signal(goal)`.
- signal/2:** PROPERTY
Usage: `signal(Goal,E)`
 A call to `Goal` sends a signal that unifies with `E`.
Meta-predicate with arguments: `signal(goal,?)`.
- signals/2:** PROPERTY
Usage: `signals(Goal,Es)`
 Calls of the form `Goal` can generate only the signals that unify with the terms listed in `Es`.
 – *The following properties should hold globally:*
 The runtime check of the property have the status `unimplemented`. (`rtcheck/2`)
Meta-predicate with arguments: `signals(goal,?)`.
- size/2:** PROPERTY
Usage: `size(X,Y)`
 Y is the size of argument X, for any approximation.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

- size/3:** PROPERTY
Usage: `size(A,X,Y)`
 Y is the size of argument X, for the approximation A.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
- size_lb/2:** PROPERTY
size_lb(X,Y)
 The minimum size of the terms to which the argument Y is bound is given by the expression Y. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93,LGHD96].
Usage: `size_lb(X,Y)`
 Y is a lower bound on the size of argument X.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
- size_o/2:** PROPERTY
Usage: `size_o(X,Y)`
 The size of argument X is in the order of Y.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
- size_ub/2:** PROPERTY
size_ub(X,Y)
 The maximum size of the terms to which the argument Y is bound is given by the expression Y. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93,LGHD96].
Usage: `size_ub(X,Y)`
 Y is an upper bound on the size of argument X.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
- size_metric/3:** PROPERTY
Usage: `size_metric(Head,Var,Metric)`
 Metric is the metric of the variable Var, for any approximation.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
 Meta-predicate with arguments: `size_metric(goal,?,?)`.

size_metric/4: PROPERTY

Usage: `size_metric(Head, Approx, Var, Metric)`

`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`.

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

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

succeeds/1: PROPERTY

Usage: `succeeds(Prop)`

A call to `Prop` succeeds.

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

Meta-predicate with arguments: `succeeds(goal)`.

steps/2: PROPERTY

`steps(X,Y)`

The time (in resolution steps) spent by any call of the form `X` is given by the expression `Y`

Usage: `steps(X,Y)`

`Y` is the cost (number of resolution steps) of any call of the form `X`.

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

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

steps_lb/2: PROPERTY

`steps_lb(X,Y)`

The minimum computation time (in resolution steps) spent by any call of the form `X` is given by the expression `Y` [DLGHL97,LGHD96]

Usage: `steps_lb(X,Y)`

`Y` is a lower bound on the cost of any call of the form `X`.

– *The following properties should hold globally:*

Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)

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

- steps_o/2:** PROPERTY
Usage: `steps_o(X,Y)`
 Y is the complexity order of the cost of any call of the form X.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
Meta-predicate with arguments: `steps_o(goal,?)`.
- steps_ub/2:** PROPERTY
steps_ub(X,Y)
 The maximum computation time (in resolution steps) spent by any call of the form X is given by the expression Y [DL93,LGHD96].
Usage: `steps_ub(X,Y)`
 Y is a upper bound on the cost of any call of the form X.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
Meta-predicate with arguments: `steps_ub(goal,?)`.
- 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
(True) Usage: `tau(TypeInfo)`
 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:** PROPERTY
terminates(X)
 Calls of the form X always terminate [DLGH97].
Usage: `terminates(X)`
 All calls of the form X terminate.
 – *The following properties should hold globally:*
 Declares that the assertion in which this comp property appears must not be checked at run-time. Equivalent to `rtcheck(G, impossible)`. (`no_rtcheck/1`)
Meta-predicate with arguments: `terminates(goal)`.
- test_type/2:** PROPERTY
Usage: `test_type(X,T)`
 Indicates the type of test that a predicate performs. Required by the nonfailure analysis.
Meta-predicate with arguments: `test_type(goal,?)`.

- throws/2:** PROPERTY
Usage: `throws(Goal,Es)`
Calls of the form `Goal` can throw only the exceptions that unify with the terms listed in `Es`.
– *The following properties should hold globally:*
 The runtime check of the property have the status `unimplemented`. (`rtcheck/2`)
Meta-predicate with arguments: `throws(goal,?)`.
- user_output/2:** PROPERTY
Usage: `user_output(Goal,S)`
Calls of the form `Goal` write `S` to standard output.
Meta-predicate with arguments: `user_output(goal,?)`.
- instance/2:** PROPERTY
(True) Usage: `instance(Term1,Term2)`
`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:

```
:- pred p(X) : Prop(X) => Prop(X).
```

Another example is using shorthands for properties when documenting:

```
:- pred p(X) : regtype(X, (^ (list; list); list)).
```

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

9.1 Usage and interface (meta_props)

- **Library usage:**

```
:- use_module(library(assertions(meta_props)))
```

or also as a package `:- use_package(metaprops).`

Note the different names of the library and the package.

- **Exports:**

- *Properties:*

```
call/2, prop/2, regtype/2.
```

- *Multifiles:*

```
callme/2.
```

- **Imports:**

- *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
```

- *Packages:*

```
prelude, nonpure, assertions, hiord.
```

9.2 Documentation on exports (meta_props)

call/2:

```
call(P,A)
```

A has property P (provided that P is a property). Equivalent to P(A).

(True) Usage: call(P,A)

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.

PROPERTY

(callable/1)

prop/2: PROPERTY
 (True) Usage: A prop P
 A has property P.
 – If the following properties hold at call time:
 P has property $\hat{\ }(\text{callable};\text{prop_abs})$. ((prop)/2)

regtype/2: PROPERTY
 (True) Usage: A regtype T
 A is of type T.
 – If the following properties hold at call time:
 T has property $\hat{\ }((\text{regtype});\text{prop_abs})$. ((prop)/2)

9.3 Documentation on multifiles (meta_props)

callme/2: PREDICATE
 (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).
 (Trust) 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. (callable/1)
 The predicate is *multifile*.

9.4 Documentation on internals (meta_props)

prop_abs/1: PROPERTY
`prop_abs(Prop)`
 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 $\hat{\ }$.
 One particular case of property abstractions are *parametric regular type abstractions*, i.e., a parametric type functor or a $\hat{\ }$ -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:

```
 $\hat{\ }(\text{list};\text{list});\text{list}$ 
```

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)`
 Prop is a property abstraction.

10 An Example - Documenting a Library Module

Author(s): Manuel Hermenegildo.

A simple example of the use of `lpdoc` is this manual, which can be built in the `doc` directory of the `lpdoc` distribution. Other examples of manuals generated using `lpdoc` can be found in the `Ciao` system and preprocessor `doc` directories (i.e., most of the `Ciao` manuals are generated using `lpdoc`). Some simpler examples can be found in the `examples` directory of the `lpdoc` 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 `lpdoc`:

```
%% The module headers produce documentation on the module interface
%% 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).

%% We reexport list/1
:- reexport(library(engine(basic_props)), [ 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/1}

```

```

    ").

%% The comment here will be used to document any predicate which has an
%% assertion which uses the property
:- prop long(L) # "@var{L} 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).
:- entry p/3 : gnd * var * var.

%% This describes all the calls
:- calls p/3 : foo * bar * baz.

foo(_).

%% This describes the successes (for a given type of calls)
:- success p/3 : int * int * var => int * int * gnd.

%% This describes a global property (for a given type of calls)
:- comp p/3 : int * int * var + not_fails.

:- doc(p/3,"A @bf{general comment} on the predicate." ).
%% Documenting some typical usages of the predicate
:- pred p/3
    : int * int * var
    => int * int * list
    + (iso,not_fails)
    # "This mode is nice.".
:- pred p(Preds,Value,Assoc)
    : var * var * list
    => int * int * list
    + not_fails # "This mode is also nice.".
:- pred p/3
    => list * int * list
    + (not_fails,not_fails)
    # "Just playing around.".

:- pred q(A)
    : list(A)
    => (list(A),gnd(A))
    + not_fails
    # "Foo".
:- pred q(A)
    # "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

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

:- doc(doinclude,og/2).

:- modedef 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:**
 - :- 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.
- **Imports:**
 - *Files of module user:*
foo.
 - *System library modules:*
assertions/native_props, engine/basic_props, lists.
 - *Internal (engine) modules:*
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.
 - *Packages:*
prelude, nonpure, assertions, basicmodes, fsyntax, regtypes, hiord, nativeprops.

11.2 Documentation on exports (example_module)

bar/1: REGTYPE
(True) Usage: bar(X)
X is an acceptable kind of bar.

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

aorb/1:	REGTYPE
A regular type, defined as follows:	
<pre> aorb(a). aorb(b). </pre>	
tree_of/2:	REGTYPE
A regular type, defined as follows:	
<pre> tree_of(_1,void). tree_of(T,tree(_1,_2,_3)) :- call(T,_1), tree_of(T,_2), tree_of(T,_3). </pre>	
list_or_aorb/2:	REGTYPE
A regular type, defined as follows:	
<pre> list_or_aorb(T,_1) :- list(T,_1). list_or_aorb(_T,_1) :- aorb(_1). </pre>	
q/2:	PREDICATE
Usage 1:	
– <i>The following properties should hold at call time:</i>	
Arg1 is a free variable.	(var/1)
Arg2 is ground.	(gnd/1)
Arg2 is an integer.	(int/1)
– <i>The following properties should hold upon exit:</i>	
Arg1 is ground.	(gnd/1)
Arg1 is an integer.	(int/1)
Arg2 is an integer.	(int/1)
Usage 2:	
Non-moded types are best used this way.	
– <i>Call and exit should be compatible with:</i>	
Arg1 is an integer.	(int/1)
Arg2 is a list.	(list/1)
The predicate is of type <i>dynamic</i> .	
r/1:	PREDICATE
Usage: r(A)	
This uses parametric types	
– <i>The following properties should hold at call time:</i>	
A is a list.	(list/1)

- *The following properties should hold upon exit:*
 - A is a list of ints. (list/2)
 - A is ground. (gnd/1)
- *The following properties should hold globally:*
 - All the calls of the form $r(A)$ do not fail. (not_fails/1)

The predicate is of type *data*.

p/1: PREDICATE

Usage:

- *The following properties should hold at call time:*
 - Arg1 is a free variable. (var/1)
- *The following properties should hold upon exit:*
 - Arg1 is a list. (list/1)

p/5: PREDICATE

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

- *Call and exit should be compatible with:*
 - Arg1 is an integer. (int/1)
 - Arg3 is a list of ints. (list/2)
- *The following properties should hold at call time:*
 - Arg2 is currently ground (it contains no variables). (ground/1)
 - Arg4 is a free variable. (var/1)
 - A is currently a term which is not a free variable. (nonvar/1)
- *The following properties should hold upon exit:*
 - Arg1 is ground. (gnd/1)
 - Arg2 is currently ground (it contains no variables). (ground/1)
- *The following properties should hold globally:*
 - Arg3 is not further instantiated. (not_further_inst/2)
 - $1 + \text{length}(A)$ is a lower bound on the cost of any call of the form $p(\text{Arg1}, \text{Arg2}, \text{Arg3}, \text{Arg4}, A)$. (steps_lb/2)

u/3: PREDICATE

Usage 1:

- *The following properties should hold at call time:*
 - Arg1 is currently a term which is not a free variable. (nonvar/1)
 - Arg2 is a free variable. (var/1)
- *The following properties should hold upon exit:*
 - Arg3 is ground. (gnd/1)

long/1:		PROPERTY
This is a property, describing a list that is longish. The definition is:		
	<pre>long(L) :- length(L,N), N>100.</pre>	
Usage: long(L)		
L is rather long.		
w/1:		PREDICATE
(True) Usage:		
– <i>Calls should, and exit will be compatible with:</i>		
Arg1 is a list of mytypes.		(list/2)
– <i>The following properties should hold at call time:</i>		
Arg1 is currently a term which is not a free variable.		(nonvar/1)
mytype/1:		PREDICATE
No further documentation available for this predicate.		
t/5:		PREDICATE
Usage: t(A,B,C,D,E)		
This predicate uses <i>modes</i> extensively.		
– <i>Call and exit should be compatible with:</i>		
A is a list.		(list/1)
B is a list.		(list/1)
C is an integer.		(int/1)
D is an integer.		(int/1)
E is a list.		(list/1)
– <i>The following properties should hold at call time:</i>		
A is currently a term which is not a free variable.		(nonvar/1)
B is a free variable.		(var/1)
B is rather long.		(long/1)
– <i>The following properties should hold upon exit:</i>		
E is ground.		(gnd/1)
C is ground.		(gnd/1)
A is ground.		(gnd/1)
– <i>The following properties should hold globally:</i>		
D is not further instantiated.		(not_further_inst/2)
All the calls of the form t(A,B,C,D,E) do not fail.		(not_fails/1)

s/1:		PREDICATE
Usage: s(A)		
– <i>The following properties should hold at call time:</i>		
A is a list.		(list/1)
– <i>The following properties should hold upon exit:</i>		
A is a list.		(list/1)
A is ground.		(gnd/1)
– <i>The following properties should hold globally:</i>		
All the calls of the form s(A) do not fail.		(not_fails/1)
q/1:		PREDICATE
Usage 1: q(A)		
Foo		
– <i>The following properties should hold at call time:</i>		
A is a list.		(list/1)
– <i>The following properties should hold upon exit:</i>		
A is a list.		(list/1)
A is ground.		(gnd/1)
– <i>The following properties should hold globally:</i>		
All the calls of the form q(A) do not fail.		(not_fails/1)
Usage 2: q(A)		
Not a bad use at all.		
list/1:		REGTYPE
(True) Usage: list(L)		
L is a list.		
General properties:		
True: list(L)		
– <i>The following properties hold globally:</i>		
list(L) is side-effect free.		(sideff/2)
True: list(L)		
– <i>If the following properties hold at call time:</i>		
L is currently ground (it contains no variables).		(ground/1)
<i>then the following properties hold globally:</i>		
list(L) is evaluable at compile-time.		(eval/1)
All calls of the form list(L) are deterministic.		(is_det/1)
Trust: list(T)		
– <i>The following properties hold upon exit:</i>		
T is a list.		(list/1)

11.3 Documentation on multifiles (example_module)

p/3:		PREDICATE
A general comment on the predicate.		
Usage 1:		• ISO •
This mode is nice.		
– <i>The following properties should hold at call time:</i>		
Arg1 is an integer.		(int/1)
Arg2 is an integer.		(int/1)
Arg3 is a free variable.		(var/1)
– <i>The following properties should hold upon exit:</i>		
Arg1 is an integer.		(int/1)
Arg2 is an integer.		(int/1)
Arg3 is a list.		(list/1)
– <i>The following properties should hold globally:</i>		
<i>Complies with the ISO-Prolog standard.</i>		(iso/1)
All the calls of the form p(Arg1,Arg2,Arg3) do not fail.		(not_fails/1)
Usage 2: p(Preds,Value,Assoc)		
This mode is also nice.		
– <i>The following properties should hold at call time:</i>		
Preds is a free variable.		(var/1)
Value is a free variable.		(var/1)
Assoc is a list.		(list/1)
– <i>The following properties should hold upon exit:</i>		
Preds is an integer.		(int/1)
Value is an integer.		(int/1)
Assoc is a list.		(list/1)
– <i>The following properties should hold globally:</i>		
All the calls of the form p(Preds,Value,Assoc) do not fail.		(not_fails/1)
Usage 3:		
Just playing around.		
– <i>The following properties should hold upon exit:</i>		
Arg1 is a list.		(list/1)
Arg2 is an integer.		(int/1)
Arg3 is a list.		(list/1)
– <i>The following properties should hold globally:</i>		
All the calls of the form p(Arg1,Arg2,Arg3) do not fail.		(not_fails/1)
All the calls of the form p(Arg1,Arg2,Arg3) do not fail.		(not_fails/1)
The predicate is <i>multifile</i> .		
The predicate is of type <i>dynamic</i> .		
General properties:		
True:		

- *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)

Check:

- *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)

Check:

- *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)

Check:

- *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(Arg1,Arg2,Arg3) do not fail. (not_fails/1)

11.4 Documentation on internals (example_module)

list/2: REGTYPE

(True) Usage: list(L,T)

L is a list of Ts.

General properties:

True: list(L,T)

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

True: list(L,T)

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

Trust: list(X,T)

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

og/2:

MODE

(True) Usage: og(A,T)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.

(gnd/1)

is/2:

PREDICATE

(Trust) Usage: Num is Expr

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)

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.
- `rtchecks_abort_on_error`

Controls if run time checks must abort the execution of a program (by raising an exception), or if the execution of the program have to continue.

Note that this option only affect the default handler and the predicate `call_rtc/1`, so if you use your own handler it will not have effect.

 - `yes` : Raising a run time error will abort the program.
 - `no` : Raising a run time error will not stop the execution, but a message will be shown. Default.

12.1 Usage and interface (`rtchecks_doc`)

- **Library usage:**

```
:- use_package(rtchecks).
```

or

```
:- module(...,[rtchecks]).
```
- **Imports:**
 - *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
```
 - *Packages:*

```
prelude, nonpure, assertions.
```

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:**

- :- use_module(library(unittest)).

- **Imports:**

- *Internal (engine) modules:*

- term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.

- *Packages:*

- prelude, nonpure, assertions, regtypes.

14 Installing lpdoc

Author(s): Manuel Hermenegildo.

The source and binaries distributions of lpdoc are currently included in Ciao (<http://ciao-lang.org>). Please, refer to the Ciao installation instructions for more details regarding the building and installation process.

14.1 Other software packages required (lpdoc)

The most basic functionality of lpdoc (generating manuals in `.texi` format, short manual entries in `.man1` 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.

- **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.

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 .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.
- **Generating .html files:** `.html` files are generated directly from the HTML lpdoc backend.
- If pictures are used in the manual, the command `convert` (ImageMagick (<http://www.imagemagick.org>)) is required in order to convert figures from several common graphical formats.

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.

Note: This part is obsolete. – JFMC

The output format in which the documentation is generated is defined by the backend modules (`autodoc_texinfo`, `autodoc_html`, `autodoc_man`, etc.).

The main output format supported is `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 `man` format and direct `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 `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:**
 - :- use_module(library(autodoc)).
- **Exports:**
 - *Predicates:*
 - 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:*
 - autodoc_finish_hook/1, autodoc_gen_alternative_hook/2.
- **Imports:**
 - *Application modules:*
 - lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_settings)),
 lpdocsrc(src(autodoc_filesystem)), lpdocsrc(src(autodoc_structure)),
 lpdocsrc(src(autodoc_doctree)), lpdocsrc(src(autodoc_refsdb)),
 lpdocsrc(src(autodoc_parse)), lpdocsrc(src(autodoc_index)),
 lpdocsrc(src(comments)), lpdocsrc(src(autodoc_html_resources)),
 lpdocsrc(src(autodoc_texinfo)), lpdocsrc(src(autodoc_aux)).
 - *System library modules:*
 - format, ttyout, aggregates, read, make/make_rt, dict, compiler/compiler,
 assertions/assrt_lib, compiler/c_itf, messages, filenames, lists, terms,
 system_extra, assertions/assertions_props, system.
 - *Internal (engine) modules:*
 - term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
 exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare,
 term_typing, hiord_rt, debugger_support.
 - *Packages:*
 - prelude, nonpure, assertions, regtypes, dcg, basicmodes, fsyntax.

15.2 Documentation on exports (autodoc)

- index_comment/2:** PREDICATE
Usage: index_comment(Index,Text)
 Type is a type of index which is supported. Text describes the index contents.
- *The following properties should hold upon exit:*
 - 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:** PREDICATE
Usage: `ensure_output_dir_prepared(Backend,Opts)`
 Ensure that the output directories for backend `Backend` are prepared.
- get_autodoc_opts/3:** PREDICATE
Usage: `get_autodoc_opts(Backend,Mod,Opts)`
 Get the list of documentation options `Opts` for the `FileBase` file.
 – *The following properties should hold at call time:*
 `Backend` is an atom. (`atm/1`)
 `Mod` is an atom. (`atm/1`)
 `Opts` is a list of `supported_options`. (`list/2`)
- autodoc_gen_doctree/5:** PREDICATE
Usage: `autodoc_gen_doctree(Backend,FileBase,SourceSuffix,Opts,Mod)`
`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/1`.
 – *Call and exit should be compatible with:*
 `Backend` is a supported backend. (`backend_id/1`)
 `FileBase` is the base name of a file (without extension). (`basename/1`)
 `SourceSuffix` is an atom. (`atm/1`)
 `Opts` is a list of `supported_options`. (`list/2`)
 `Mod` is an atom. (`atm/1`)
- fmt_infodir_entry/3:** PREDICATE
Usage: `fmt_infodir_entry(DocSt,Version,Mod)`
 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/1`)
 `Version` is any term. (`term/1`)
 `Mod` is the base name of a file (without extension). (`basename/1`)
- autodoc_compute_grefs/3:** PREDICATE
Usage:
 Compute the globally resolved references (including bibliography)
- autodoc_translate_doctree/3:** PREDICATE
Usage:
 Translate the doctree using the specific backend

autodoc_finish/1:

No further documentation available for this predicate.

PREDICATE

autodoc_gen_alternative/2:

No further documentation available for this predicate.

PREDICATE

15.3 Documentation on multifiles (autodoc)

autodoc_finish_hook/1:

No further documentation available for this predicate. The predicate is *multifile*.

PREDICATE

autodoc_gen_alternative_hook/2:

No further documentation available for this predicate. The predicate is *multifile*.

PREDICATE

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:**

```
:- use_module(library(autodoc_state)).
```

- **Exports:**

- *Predicates:*

```
option_comment/2, backend_ignores_components/1, backend_alt_format/2, top_
suffix/2, docst_backend/2, docst_currmod/2, docst_set_currmod/3, docst_
opts/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_currmod_is_main/1, docst_no_components/1, docst_modname/2, labgen_
init/1,                                labgen_clean/1,                                labgen_
get/2, docst_mvar_lookup/3, docst_mvar_replace/4, docst_mvar_get/3, docst_
mdata_clean/1, docst_mdata_assertz/2, docst_mdata_save/1, docst_gdata/3,
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_changes/3, get_doc_pred_varnames/2, doc_
assertion_read/9, bind_dict_varnames/1, get_mod_doc/3, pred_has_docprop/2,
docst_modtype/2, get_first_loc_for_pred/3.
```

- *Properties:*

```
supported_option/1.
```

- *Regular Types:*

```
backend_id/1, docstate/1, modtype/1.
```

- **Imports:**

- *Application modules:*

```
lpdocsrc(src(autodoc_settings)),    lpdocsrc(src(autodoc_filesystem)),
lpdocsrc(src(autodoc_structure)),    lpdocsrc(src(autodoc_
doctree)), lpdocsrc(src(autodoc_refsdbs)), lpdocsrc(src(autodoc_parse)),
lpdocsrc(src(autodoc_index)),        lpdocsrc(src(comments)),
lpdocsrc(src(autodoc_aux)).
```

- *System library modules:*

```
make/make_rt, dict, aggregates, compiler/compiler, assertions/assrt_lib,
compiler/c_itf, assertions/assertions_props, messages, filenames, lists,
terms, system_extra, write, read, system.
```

- *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
```

- *Packages:*

```
prelude, nonpure, dcg, assertions, regtypes, basicmodes, fsyntax.
```


16.2 Documentation on exports (autodoc_state)

supported_option/1: PROPERTY

Usage: supported_option(Option)

Option is a supported documentation option.

option_comment/2: PREDICATE

Usage: option_comment(Option,Text)

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

```
option_comment(verbose,"Verbose output (good for debugging).").
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_lpdock,"Do not include an ack of LPdoc in output.").
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 proper).
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 the import list.").
option_comment(no_engmods,"Do not include system engine modules in the import list.").
option_comment(no_packages,"Do not include packages in the import list.").
option_comment(no_isoline,"Do not include *textual* description that a given usage
option_comment(propmods,"Include module name to which props belong.").
option_comment(no_propuses,"Do not include property uses (from assertions) in individual
option_comment(shorttoc,"Produce shorter table of contents (no entries for individual
option_comment(regtype_props,"Include in the doc for regtypes the global prop state
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. (supported_option/1)

Text is a string (a list of character codes). (string/1)

backend_id/1: REGTYPE

Usage: backend_id(Id)

Id is a supported backend.

backend_ignores_components/1:	PREDICATE
Usage: <code>backend_ignores_components(Id)</code> Id does not take into account components (only documents the <i>mainfile</i>)	
backend_alt_format/2:	PREDICATE
Usage: <code>backend_alt_format(Id,Ext)</code> Ext is an alternative file format that can be generated by the backend Id	
top_suffix/2:	PREDICATE
Usage: <code>top_suffix(FileFormat,PrincipalExt)</code> PrincipalExt is extension of the target file that will generate the file with FileFormat extension.	
docstate/1:	REGTYPE
A regular type, defined as follows: <pre>docstate(docstate(Backend,Name,Opts,MVarDic,I)) :- backend_id(Backend), atom(Name), list(Opts,supported_option), dictionary(MVarDic), filename(I).</pre>	
docst_backend/2:	PREDICATE
No further documentation available for this predicate.	
docst_currmod/2:	PREDICATE
No further documentation available for this predicate.	
docst_set_currmod/3:	PREDICATE
No further documentation available for this predicate.	
docst_opts/2:	PREDICATE
No further documentation available for this predicate.	
docst_set_opts/3:	PREDICATE
No further documentation available for this predicate.	

docst_inputfile/2: No further documentation available for this predicate.	PREDICATE
docst_new_no_src/4: No further documentation available for this predicate.	PREDICATE
docst_new_with_src/6: No further documentation available for this predicate.	PREDICATE
docst_new_sub/3: No further documentation available for this predicate.	PREDICATE
docst_message/2: No further documentation available for this predicate.	PREDICATE
docst_message/3: No further documentation available for this predicate.	PREDICATE
docst_opt/2: No further documentation available for this predicate.	PREDICATE
docst_currmod_is_main/1: No further documentation available for this predicate.	PREDICATE
docst_no_components/1: Usage: <code>docst_no_components(DocSt)</code> DocSt specify an empty list of components	PREDICATE
docst_modname/2: Usage: <code>docst_modname(DocSt,ModName)</code> ModName is the name of the module that we are documenting.	PREDICATE
labgen_init/1: No further documentation available for this predicate.	PREDICATE

labgen_clean/1: No further documentation available for this predicate.	PREDICATE
labgen_get/2: No further documentation available for this predicate.	PREDICATE
docst_mvar_lookup/3: No further documentation available for this predicate.	PREDICATE
docst_mvar_replace/4: No further documentation available for this predicate.	PREDICATE
docst_mvar_get/3: No further documentation available for this predicate.	PREDICATE
docst_mdata_clean/1: No further documentation available for this predicate.	PREDICATE
docst_mdata_assertz/2: No further documentation available for this predicate.	PREDICATE
docst_mdata_save/1: No further documentation available for this predicate.	PREDICATE
docst_gdata/3: No further documentation available for this predicate. The predicate is of type <i>data</i> .	PREDICATE
docst_gdata_query/2: No further documentation available for this predicate.	PREDICATE
docst_gdata_query/3: No further documentation available for this predicate.	PREDICATE
docst_gdata_restore/1: No further documentation available for this predicate.	PREDICATE

docst_gdata_clean/1: No further documentation available for this predicate.	PREDICATE
docst_gvar_save/2: No further documentation available for this predicate.	PREDICATE
docst_gvar_restore/2: No further documentation available for this predicate.	PREDICATE
docst_has_index/2: No further documentation available for this predicate.	PREDICATE
all_indices/2: No further documentation available for this predicate.	PREDICATE
get_doc/4: No further documentation available for this predicate.	PREDICATE
get_doc_changes/3: No further documentation available for this predicate.	PREDICATE
get_doc_pred_varnames/2: No further documentation available for this predicate.	PREDICATE
doc_assertion_read/9: No further documentation available for this predicate.	PREDICATE
bind_dict_varnames/1: Usage: <code>bind_dict_varnames(Dict)</code> Binds the variables in <code>Dict</code> to the corresponding names (i.e., the names that appeared in the source during read.	PREDICATE
get_mod_doc/3: No further documentation available for this predicate.	PREDICATE

pred_has_docprop/2:

No further documentation available for this predicate.

PREDICATE

modtype/1:

```
modtype(part).  
modtype(application).  
modtype(documentation).  
modtype(module).  
modtype(user).  
modtype(include).  
modtype(package).
```

Usage:

Represents the type of file being documented.

REGTYPE

docst_modtype/2:

No further documentation available for this predicate.

PREDICATE

get_first_loc_for_pred/3:

No further documentation available for this predicate.

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:**

```
:- 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, doctree_translate_and_write/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.
```

- **Imports:**

- *Application modules:*

```
lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_
refsdb)), lpdocsrc(src(autodoc_index)), lpdocsrc(src(autodoc_structure)),
lpdocsrc(src(autodoc_filesystem)), lpdocsrc(src(autodoc_
settings)), lpdocsrc(src(comments)), lpdocsrc(src(autodoc_texinfo)),
lpdocsrc(src(autodoc_man)), lpdocsrc(src(autodoc_html)).
```

- *System library modules:*

```
write, operators, format, lists, system_extra, read, terms, make/make_rt.
```

- *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
```

- *Packages:*

```
prelude, nonpure, dcg, assertions, regtypes, basicmodes, fsyntax.
```

17.2 Documentation on exports (`autodoc_doctree`)

cmd_type/1:	PREDICATE
No further documentation available for this predicate.	
doctree/1:	REGTYPE
Usage:	
Intermediate tree representation for documentation	
doctree_is_empty/1:	PREDICATE
Usage: doctree_is_empty(R)	
Emptiness test	
– <i>The following properties should hold at call time:</i>	
R is currently a term which is not a free variable.	(nonvar/1)
Intermediate tree representation for documentation	(doctree/1)
is_nonempty_doctree/1:	PREDICATE
Usage: is_nonempty_doctree(R)	
Not empty test	
– <i>The following properties should hold at call time:</i>	
R is currently a term which is not a free variable.	(nonvar/1)
Intermediate tree representation for documentation	(doctree/1)
empty_doctree/1:	PREDICATE
Usage: empty_doctree(R)	
Empty	
– <i>The following properties should hold at call time:</i>	
R is a free variable.	(var/1)
Intermediate tree representation for documentation	(doctree/1)
doctree_insert_end/3:	PREDICATE
Usage: doctree_insert_end(A0,Elem,A)	
Insert Elem in A0 at the end, obtaining A	
– <i>Call and exit should be compatible with:</i>	
Intermediate tree representation for documentation	(doctree/1)
Intermediate tree representation for documentation	(doctree/1)
Intermediate tree representation for documentation	(doctree/1)

doctree_insert_before_section/3:	PREDICATE
Usage: <code>doctree_insert_before_section(A0,Elem,A)</code>	
Insert Elem in A0 before the first section, obtaining A	
– <i>Call and exit should be compatible with:</i>	
Intermediate tree representation for documentation	(doctree/1)
Intermediate tree representation for documentation	(doctree/1)
Intermediate tree representation for documentation	(doctree/1)
doctree_concat/3:	PREDICATE
No further documentation available for this predicate.	
doclink/1:	REGTYPE
Usage:	
A link to a document label	
doclabel/1:	REGTYPE
Usage:	
An internal label	
doclink_at/2:	PREDICATE
No further documentation available for this predicate.	
doclink_is_local/1:	PREDICATE
No further documentation available for this predicate.	
doctokens/1:	REGTYPE
Usage:	
Primitive doctree subset (ready for output, not further reducible)	
section_prop/2:	PREDICATE
No further documentation available for this predicate.	
section_select_prop/3:	PREDICATE
No further documentation available for this predicate.	

doctree_save/2:	PREDICATE
Usage:	
– <i>Call and exit should be compatible with:</i>	
Arg1 is an atom.	(atm/1)
Intermediate tree representation for documentation	(doctree/1)
– <i>The following properties should hold at call time:</i>	
Arg1 is currently a term which is not a free variable.	(nonvar/1)
Arg2 is currently a term which is not a free variable.	(nonvar/1)
doctree_restore/2:	PREDICATE
Usage:	
– <i>Call and exit should be compatible with:</i>	
Arg1 is an atom.	(atm/1)
Intermediate tree representation for documentation	(doctree/1)
– <i>The following properties should hold at call time:</i>	
Arg1 is currently a term which is not a free variable.	(nonvar/1)
doctree_simplify/2:	PREDICATE
No further documentation available for this predicate.	
doctree_putvars/5:	PREDICATE
Usage: doctree_putvars(R0,DocSt,PDict,VarDict,R)	
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.	
– <i>The following properties should hold at call time:</i>	
Intermediate tree representation for documentation	(doctree/1)
docstate(DocSt)	(docstate/1)
– <i>The following properties should hold upon exit:</i>	
Intermediate tree representation for documentation	(doctree/1)
doctree_scan_and_save_refs/2:	PREDICATE
Usage: doctree_scan_and_save_refs(R,DocSt)	
Scan and save the references of the doctree	
– <i>The following properties should hold at call time:</i>	
Intermediate tree representation for documentation	(doctree/1)
docstate(DocSt)	(docstate/1)
doctree_prepare_docst_translate_and_write/3:	PREDICATE
No further documentation available for this predicate.	

doctree_to_rawtext/3:	PREDICATE
Usage: <code>doctree_to_rawtext(X,DocSt,Y)</code>	
Y is a simplified raw text representation of the X	
– <i>Call and exit should be compatible with:</i>	
Intermediate tree representation for documentation	(<code>doctree/1</code>)
<code>docstate(DocSt)</code>	(<code>docstate/1</code>)
Y is a string (a list of character codes).	(<code>string/1</code>)
doctree_translate_and_write/3:	PREDICATE
No further documentation available for this predicate.	
escape_string/4:	PREDICATE
Usage:	
Escapes needed characters in input string as needed for the target format.	
– <i>The following properties should hold upon exit:</i>	
Arg1 is currently instantiated to an atom.	(<code>atom/1</code>)
Arg2 is a string (a list of character codes).	(<code>string/1</code>)
<code>docstate(Arg3)</code>	(<code>docstate/1</code>)
Arg4 is a string (a list of character codes).	(<code>string/1</code>)
is_version/1:	PREDICATE
No further documentation available for this predicate.	
version_patch/2:	PREDICATE
No further documentation available for this predicate.	
version_date/2:	PREDICATE
No further documentation available for this predicate.	
version_numstr/2:	PREDICATE
Usage: <code>version_numstr(Version,Str)</code>	
Obtain the string <code>Str</code> representation of version <code>Version</code> (except date)	
version_string/2:	PREDICATE
Usage: <code>version_string(Version,Str)</code>	
Obtain the string <code>Str</code> representation of version <code>Version</code> (including date)	
insert_show_toc/3:	PREDICATE
Usage: <code>insert_show_toc(R0,DocSt,R)</code>	
Insert the command to show the table of contents in a given <code>doctree/1</code> . The right place may be different depending on the chosen backend.	

17.3 Documentation on multifiles (autodoc_doctree)

autodoc_rw_command_hook/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

autodoc_escape_string_hook/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

18 Handling the Document Structure

Author(s): Jose F. Morales.

18.1 Usage and interface (autodoc_structure)

- **Library usage:**
`:- 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.`
- **Imports:**
 - *Application modules:*
`lpdocsrc(src(autodoc_settings)).`
 - *System library modules:*
`filenames, aggregates, terms, make/make_rt.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, dcg, assertions, regtypes, basicmodes, fsyntax.`

18.2 Documentation on exports (autodoc_structure)

- | | |
|--|-----------|
| docstr_node/4:
No further documentation available for this predicate. The predicate is of type <i>data</i> . | PREDICATE |
| clean_docstr/0:
No further documentation available for this predicate. | PREDICATE |
| parse_structure/0:
No further documentation available for this predicate. | PREDICATE |
| standalone_docstr/1:
No further documentation available for this predicate. | PREDICATE |
| get_mainmod/1:
No further documentation available for this predicate. | PREDICATE |

get_mainmod_spec/1:

No further documentation available for this predicate.

PREDICATE

all_component_specs/1:

No further documentation available for this predicate.

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:**
 - :- use_module(library(autodoc_settings)).
- **Exports:**
 - *Predicates:*
 - 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/4, xdvi/1, xdvisize/1, bibtex/1, tex/1,
 - texindex/1, dvips/1, ps2pdf/1, makeinfo/1, makertf/1, rtftohlp/1, convertc/1.
- **Imports:**
 - *System library modules:*
 - system_extra, make/make_rt, aggregates, lpdist/distutils.
 - *Internal (engine) modules:*
 - term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
 - exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
 - compare, term_typing, hiord_rt, debugger_support.
 - *Packages:*
 - prelude, nonpure, dcg, assertions, regtypes, fsyntax.

19.2 Documentation on exports (autodoc_settings)

lpdoc_option/1: PREDICATE
 Defines the global options of lpdoc. The predicate is of type *data*.

verify_settings/0: PREDICATE
 No further documentation available for this predicate.

check_setting/1: PREDICATE
 No further documentation available for this predicate.

setting_value_or_default/2: Usage: <code>setting_value_or_default(Var,Value)</code> Returns in <code>Value</code> the value of the variable <code>Var</code> . In case this variable does not exists, it returns a default value. If there is no default value for the variable <code>Var</code> it fails.	PREDICATE
setting_value_or_default/3: No further documentation available for this predicate.	PREDICATE
setting_value/2: No further documentation available for this predicate.	PREDICATE
all_setting_values/2: No further documentation available for this predicate.	PREDICATE
get_command_option/1: No further documentation available for this predicate.	PREDICATE
requested_file_formats/1: Usage: <code>requested_file_formats(F)</code> F is a requested file format	PREDICATE
load_vpaths/0: No further documentation available for this predicate.	PREDICATE
viewer/4: No further documentation available for this predicate.	PREDICATE
xdvi/1: No further documentation available for this predicate.	PREDICATE
xdvisize/1: No further documentation available for this predicate.	PREDICATE
bibtex/1: No further documentation available for this predicate.	PREDICATE

tex/1: No further documentation available for this predicate.	PREDICATE
texindex/1: No further documentation available for this predicate.	PREDICATE
dvips/1: No further documentation available for this predicate.	PREDICATE
ps2pdf/1: No further documentation available for this predicate.	PREDICATE
makeinfo/1: No further documentation available for this predicate.	PREDICATE
makertf/1: No further documentation available for this predicate.	PREDICATE
rtftohlp/1: No further documentation available for this predicate.	PREDICATE
convertc/1: No further documentation available for this predicate.	PREDICATE

LPdoc Backends



20 Texinfo Backend

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

20.1 Usage and interface (autodoc_texinfo)

- **Library usage:**
`:- use_module(library(autodoc_texinfo)).`
- **Exports:**
 - *Predicates:*
`infodir_base/2.`
 - *Multifiles:*
`autodoc_escape_string_hook/5, autodoc_rw_command_hook/4, autodoc_finish_hook/1, autodoc_gen_alternative_hook/2.`
- **Imports:**
 - *Application modules:*
`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)),
lpdocsrc(src(comments)), lpdocsrc(src(autodoc_aux)).`
 - *System library modules:*
`lists, terms, format, format_to_string, messages, system, make/make_rt,
lpdist/ciao_config_options, file_utils, system_extra.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare,
term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, assertions, regtypes, fsyntax.`

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

Usage: autodoc_rw_command_hook(Backend,DocSt,Command,NewCommand)

– *The following properties should hold at call time:*

Backend is a supported backend.

(backend_id/1)

docstate(DocSt)

(docstate/1)

Intermediate tree representation for documentation

(doctree/1)

Intermediate tree representation for documentation

(doctree/1)

The predicate is *multifile*.**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*.

21 HTML Backend

Author(s): Jose F. Morales.

21.1 Usage and interface (autodoc_html)

- **Library usage:**

```
:- use_module(library(autodoc_html)).
```

- **Imports:**

- *Application modules:*

```
lpdocsrc(src(autodoc_state)),      lpdocsrc(src(autodoc_structure)),
lpdocsrc(src(autodoc_filesystem)), lpdocsrc(src(autodoc_
doctree)), lpdocsrc(src(autodoc_index)), lpdocsrc(src(autodoc_refsdb)),
lpdocsrc(src(autodoc_images)),      lpdocsrc(src(autodoc_
settings)), lpdocsrc(src(comments)), lpdocsrc(src(autodoc_html_template)),
lpdocsrc(src(pbundle_download)), lpdocsrc(src(autodoc_html_resources)).
```

- *System library modules:*

```
lists, dict, format_to_string, system, file_utils.
```

- *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
```

- *Packages:*

```
prelude, nonpure, assertions, regtypes, fsyntax.
```

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

Usage: autodoc_rw_command_hook(Backend,DocSt,Command,NewCommand)

- *The following properties should hold at call time:*

Backend is a supported backend.	(backend_id/1)
docstate(DocSt)	(docstate/1)
Intermediate tree representation for documentation	(doctree/1)
Intermediate tree representation for documentation	(doctree/1)

The predicate is *multifile*.

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*.

22 Resource Handling for the HTML Backend

Author(s): Jose F. Morales.

22.1 Usage and interface (autodoc_html_resources)

- **Library usage:**
`:- use_module(library(autodoc_html_resources)).`
- **Exports:**
 - *Predicates:*
`prepare_web_skel/1, prepare_mathjax/0, using_mathjax/1.`
- **Imports:**
 - *Application modules:*
`lpdocsrc(src(autodoc)), lpdocsrc(src(autodoc_settings)),
lpdocsrc(src(autodoc_filesystem)).`
 - *System library modules:*
`messages, file_utils, system_extra, dirutils, terms.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, assertions, fsyntax.`

22.2 Documentation on exports (autodoc_html_resources)

- | | |
|--|-----------|
| prepare_web_skel/1:
No further documentation available for this predicate. | PREDICATE |
| prepare_mathjax/0:
No further documentation available for this predicate. | PREDICATE |
| using_mathjax/1:
No further documentation available for this predicate. | PREDICATE |

23 Template Support for the HTML Backend

Author(s): Jose F. Morales.

23.1 Usage and interface (autodoc_html_template)

- **Library usage:**
`:- use_module(library(autodoc_html_template)).`
- **Exports:**
 - *Predicates:*
`img_url/2, fmt_html_template/3.`
- **Imports:**
 - *Application modules:*
`lpdocsrc(src(autodoc_settings)).`
 - *System library modules:*
`bundle_registry/bundle_registry_load, messages, aggregates, system, file_`
`utils, system_extra, dirutils, lists, terms, make/make_rt, pillow/html.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,`
`exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_`
`compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, ciaopaths, assertions, fsyntax.`

23.2 Documentation on exports (autodoc_html_template)

- img_url/2:** PREDICATE
Usage: `img_url(Name,Url)`
 Obtain the URL where image `Name` is or will be found.
 – *Call and exit should be compatible with:*
- | | |
|--|---------------------------|
| Name is an atom. | (<code>atom</code> /1) |
| Url is a string (a list of character codes). | (<code>string</code> /1) |
-
- fmt_html_template/3:** PREDICATE
 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:**
 - `:- use_module(library(autodoc_man)).`
- **Imports:**
 - *Application modules:*
 - `lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_doctree)), lpdocsrc(src(autodoc_images)), lpdocsrc(src(autodoc_aux)), lpdocsrc(src(comments)).`
 - *System library modules:*
 - `lists, format_to_string.`
 - *Internal (engine) modules:*
 - `term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
 - `prelude, nonpure, assertions, regtypes, fsyntax.`

24.2 Documentation on multifiles (autodoc_man)

autodoc_rw_command_hook/4: PREDICATE

Usage: `autodoc_rw_command_hook(Backend, DocSt, Command, NewCommand)`

- *The following properties should hold at call time:*

Backend is a supported backend.	(backend_id/1)
docstate(DocSt)	(docstate/1)
Intermediate tree representation for documentation	(doctree/1)
Intermediate tree representation for documentation	(doctree/1)

The predicate is *multifile*.

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:**
`:- use_module(library(autodoc_filesystem)).`
- **Exports:**
 - *Predicates:*
`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:*
`filename/1, basename/1, subtarget/1.`
- **Imports:**
 - *Application modules:*
`lpdocsrc(src(autodoc_settings)), lpdocsrc(src(autodoc_structure)),`
`lpdocsrc(src(autodoc_state)).`
 - *System library modules:*
`aggregates, system_extra, terms, dirutils, system, lpdist/makedir_aux,`
`lpdist/ciao_bundle_db.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,`
`exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_`
`compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, dcg, assertions, regtypes, basicmodes, fsyntax.`

25.2 Documentation on exports (autodoc_filesystem)

filename/1: REGTYPE
Usage: filename(X)
 X is the name of a file.

basename/1: REGTYPE
Usage: basename(X)
 X is the base name of a file (without extension).

subtarget/1:	REGTYPE
Usage:	
The kind of intermediate/final results for a single documentation processing unit (module).	
file_format_name/2:	PREDICATE
No further documentation available for this predicate.	
supported_file_format/1:	PREDICATE
No further documentation available for this predicate.	
file_format_provided_by_backend/3:	PREDICATE
Usage: <code>file_format_provided_by_backend(Ext,Backend,Subtarget)</code>	
Backend is the backend that generates files with format Ext	
– <i>Call and exit should be compatible with:</i>	
Ext is an atom.	(atm/1)
Backend is a supported backend.	(backend_id/1)
Subtarget is an atom.	(atm/1)
clean_fs_db/0:	PREDICATE
Usage:	
Clean the cached information for the filesystem mapping of the documentaton generation.	
get_output_dir/2:	PREDICATE
Usage: <code>get_output_dir(Backend,Dir)</code>	
Obtain the Dir directory where the documentation files are generated. Note that this is not the installation directory.	
get_cache_dir/2:	PREDICATE
Usage: <code>get_cache_dir(Backend,Dir)</code>	
Obtain the Dir directory where final documentation files will be stored	
ensure_output_dir/1:	PREDICATE
No further documentation available for this predicate.	
ensure_cache_dir/1:	PREDICATE
No further documentation available for this predicate.	

main_absfile_in_format/2:	PREDICATE
Usage: <code>main_absfile_in_format(Ext,File)</code>	
File is the absolute file name for the documentation in Ext format of the <i>main</i> module	
main_absfile_for_subtarget/3:	PREDICATE
No further documentation available for this predicate.	
absfile_for_aux/3:	PREDICATE
Usage: <code>absfile_for_aux(AuxName,Backend,AbsFile)</code>	
Absolute file for an auxiliary output file (e.g. CSS, images, etc.)	
absfile_for_subtarget/4:	PREDICATE
No further documentation available for this predicate.	
main_output_name/2:	PREDICATE
No further documentation available for this predicate.	
get_subbase/3:	PREDICATE
Usage: <code>get_subbase(Base,Sub,SubBase)</code>	
SubBase is the name for the sub-file (Sub) associated with Base	
– <i>The following properties should hold upon exit:</i>	
Base is the base name of a file (without extension).	(<code>basename/1</code>)
Sub is an atom.	(<code>atom/1</code>)
SubBase is the base name of a file (without extension).	(<code>basename/1</code>)
absfile_to_relfile/3:	PREDICATE
Usage: <code>absfile_to_relfile(A,Backend,B)</code>	
Obtain the relative path, w.r.t. the output directory, of an absolute file. This is useful, e.g., for URLs.	
clean_all/0:	PREDICATE
No further documentation available for this predicate.	
clean_docs_no_texi/0:	PREDICATE
No further documentation available for this predicate.	

clean_all_temporal/0:

No further documentation available for this predicate.

PREDICATE

clean_intermediate/0:

No further documentation available for this predicate.

PREDICATE

clean_tex_intermediate/0:

No further documentation available for this predicate.

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:**

```
:- 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.
```
- **Imports:**
 - *Application modules:*

```
lpdocsrc(src(autodoc_state)),      lpdocsrc(src(autodoc_filesystem)),
lpdocsrc(src(autodoc_doctree)),    lpdocsrc(src(autodoc_structure)),
lpdocsrc(src(autodoc_refsdb)).
```
 - *System library modules:*

```
dict, lists, aggregates.
```
 - *Internal (engine) modules:*

```
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
```
 - *Packages:*

```
prelude, nonpure, dcg, assertions, regtypes.
```

26.2 Documentation on exports (autodoc_index)

get_idxsub/2: PREDICATE
 No further documentation available for this predicate.

get_idxbase/3: PREDICATE
 No further documentation available for this predicate.

typeindex/5: PREDICATE
Usage: `typeindex(Type, Index, IType, Name, Comment)`
`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: PREDICATE
No further documentation available for this predicate.

is_index_cmd/1: PREDICATE
No further documentation available for this predicate.

codetype/1: PREDICATE
No further documentation available for this predicate.

normalize_index_cmd/3: PREDICATE
No further documentation available for this predicate.

fmt_idx_env/7: PREDICATE
No further documentation available for this predicate.

fmt_index/3: PREDICATE
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:**
:- 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.
- **Imports:**
 - *Application modules:*
lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_doctree)),
lpdocsrc(src(autodoc_structure)), lpdocsrc(src(autodoc_filesystem)),
lpdocsrc(src(autodoc_bibrefs)), .(autodoc_structure).
 - *System library modules:*
aggregates, lists.
 - *Internal (engine) modules:*
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
 - *Packages:*
prelude, nonpure, dcg, assertions, regtypes.

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:
Prepare references for the translation of the current file.
– *Call and exit should be compatible with:*
docstate(Arg1) (docstate/1)

- clean_current_refs/1:** PREDICATE
Usage:
 Clean the data stored by `prepare_current_refs/1`.
 – *Call and exit should be compatible with:*
 `docstate(Arg1)` (`docstate/1`)
- sectree/1:** REGTYPE
Usage:
 A tree of sections
- sectree_resolve/3:** PREDICATE
Usage: `sectree_resolve(LabelName,Tree,Link)`
 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:**
 - `:- use_module(library(autodoc_errors)).`
- **Exports:**
 - *Predicates:*
 - `error_text/3.`
- **Imports:**
 - *Internal (engine) modules:*
 - `term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
 - `prelude, nonpure, assertions, regtypes, basicmodes.`

28.2 Documentation on exports (autodoc_errors)

error_text/3:

No further documentation available for this predicate.

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:**
`:- use_module(library(autodoc_bibrefs)).`
- **Exports:**
 - *Predicates:*
`resolve_bibliography/1, parse_commands/3.`
- **Imports:**
 - *Application modules:*
`lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_`
`doctree)), lpdocsrc(src(autodoc_refsdb)), lpdocsrc(src(autodoc_aux)),`
`lpdocsrc(src(autodoc_settings)), lpdocsrc(src(autodoc_parse)).`
 - *System library modules:*
`dict, aggregates, terms, file_utils, lists, format, make/make_rt, system_extra.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,`
`exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_`
`compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, dcg, assertions, regtypes.`

29.2 Documentation on exports (autodoc_bibrefs)

resolve_bibliography/1:

PREDICATE

Usage: `resolve_bibliography(DocSt)`

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 `bibitem` commands.

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:*

`docstate(DocSt)` (docstate/1)

parse_commands/3:

PREDICATE

No further documentation available for this predicate.

30 Auxiliary Definitions

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

30.1 Usage and interface (autodoc_aux)

- **Library usage:**
`:- use_module(library(autodoc_aux)).`
- **Exports:**
 - *Predicates:*
`all_vars/1, read_file/2, ascii_blank_lines/2, sh_exec/2.`
- **Imports:**
 - *Application modules:*
`lpdocsrc(src(autodoc_settings)).`
 - *System library modules:*
`messages, system, system_extra, lists.`
 - *Internal (engine) modules:*
`term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_compare, term_typing, hiord_rt, debugger_support.`
 - *Packages:*
`prelude, nonpure, assertions, regtypes, basicmodes, fsyntax.`

30.2 Documentation on exports (autodoc_aux)

all_vars/1: No further documentation available for this predicate.	PREDICATE
read_file/2: No further documentation available for this predicate.	PREDICATE
ascii_blank_lines/2: No further documentation available for this predicate.	PREDICATE
sh_exec/2: No further documentation available for this predicate.	PREDICATE

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:**
:- use_module(library(autodoc_images)).
- **Exports:**
 - *Predicates:*
locate_and_convert_image/4, clean_image_cache/0.
- **Imports:**
 - *Application modules:*
lpdocsrc(src(autodoc_state)), lpdocsrc(src(autodoc_filesystem)),
lpdocsrc(src(autodoc_settings)), lpdocsrc(src(autodoc_aux)).
 - *System library modules:*
terms, make/make_rt, system_extra, system, errhandle, messages, format.
 - *Internal (engine) modules:*
term_basic, arithmetic, atomic_basic, basic_props, basiccontrol, data_facts,
exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_
compare, term_typing, hiord_rt, debugger_support.
 - *Packages:*
prelude, nonpure, dcg, assertions, regtypes, fsyntax.

31.2 Documentation on exports (autodoc_images)

locate_and_convert_image/4:

PREDICATE

Usage:

locate_and_convert_image(SrcSpecS, AcceptedFormats, DocSt, TargetFileS)

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). (string/1)

AcceptedFormats is a list of atoms. (list/2)

docstate(DocSt) (docstate/1)

TargetFileS is a string (a list of character codes). (string/1)

clean_image_cache/0:

PREDICATE

Usage:

Clean the cache for image copy/conversions.

References

- [Bue95] F. Bueno.
The CIAO Multiparadigm Compiler: A User's Manual.
Technical Report CLIP8/95.0, Facultad de Informática, UPM, June 1995.
- [Bue98] F. Bueno.
Using Assertions for Static Debugging of CLP: A Manual.
Technical Report CLIP1/98.0, DISCIPL Project/CLIP Group, UPM, June 1998.
- [DEDC96] P. Deransart, A. Ed-Dbali, and L. Cervoni.
Prolog: The Standard.
Springer-Verlag, 1996.
- [DL93] S. K. Debray and N. W. Lin.
Cost Analysis of Logic Programs.
ACM Transactions on Programming Languages and Systems, 15(5):826–875,
November 1993.
- [DLGH97] S.K. Debray, P. López-García, and M. Hermenegildo.
Non-Failure Analysis for Logic Programs.
In *1997 International Conference on Logic Programming*, pages 48–62, Cambridge,
MA, June 1997. MIT Press, Cambridge, MA.
- [DLGHL97] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Lower Bound Cost Estimation for Logic Programs.
In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press,
Cambridge, MA, October 1997.
- [Her99] M. Hermenegildo.
A Documentation Generator for Logic Programming Systems.
Technical Report CLIP10/99.0, Facultad de Informática, UPM, September 1999.
- [Her00] M. Hermenegildo.
A Documentation Generator for (C)LP Systems.
In *International Conference on Computational Logic, CL2000*, number 1861 in
LNAI, pages 1345–1361. Springer-Verlag, July 2000.
- [JL88] D. Jacobs and A. Langen.
Compilation of Logic Programs for Restricted And-Parallelism.
In *European Symposium on Programming*, pages 284–297, 1988.
- [JM94] J. Jaffar and M.J. Maher.
Constraint Logic Programming: A Survey.
Journal of Logic Programming, 19/20:503–581, 1994.
- [Knu84] D. Knuth.
Literate programming.
Computer Journal, 27:97–111, 1984.
- [LGHD96] P. López-García, M. Hermenegildo, and S. K. Debray.
A Methodology for Granularity Based Control of Parallelism in Logic Programs.
Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation,
21(4–6):715–734, 1996.
- [MH89] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Ab-
stract Interpretation.
In *1989 North American Conference on Logic Programming*, pages 166–189. MIT
Press, October 1989.

- [PBH97] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Debugging of Constraint Logic Programs.
Technical Report CLIP2/97.1, Facultad de Informática, UPM, July 1997.
- [PBH98] G. Puebla, F. Bueno, and M. Hermenegildo.
A Framework for Assertion-based Debugging in Constraint Logic Programming.
In *Proceedings of the JICSLP'98 Workshop on Types for CLP*, pages 3–15, Manchester, UK, June 1998.
- [PBH00] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Constraint Logic Programs.
In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

Library/Module Index

A

assertions	41
assertions_props	51
autodoc	119
autodoc_aux	167
autodoc_bibrefs	165
autodoc_doctree	131
autodoc_errors	163
autodoc_filesystem	155
autodoc_html	147
autodoc_html_resources	149
autodoc_html_template	151
autodoc_images	169
autodoc_index	159
autodoc_man	153
autodoc_refsdb	161
autodoc_settings	139
autodoc_state	123
autodoc_structure	137
autodoc_texinfo	145

B

basic_props	63
-------------	----

C

comments	25
----------	----

E

example_module	103
----------------	-----

G

Generating	15
------------	----

L

lpdoc_examples	97
lpdoc_install	115

M

meta_props	95
------------	----

N

native_props	81
--------------	----

R

regtypes	57
rtchecks	111

U

unittest	113
----------	-----

Predicate/Method Index

A

absfile_for_aux/3	157
absfile_for_subtarget/4	157
absfile_to_relfile/3	157
all_component_specs/1	138
all_indices/2	128
all_setting_values/2	140
all_vars/1	167
ascii_blank_lines/2	167
autodoc_compute_grefs/3	121
autodoc_escape_string_hook/5	136, 145, 147
autodoc_finish/1	122
autodoc_finish_hook/1	122, 146, 147, 153
autodoc_gen_alternative/2	122
autodoc_gen_alternative_hook/2	122, 146, 148, 153
autodoc_gen_doctree/5	121
autodoc_rw_command_hook/4	136, 145, 147, 153
autodoc_translate_doctree/3	121

B

backend_alt_format/2	125
backend_ignores_components/1	124
bibtex/1	140
bind_dict_varnames/1	128

C

callme/2	96
check/1	48
check_setting/1	139
clean_all/0	157
clean_all_temporal/0	157
clean_current_refs/1	161
clean_docs_no_texi/0	157
clean_docstr/0	137
clean_fs_db/0	156
clean_image_cache/0	169
clean_intermediate/0	158
clean_tex_intermediate/0	158
cmd_type/1	131
codetype/1	160
compute_refs_and_biblio/1	161
convertc/1	141

D

doc_assertion_read/9	128
doc_id_type/3	32
doclink_at/2	133
doclink_is_local/1	133
docst_backend/2	125
docst_currmod/2	125
docst_currmod_is_main/1	126
docst_gdata/3	127
docst_gdata_clean/1	128
docst_gdata_query/2	127
docst_gdata_query/3	127
docst_gdata_restore/1	127
docst_gvar_restore/2	128
docst_gvar_save/2	128
docst_has_index/2	128
docst_inputfile/2	125
docst_mdata_assertz/2	127
docst_mdata_clean/1	127
docst_mdata_save/1	127
docst_message/2	126
docst_message/3	126
docst_modname/2	126
docst_modtype/2	129
docst_mvar_get/3	127
docst_mvar_lookup/3	127
docst_mvar_replace/4	127
docst_new_no_src/4	126
docst_new_sub/3	126
docst_new_with_src/6	126
docst_no_components/1	126
docst_opt/2	126
docst_opts/2	125
docst_set_currmod/3	125
docst_set_opts/3	125
docstr_node/4	137
doctree_concat/3	133
doctree_insert_before_section/3	132
doctree_insert_end/3	132
doctree_is_empty/1	132
doctree_prepare_docst_translate_and_write/3	134
doctree_putvars/5	134
doctree_restore/2	134
doctree_save/2	133
doctree_scan_and_save_refs/2	134
doctree_simplify/2	134
doctree_to_rawtext/3	135
doctree_translate_and_write/3	135
dvips/1	141

E

empty_doctree/1	132
ensure_cache_dir/1	156
ensure_output_dir/1	156
ensure_output_dir_prepared/2	120
error_text/3	163
escape_string/4	135

F

false/1	49
file_format_name/2	156
file_format_provided_by_backend/3	156
fnt_html_template/3	151
fnt_idx_env/7	160
fnt_index/3	160
fnt_infodir_entry/3	121

G

get_autodoc_opts/3	121
get_cache_dir/2	156
get_command_option/1	140
get_doc/4	128
get_doc_changes/3	128
get_doc_pred_varnames/2	128
get_first_loc_for_pred/3	129
get_idxbase/3	159
get_idxsub/2	159
get_mainmod/1	137
get_mainmod_spec/1	138
get_mod_doc/3	128
get_output_dir/2	156
get_subbase/3	157

I

idx_get_indices/3	160
img_url/2	151
index_comment/2	120
infodir_base/2	145
insert_show_toc/3	135
is/2	110
is_index_cmd/1	160
is_nonempty_doctree/1	132
is_version/1	135

L

labgen_clean/1	126
labgen_get/2	127
labgen_init/1	126
load_vpaths/0	140
locate_and_convert_image/4	169
lpdoc_option/1	139

M

main_absfile_for_subtarget/3	157
main_absfile_in_format/2	156
main_output_name/2	157
makeinfo/1	141
makertf/1	141
mytype/1	106

N

normalize_index_cmd/3	160
-----------------------------	-----

O

option_comment/2	124
------------------------	-----

P

p/1	105
p/3	108
p/5	105
parse_commands/3	165
parse_structure/0	137
pred_has_docprop/2	128
prepare_current_refs/1	161
prepare_mathjax/0	149
prepare_web_skel/1	149
ps2pdf/1	141

Q

q/1	107
q/2	104

R

r/1	104
read_file/2	167
requested_file_formats/1	140
reset_output_dir_db/0	120
resolve_bibliography/1	165

rtftohlp/1 141

S

s/1 106
 section_prop/2 133
 section_select_prop/3 133
 secttree_resolve/3 162
 setting_value/2 140
 setting_value_or_default/2 139
 setting_value_or_default/3 140
 sh_exec/2 167
 standalone_docstr/1 137
 supported_file_format/1 156

T

t/5 106
 tex/1 140
 texindex/1 141
 top_suffix/2 125
 true/1 49
 trust/1 48
 typeindex/5 159

U

u/3 105
 using_mathjax/1 149

V

verify_settings/0 139
 version_date/2 135
 version_numstr/2 135
 version_patch/2 135
 version_string/2 135
 viewer/4 140

W

w/1 106

X

xdvi/1 140
 xdvisize/1 140

Property Index

B

bind_ins/1 79

C

call/2 95
 clique/1 81
 clique_1/1 82
 compat/1 82
 compat/2 75
 constraint/1 82
 covered/1 82
 covered/2 83

D

deprecated/1 76
 docstring/1 25, 56

E

equiv/2 79
 error_free/1 79
 eval/1 79
 exception/1 83
 exception/2 83

F

fails/1 83
 filter/2 79
 finite_solutions/1 83

H

have_choicepoints/1 83
 head_pattern/1 52

I

indep/1 84
 indep/2 84
 inst/2 75
 instance/1 84
 instance/2 93
 is_det/1 84
 iso/1 75

L

linear/1 84
 long/1 105

M

member/2 71
 memo/1 79
 mshare/1 85
 mut_exclusive/1 85

N

nabody/1 54
 native/1 77
 native/2 77
 no_choicepoints/1 85
 no_exception/1 86
 no_exception/2 86
 no_rtcheck/1 78
 no_signal/1 86
 no_signal/2 86
 non_det/1 86
 nonground/1 86
 not_covered/1 86
 not_fails/1 87
 not_further_inst/2 76
 not_mut_exclusive/1 87
 num_solutions/2 87

P

pe_type/1 79
 possibly_fails/1 87
 possibly_nondet/1 88
 prop/2 96
 prop_abs/1 96

R

regtype/1 77
 regtype/2 96
 relations/2 88
 rtcheck/1 77
 rtcheck/2 78

S

sideff/2.....	76
sideff_hard/1.....	88
sideff_pure/1.....	88
sideff_soft/1.....	89
signal/1.....	89
signal/2.....	89
signals/2.....	89
size/2.....	89
size/3.....	90
size_lb/2.....	90
size_metric/3.....	90
size_metric/4.....	91
size_o/2.....	90
size_ub/2.....	90
solutions/2.....	87
steps/2.....	91
steps_lb/2.....	91
steps_o/2.....	91
steps_ub/2.....	92
stringcommand/1.....	26
succeeds/1.....	91
supported_option/1.....	124

T

tau/1.....	92
terminates/1.....	92
test_type/2.....	92
throws/2.....	93

U

user_output/2.....	93
--------------------	----

Regular Type Index

A

aorb/1	104
assrt_body/1	51
assrt_status/1	55
assrt_type/1	56
atm/1	66
atm_or_atm_list/1	74

B

backend_id/1	124
bar/1	103
basename/1	155
baz/1	103

C

c_assrt_body/1	54
callable/1	69
character_code/1	73
complex_arg_property/1	53
complex_goal_property/1	53
constant/1	68

D

dictionary/1	54
doclabel/1	133
doclink/1	133
docstate/1	125
doctokens/1	133
doctree/1	132

F

filename/1	155
filetype/1	32
flag_values/1	79
flt/1	65

G

g_assrt_body/1	55
gnd/1	67
gndstr/1	68

I

int/1	64
-------	----

Y

ynd_date/1	39
------------	----

L

list/1	70, 107
list/2	70, 109
list_or_aorb/2	104

M

modtype/1	129
-----------	-----

N

nlist/2	71
nnegint/1	64
num/1	66
num_code/1	74

O

operator_specifier/1	69
----------------------	----

P

predfunctor/1	56
predname/1	74
property_conjunction/1	53
property_starterm/1	53
propfunctor/1	56

S

s_assrt_body/1	54
sectree/1	162
sequence/2	72
sequence_or_list/2	72
string/1	73
struct/1	67
subtarget/1	156

T

term/1	63
time_struct/1	39
tree_of/2	104

V

version_descriptor/1	32
version_maintenance_type/1	39
version_number/1	38

Declaration Index

C

calls/1	43
calls/2	44
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

modedef/1	47
-----------------	----

P

pred/1	42
pred/2	43
prop/1	45
prop/2	46

R

regtype/1	60
regtype/2	61

S

success/1	44
success/2	44

T

test/1	44
test/2	44
texec/1	43
texec/2	43

Concept Index

, 6
 •
 .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
 @em 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
 @file command 29
 @footnote command 27
 @H command 31
 @hfill command 27
 @href command 29
 @i command 31
 @image command 30
 @include command 30
 @includedef command 30
 @includefact command 30
 @includeverbatim command 30
 @index command 28
 @iso command 31
 @item command 26, 27
 @j command 31
 @key command 27
 @l command 31
 @L command 31
 @lib command 28
 @math command 30
 @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
 @subsubsection command 27
 @t command 31
 @today command 29
 @tt command 27
 @u command 31
 @v command 31
 @var command 29
 @version command 29

A

A4 paper	17
abstract	34
accents	31
acceptable modes	52
acknowledgements	34
address	33
appendix	34
application	4
assertion body syntax	51, 54, 55
assertions	25
author	33
automatic documentation	119
automatic documentation library	119
avoiding indentation	27

B

bibliographic citations	29
bibliographic entries	16
bibliographic entry	29
bibtex	16, 29
blank lines	27
bold face	27
bug	36

C

calls assertion	43, 44
check assertion	48
Ciao	15
comment	26
comment assertion	48
comments, machine readable	41
comp assertion	45
compatibility properties	57
component files	4
contents area	18
copyright	34

D

date	29
decl assertion	47
description list	26
document structure	16
documentation format	119
documentation strings	32

E

emacs Ciao mode	36
Emacs, accessing info files	19
Emacs, generating manuals from	15
Emacs, LPdoc mode	15
email address	29
email addresses	29
emphasis face	27
encapsulated postscript	30
entry assertion	46
enumerated list	26
escape sequences	26
example of lpdoc use	97
exit assertion	46

F

false assertion	49
fixed format text	27
fixed-width font	27
footnote	27
formatting commands	25, 41
framed box	27

G

generating from Emacs	15
generating manuals	15

H

hard side-effects	88
html index page	18

I

image file	30
images, inserting	30
images, scaling	30
including a predicate definition	30
including an image	30
including code	30
including files	30
including images	30
including or not authors	16
including or not bug info	16
including or not changelog	16
including or not versions, patches	16
indentation, avoiding	27
index pages out of order	18

- info path list 20
 installation 3
 instantiation properties 57
 internals manual 21
 introduction 34
 italics face 27
 item in an itemized list 26
 itemized list 26
- K**
- keyboard key 27
- L**
- LaTeX notation 29
 letter size paper 17
 library 3
 literate programming 18
 log of changes 36
- M**
- machine readable comments 32
 main body 34
 main file 4
 module comment 34
 module declaration 119
- N**
- new item in description list 27
- O**
- one-sided printing 16
- P**
- page numbering, changing 16
 page size, changing 17
 page style, changing 17
 paragraph break 27
 parametric property 96
 parametric regular type abstractions 96
 parametric type functor 60
 parts in a large document 22
 parts in large documents 38
 planned improvement 36
 pred assertion 42, 43
- program section 35
 program subsection 35
 program subsection 35
 Prolog, Ciao 15
 prop assertion 45, 46
 properties of computations 57
 properties of execution states 57
 properties, basic 63
 properties, native 81
 property abstraction 96
- R**
- references 29
 regtype assertion 60, 61
 regular type expression 61
 running unit tests 113
- S**
- section 27
 sections 29
 SETTINGS.pl 15
 sharing pieces of text 30
 sharing sets 85
 soft side-effects 89
 space, extra lines 27
 spcae, horizontal fill 27
 special characters 31
 strong face 27
 subsection 27
 subsection 27
 subsection 27
 subtitle 33
 success assertion 44
 supported documentation formats 121
 synopsis section of the man page 16
 syntax of formatting commands 26
 system modules 16
- T**
- test assertion 44, 45, 113
 texec assertion 43
 texinfo 119
 texinfo files 4
 textual comments 25
 thesis-like style 17
 title 32, 33
 true assertion 49
 trust assertion 48

two-sided 16
typewriter-like font 27

U

unit tests 113
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..... 103
Anonymous Author 2..... 103

D

Daniel Cabeza 63

E

Edison Mera 81, 111, 113

F

Francisco Bueno 41, 57, 81, 95

G

German Puebla 41

J

Jose F. Morales 119, 123, 131, 137, 139, 145, 147,
149, 151, 153, 155, 159, 161, 165, 167, 169

M

Manuel Hermenegildo . . 15, 25, 41, 51, 57, 63, 81, 97,
115, 119, 123, 131, 145, 153, 163, 165, 167

P

Pedro Lopez 57, 81

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.

.....	6	@apl command.....	28
,		@author command.....	29
','/2.....	72	@b command.....	31
(@begin{alert} command.....	27
(prop)/2.....	96	@begin{cartouche} command.....	27
*		@begin{description} command.....	26
*/2.....	53	@begin{displaymath} command.....	30
.		@begin{enumerate} command.....	26
.(autodoc_structure).....	161	@begin{itemize} command.....	26
.bib files.....	16, 29	@begin{verbatim} command.....	27
.cshrc.....	18	@bf command.....	27
.profile.....	18	@bullet command.....	31
:		@c command.....	31
::/2.....	42	@cindex command.....	28
=		@cite command.....	16, 29
=/2.....	32, 33, 34, 35, 36, 37, 38, 85	@comment command.....	26
=>/2.....	42	@concept command.....	28
@		@copyright command.....	31
@! command.....	31	@d command.....	31
@' command.....	31	@decl command.....	28
@. command.....	31	@defmathcmd/2 command.....	30
@. . command.....	31	@defmathcmd/3 command.....	30
@= command.....	31	@em command.....	27
@? command.....	31	@email command.....	29
@' command.....	31	@end{alert} command.....	27
@~ command.....	31	@end{cartouche} command.....	27
@" command.....	31	@end{description} command.....	27
@^ command.....	31	@end{displaymath} command.....	30
@aa command.....	31	@end{enumerate} command.....	26
@AA command.....	31	@end{itemize} command.....	26
@ae command.....	31	@end{verbatim} command.....	27
@AE command.....	31	@file command.....	29
		@footnote command.....	27
		@H command.....	31
		@hfill command.....	27
		@href command.....	29
		@i command.....	31
		@image command.....	30
		@include command.....	30
		@includedef command.....	30
		@includefact command.....	30
		@includeverbatim command.....	30
		@index command.....	28
		@iso command.....	31
		@item command.....	26, 27
		@j command.....	31
		@key command.....	27
		@l command.....	31

@L command	31
@lib command	28
@math command	30
@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
@subsubsection command	27
@t command	31
@today command	29
@tt command	27
@u command	31
@v command	31
@var command	29
@version command	29
+	
+/1	52
+/2	52
A	
A4 paper	17
absfile_for_aux/3	155, 157
absfile_for_subtarget/4	155, 157
absfile_to_relfile/3	155, 157
abstract	34
accents	31
acceptable modes	52
acknowledgements	34
address	29, 33
aggregates	120, 123, 137, 139, 151, 155, 159, 161, 165
all_component_specs/1	137, 138
all_indices/2	123, 128
all_setting_values/2	139, 140
all_vars/1	167
analyzer output	49
Anonymous Author 1	103
Anonymous Author 2	103
aorb/1	103, 104
appendix	34
application	4
arithexpression/1	110
arithmetic	25, 42, 43, 51, 63, 81, 95, 103, 112, 114, 120, 123, 131, 137, 139, 145, 147, 149, 151, 153, 155, 159, 161, 163, 165, 167, 169
ascii_blank_lines/2	167
assertion body syntax	51, 54, 55
assertion status	43, 44, 45, 47
assertions	3, 22, 23, 25, 41, 42, 51, 60, 63, 81, 95, 103, 112, 114, 119, 120, 123, 131, 137, 139, 145, 147, 149, 151, 153, 155, 159, 161, 163, 165, 167, 169
assertions/assertions_props	42, 60, 120, 123
assertions/assrt_lib	120, 123
assertions/native_props	63, 103
assertions_props	51
assrt_body/1	42, 43, 46, 47, 51, 61
assrt_status/1	43, 44, 45, 46, 47, 51, 55, 61
assrt_type/1	51, 56
atm/1	38, 63, 66, 67, 121, 134, 151, 156, 157, 160
atm_or_atm_list/1	63, 74, 75
atom/1	120, 135, 160
atomic_basic	25, 42, 51, 63, 81, 95, 103, 112, 114, 120, 123, 131, 137, 139, 145, 147, 149, 151, 153, 155, 159, 161, 163, 165, 167, 169
author	33
author indexing	33
autodoc	3, 16, 119
autodoc_aux	167
autodoc_bibrefs	165
autodoc_compute_grefs/3	120, 121
autodoc_doctree	131
autodoc_errors	163
autodoc_escape_string_hook/5	131, 136, 145, 147
autodoc_filesystem	155
autodoc_finish/1	120, 122
autodoc_finish_hook/1	120, 122, 145, 146, 147, 153
autodoc_gen_alternative/2	120, 122
autodoc_gen_alternative_hook/2	120, 122, 145, 146, 148, 153
autodoc_gen_doctree/5	120, 121
autodoc_html	119, 147

autodoc_html_resources 149
 autodoc_html_template 151
 autodoc_images 169
 autodoc_index 159
 autodoc_man 119, 153
 autodoc_refsdbs 161
 autodoc_rw_command_hook/4 131, 136, 145, 147,
 153
 autodoc_settings 139
 autodoc_state 123
 autodoc_structure 137
 autodoc_texinfo 119, 145
 autodoc_translate_doctree/3 120, 121
 autodocformats 3
 automatic documentation 119
 automatic documentation library 3, 119
 avoiding indentation 27

B

backend_alt_format/2 123, 125
 backend_id/1 121, 123, 124, 146, 147, 153, 156
 backend_ignores_components/1 123, 124
 bar/1 103, 109
 basename/1 121, 155, 157
 bash 18
 basic_props .. 3, 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 basic_props.pl 111
 basic_props:regtype/1 57
 basiccontrol... 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 basicmodes ... 103, 120, 123, 131, 137, 155, 163, 167
 baz(Arg3) 109
 baz/1 103, 109
 bibliographic citations 29
 bibliographic entries 16
 bibliographic entry 29
 bibtex 11, 16, 29, 115
 bibtex/1 139, 140
 bind_dict_varnames/1 123, 128
 bind_ins/1 63, 66, 71, 79
 blank lines 27
 bold face 27
 bug 36
 bundle_registry/bundle_registry_load 151

C

c_assert_body/1 43, 44, 46, 51, 54
 c_itf 9
 call(T,A) 110
 call/1 54
 call/2 95
 call_rtc/1 112
 callable/1 63, 69, 76, 78, 87, 95, 96
 callme/2 95, 96
 calls assertion 43, 44
 calls/1 42, 43, 44, 46
 calls/2 42, 44
 character string 41
 character_code/1 63, 73
 check assertion 48
 Check(X) 87
 check/1 42, 48, 49
 check_setting/1 139
 Ciao 3, 15, 18, 19, 20, 97, 115
 Ciao Emacs mode 15
 ciaoc 8, 9
 ciaopaths 151
 ciaopp 81
 clean_all/0 155, 157
 clean_all_temporal/0 155, 157
 clean_current_refs/1 161
 clean_docs_no_texinfo/0 155, 157
 clean_docstr/0 137
 clean_fs_db/0 155, 156
 clean_image_cache/0 169
 clean_intermediate/0 155, 158
 clean_tex_intermediate/0 155, 158
 clique/1 81
 clique_1/1 81, 82
 CLP 3
 cmd_type/1 131
 codetype/1 159, 160
 comment 26
 comment assertion 48
 comment string 52, 54, 55
 comment/2 10, 42, 48
 comments 3, 22, 25
 comments, machine readable 41
 CommentType=ack 34
 CommentType=address 33
 CommentType=appendix 35
 CommentType=author 33

CommentType=bug	36	data_facts.....	25, 42, 51, 63, 81, 95, 103, 112, 114, 120, 123, 131, 137, 139, 145, 147, 149, 151, 153, 155, 159, 161, 163, 165, 167, 169
CommentType=copyright	34	date	29
CommentType=doinclude	37	dcg....	25, 51, 120, 123, 131, 137, 139, 155, 159, 161, 165, 169
CommentType=filetype	38	debugger_support...	25, 42, 51, 63, 81, 95, 103, 112, 114, 120, 123, 131, 137, 139, 145, 147, 149, 151, 153, 155, 159, 161, 163, 165, 167, 169
CommentType=hide	37, 38	decl assertion.....	47
CommentType=logo	33	decl/1	42, 47, 51
CommentType=module	34	decl/2.....	42, 47
CommentType=nodoc	38	deprecated/1	63, 76
CommentType=section.....	35	description list.....	26
CommentType=subsection.....	35	dict	120, 123, 147, 159, 165
CommentType=subsubsection.....	35	dictionary/1.....	51, 54
CommentType=subtitle	33	dir	19
CommentType=subtitle_extra	33	dirutils	149, 151, 155
CommentType=summary.....	34	doc/2	18, 22, 23, 25, 32, 36, 38, 42, 47, 48
CommentType=title	32	doc_assertion_read/9	123, 128
CommentType=usage	35	doc_id_type/3	25, 32
CommentType=version_maintenance	37	doc_structure/1	16
comment/2.....	11	doclabel/1	131, 133
comp assertion.....	45	doclink/1.....	131, 133, 162
comp/1	42, 45, 55	doclink_at/2	131, 133
comp/2.....	42, 45	doclink_is_local/1.....	131, 133
compat/1.....	81, 82	docst_backend/2.....	123, 125
compat/2.....	63, 75	docst_currmod/2.....	123, 125
compatibility properties	57	docst_currmod_is_main/1	123, 126
compatible	51	docst_gdata/3	123, 127
compiler/c_itf.....	120, 123	docst_gdata_clean/1.....	123, 128
compiler/compiler.....	120, 123	docst_gdata_query/2.....	123, 127
complex argument property.....	51, 52, 53, 54, 55	docst_gdata_query/3.....	123, 127
complex goal property.....	52, 54, 55	docst_gdata_restore/1	123, 127
complex_arg_property/1	51, 52, 53, 54, 55	docst_gvar_restore/2	123, 128
complex_goal_property/1	51, 52, 53, 55	docst_gvar_save/2	123, 128
component files.....	4	docst_has_index/2	123, 128
compute_refs_and_biblio/1	161	docst_inputfile/2	123, 125
constant/1	63, 68	docst_mdata_assertz/2	123, 127
constraint/1	81, 82	docst_mdata_clean/1.....	123, 127
contents area.....	18	docst_mdata_save/1.....	123, 127
conversion formats.....	3	docst_message/2.....	123, 126
convert	115	docst_message/3.....	123, 126
convertc/1	139, 141	docst_modname/2.....	123, 126
copyright.....	34	docst_modtype/2.....	123, 129
covered/1	81, 82	docst_mvar_get/3.....	123, 127
covered/2	81, 83	docst_mvar_lookup/3.....	123, 127
csh.....	18, 20	docst_mvar_replace/4	123, 127
D			
Daniel Cabeza.....	63		

docst_new_no_src/4 123, 126
 docst_new_sub/3 123, 126
 docst_new_with_src/6 123, 126
 docst_no_components/1 123, 126
 docst_opt/2 123, 126
 docst_opts/2 123, 125
 docst_set_currmod/3 123, 125
 docst_set_opts/3 123, 125
 docstate(Arg1) 161, 162
 docstate(Arg3) 135
 docstate(DocSt) .. 121, 134, 135, 146, 147, 153, 165,
 169
 docstate/1 ... 121, 123, 125, 134, 135, 146, 147, 153,
 161, 162, 165, 169
 docstr_node/4 137
 docstring/1 ... 25, 32, 33, 34, 35, 36, 41, 48, 51, 52,
 54, 55, 56
 doctokens/1 131, 133
 doctree/1 ... 131, 132, 133, 134, 135, 146, 147, 153,
 160, 162
 doctree_concat/3 131, 133
 doctree_insert_before_section/3 131, 132
 doctree_insert_end/3 131, 132
 doctree_is_empty/1 131, 132
 doctree_prepare_docst_translate_and_write/3
 131, 134
 doctree_putvars/5 131, 134
 doctree_restore/2 131, 134
 doctree_save/2 131, 133
 doctree_scan_and_save_refs/2 131, 134
 doctree_simplify/2 131, 134
 doctree_to_rawtext/3 131, 135
 doctree_translate_and_write/3 131, 135
 document structure 16
 documentation format 119
 documentation strings 32
 dvips 7, 115
 dvips/1 139, 141
 dynamic/1 18

E

Edison Mera 81, 111, 113
 emacs 11, 36, 39
 Emacs 15, 19, 20
 emacs Ciao mode 36, 37
 Emacs, accessing info files 19
 Emacs, generating manuals from 15

Emacs, LPdoc mode 15
 email address 29
 email addresses 11, 29
 emphasis face 27
 empty_doctree/1 131, 132
 encapsulated postscript 30
 engine/basic_props 103
 ensure_cache_dir/1 155, 156
 ensure_loaded/1 10, 22, 38
 ensure_output_dir/1 155, 156
 ensure_output_dir_prepared/2 120
 entry assertion 46
 entry/1 42, 46, 54
 enumerated list 26
 equiv/2 63, 64, 79
 errhandle 169
 error(a,b) 113
 error_free/1 63, 79
 error_text/3 163
 escape sequences 26
 escape_string/4 131, 135
 eval/1 .. 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
 75, 79, 107, 109
 example of lpdoc use 97
 example_module 103
 exception(error(a,b)) 113
 exception/1 81, 83
 exception/2 81, 83
 exceptions 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 exit assertion 46, 47
 exit/1 42, 46, 47
 exit/2 42, 47
 exported predicates 119
 exports 111

F

fails/1 81, 83, 85
 false assertion 49
 false/1 42, 49
 file_format_name/2 155, 156
 file_format_provided_by_backend/3 155, 156
 file_utils 81, 145, 147, 149, 151, 165
 filename/1 155
 filenames 120, 123, 137
 filetype/1 25, 32, 38

filter/2..... 63, 79
 finite_solutions/1..... 81, 83
 Firefox..... 19
 fixed format text..... 27
 fixed-width font..... 27
 flag_values/1..... 63, 79
 flt/1..... 63, 65
 fmt_html_template/3..... 151
 fmt_idx_env/7..... 159, 160
 fmt_index/3..... 159, 160
 fmt_infodir_entry/3..... 120, 121
 foo..... 103
 foo(Arg1)..... 109
 footnote..... 27
 format..... 120, 131, 145, 165, 169
 format_to_string..... 145, 147, 153
 formatting commands..... 3, 25, 41
 framed box..... 27
 Francisco Bueno..... 41, 57, 81, 95
 fsyntax... 25, 103, 120, 123, 131, 137, 139, 145, 147,
 149, 151, 153, 155, 167, 169
 func/1..... 54

G

g_assrt_body/1..... 45, 51, 55
 generate_html_pointer/5..... 7
 generate_html_pointer/6..... 7
 Generating..... 15
 generating from Emacs..... 15
 generating manuals..... 15
 German Puebla..... 41
 get_autodoc_opts/3..... 120, 121
 get_cache_dir/2..... 155, 156
 get_command_option/1..... 139, 140
 get_doc/4..... 123, 128
 get_doc_changes/3..... 123, 128
 get_doc_pred_varnames/2..... 123, 128
 get_first_loc_for_pred/3..... 123, 129
 get_idxbase/3..... 159
 get_idxsub/2..... 159
 get_mainmod/1..... 137
 get_mainmod_spec/1..... 137, 138
 get_mod_doc/3..... 123, 128
 get_output_dir/2..... 155, 156
 get_subbase/3..... 155, 157
 ghostview..... 8
 gnd/1..... 63, 67, 68, 104, 105, 106, 107, 109, 110

gndstr/1..... 63, 68
 GNU general public license..... 1
 ground/1... 67, 68, 70, 71, 72, 73, 74, 75, 81, 82, 85,
 105, 107, 109

H

hard side-effects..... 88
 have_choicpoints/1..... 81, 83
 head pattern..... 51, 52, 55
 head_pattern/1..... 48, 51, 52, 55
 hiord..... 81, 95, 103
 hiord_rt .. 25, 42, 51, 63, 81, 95, 103, 112, 114, 120,
 123, 131, 137, 139, 145, 147, 149, 151, 153, 155,
 159, 161, 163, 165, 167, 169
 hiordlib..... 81
 html..... 4
 HTML..... 1
 html index page..... 18

I

idx_get_indices/3..... 159, 160
 image file..... 30
 images, inserting..... 30
 images, scaling..... 30
 img_url/2..... 151
 include files..... 22, 38
 include/1..... 22, 38
 including a predicate definition..... 30
 including an image..... 30
 including code..... 30
 including files..... 30
 including images..... 30
 including or not authors..... 16
 including or not bug info..... 16
 including or not changelog..... 16
 including or not versions, patches..... 16
 indentation, avoiding..... 27
 indep/1..... 81, 82, 84, 85
 indep/2..... 81, 82, 84, 85
 index pages out of order..... 18
 index_comment/2..... 16, 120
 info..... 1, 4, 19, 20, 21, 24, 27
 info path list..... 20
 infodir_base/2..... 145
 inner..... 111
 insert_show_toc/3..... 131, 135

inserting images 11
 inst/2 63, 75
 installation 3
 INSTALLATION.lpdoc 23
 instance/1 81, 84
 instance/2 93
 instantiation properties 57
 int/1 63, 64, 87, 104, 105, 106, 108, 109
 integer/1 53
 internals 81
 internals manual 3, 21, 22
 introduction 34
 io_aux 25, 42, 51, 63, 81, 95, 103, 112, 114, 120,
 123, 131, 137, 139, 145, 147, 149, 151, 153, 155,
 159, 161, 163, 165, 167, 169
 io_basic .. 25, 42, 51, 63, 81, 95, 103, 112, 114, 120,
 123, 131, 137, 139, 145, 147, 149, 151, 153, 155,
 159, 161, 163, 165, 167, 169
 is/2 110
 is_det/1 64, 65, 66, 67, 68, 69, 70, 81, 84, 107
 is_index_cmd/1 159, 160
 is_nonempty_doctree/1 131, 132
 is_version/1 131, 135
 iso/1 10, 63, 75, 108
 italics face 27
 item in an itemized list 26
 itemized list 26

J

Jose F. Morales ... 119, 123, 131, 137, 139, 145, 147,
 149, 151, 153, 155, 159, 161, 165, 167, 169

K

keyboard key 27
 known_format/1 169

L

L=[[A], [p(A)]] 85
 L=[[A], [p(B)]] 85
 labgen_clean/1 123, 126
 labgen_get/2 123, 127
 labgen_init/1 123, 126
 LaTeX 26
 LaTeX notation 29
 letter size paper 17

library 3
 library(basicmodes) 52
 library(isomodes) 52
 linear/1 81, 84
 Linux 115
 list/1 63, 70, 71, 72, 75, 87, 104, 105, 106, 107,
 108, 109
 list/2 37, 38, 53, 63, 70, 105, 106, 109, 121, 169
 list_or_aorb/2 103, 104
 lists ... 42, 45, 81, 103, 120, 123, 131, 145, 147, 151,
 153, 159, 161, 165, 167
 literal 112
 literate programming 18
 load_vpaths/0 139, 140
 locate_and_convert_image/4 169
 log of changes 36
 long 112
 long/1 103, 105, 106
 lpdist/ciao_bundle_db 155
 lpdist/ciao_config_options 145
 lpdist/distutils 139
 lpdist/makedir_aux 155
 lpdoc .. 1, 3, 4, 5, 8, 15, 16, 17, 18, 19, 20, 21, 22, 23,
 24, 25, 29, 37, 41, 48, 52, 56, 97, 115
 lpdoc -help 16
 lpdoc all 16, 17, 19
 lpdoc_examples 97
 lpdoc_install 115
 lpdoc_option/1 139
 lpdocsrc(src(autodoc)) 149
 lpdocsrc(src(autodoc_aux)) ... 120, 123, 145, 153,
 165, 169
 lpdocsrc(src(autodoc_bibrefs)) 161
 lpdocsrc(src(autodoc_doctree)) ... 120, 123, 145,
 147, 153, 159, 161, 165
 lpdocsrc(src(autodoc_filesystem)) 120, 123,
 131, 145, 147, 149, 159, 161, 169
 lpdocsrc(src(autodoc_html)) 131
 lpdocsrc(src(autodoc_html_resources)) 120,
 147
 lpdocsrc(src(autodoc_html_template)) 147
 lpdocsrc(src(autodoc_images)) 145, 147, 153
 lpdocsrc(src(autodoc_index)) 120, 123, 131,
 145, 147
 lpdocsrc(src(autodoc_man)) 131
 lpdocsrc(src(autodoc_parse)) 120, 123, 165
 lpdocsrc(src(autodoc_refsdb)) 120, 123, 131,
 147, 159, 165

lpdocsrc(src(autodoc_settings)) .. 120, 123, 131,
137, 145, 147, 149, 151, 155, 165, 167, 169
lpdocsrc(src(autodoc_state))..... 120, 131, 145,
147, 153, 155, 159, 161, 165, 169
lpdocsrc(src(autodoc_structure))..... 120, 123,
131, 145, 147, 155, 159, 161
lpdocsrc(src(autodoc_texinfo))..... 120, 131
lpdocsrc(src(comments)) .. 120, 123, 131, 145, 147,
153
lpdocsrc(src(pbundle_download)) 147

M

machine readable comments..... 25, 32
main body 34
main file 4
main/0 3, 4
main/1 3, 4
main_absfile_for_subtarget/3..... 155, 157
main_absfile_in_format/2..... 155, 156
main_output_name/2..... 155, 157
make/make_rt 120, 123, 131, 137, 139, 145, 151,
165, 169
Makefile 18
makeinfo 115
makeinfo/1 139, 141
makertf/1 139, 141
man 19, 20
Manuel Hermenegildo .. 15, 25, 41, 51, 57, 63, 81, 97,
115, 119, 123, 131, 145, 153, 163, 165, 167
master index 4
member/2 63, 71, 76
memo/1 63, 79
messages 120, 123, 145, 149, 151, 167, 169
meta_predicate/1 18
meta_props 3, 95
mode 42, 52
modedef/1 42, 47, 52
modtype/1 123, 129
module comment 34
module declaration 119
module/1 119
module/2 119
mshare/1 81, 85
mut_exclusive/1 81, 85
mytype/1 103, 106

N

n_assrt_body/5 54, 55
nabody/1 51, 54
native/1 ... 63, 64, 65, 66, 67, 68, 77, 82, 83, 85, 87,
92, 93
native/2 63, 77, 82, 84, 85, 86
native_props 3, 81
nativeprops 63, 103
nativeprops.pl 111
netscape 8
new item in description list 27
nlist/1 71
nlist/2 63, 71
nnegint/1 63, 64, 65
no 111, 112
no_choicepoints/1 81, 85
no_exception/1 81, 86
no_exception/2 81, 86
no_rtcheck/1 .. 63, 76, 77, 78, 82, 83, 84, 85, 88, 89,
90, 91, 92
no_signal/1 81, 86
no_signal/2 81, 86
non_det/1 81, 86
nonground/1 81, 86
nonpure ... 25, 42, 51, 63, 81, 95, 103, 112, 114, 120,
123, 131, 137, 139, 145, 147, 149, 151, 153, 155,
159, 161, 163, 165, 167, 169
nonvar/1 64, 65, 66, 67, 68, 69, 70, 72, 73, 105,
106, 132, 134
normalize_index_cmd/3 159, 160
nortchecks 63
not_covered/1 81, 86
not_fails 113
not_fails/1 81, 85, 87, 105, 106, 107, 108, 109
not_further_inst/1 54
not_further_inst/2 63, 76, 105, 106
not_mut_exclusive/1 81, 87
num/1 63, 66, 110
num_code/1 63, 74
num_solutions/2 81, 87

O

odd 81
og/2 110
one-sided printing 16
op/3 18

operator_specifier/1 63, 69, 70
 operators 131
 option_comment/2 16, 123, 124

P

p/1 103, 105
 p/3 103, 108
 p/5 103, 105
 packages 22, 38
 page numbering, changing 16
 page size, changing 17
 page style, changing 17
 paragraph break 27
 parametric property 96
 parametric regular type abstractions 96
 parametric type functor 60
 parse_commands/3 165
 parse_structure/0 137
 parts in a large document 22
 parts in large documents 38
 pdf generation 7
 pdf viewer 8
 pdftex 115
 pe_type/1 63, 79
 Pedro Lopez 57, 81
 pillow/html 151
 planned improvement 36
 possibly_fails/1 81, 87
 possibly_nondet/1 81, 88
 pred assertion 42, 43
 pred/1 29, 42, 43, 44, 45, 47, 51, 54
 pred/2 42, 43
 pred_has_docprop/2 123, 128
 predfunctor/1 51, 56
 predicate 112
 predicate/n 113
 predname/1 36, 37, 38, 52, 63, 74
 prelude 25, 42, 51, 60, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 prepare_current_refs/1 161, 162
 prepare_mathjax/0 149
 prepare_web_skel/1 149
 program assertions 41
 program section 35
 program subsection 35
 program subsection 35

Prolog 3, 16, 22
 Prolog source files 3
 Prolog, Ciao 15
 prolog_flags ... 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 prop assertion 45, 46
 prop/1 42, 45, 46
 prop/2 42, 46, 95, 96
 prop_abs/1 96
 properties 3
 properties of computations 57
 properties of execution states 57
 properties, basic 63
 properties, native 81
 property 45
 property abstraction 96
 property compatibility 75
 property_conjunction/1 48, 49, 51, 53
 property_starterm/1 51, 53
 propfunctor/1 51, 56
 providing information to the compiler 46, 48
 ps2pdf/1 139, 141
 pure 60

Q

q/1 103, 107
 q/2 103, 104

R

r/1 103, 104
 read 120, 123, 131
 read_file/2 167
 references 29, 115
 regtype assertion 60, 61
 regtype/1 60, 61, 63, 77
 regtype/2 60, 61, 95, 96
 regtypes .. 3, 25, 51, 57, 103, 114, 120, 123, 131, 137,
 139, 145, 147, 153, 155, 159, 161, 163, 165, 167,
 169
 regular type 60
 regular type definitions 57
 regular type expression 61
 regular types 57
 relations/2 70, 81, 88
 requested_file_formats/1 139, 140

reset_output_dir_db/0..... 120
 resolve_bibliography/1..... 165
 rtc_status/1..... 78
 rtcheck/1..... 63, 77
 rtcheck/2..... 63, 78, 82, 85, 86, 87, 88, 89, 93
 rtchecks..... 111
 rtchecks/rtchecks_send..... 81
 rtchecks_abort_on_error..... 112
 rtchecks_asrloc..... 111
 rtchecks_calloc..... 112
 rtchecks_entry..... 111
 rtchecks_exit..... 111
 rtchecks_inline..... 111
 rtchecks_level..... 111
 rtchecks_namefmt..... 112
 rtchecks_predloc..... 111
 rtchecks_rt.pl..... 111
 rtchecks_test..... 111
 rtchecks_trust..... 111
 rftohlp/1..... 139, 141
 run-time checks..... 45
 running unit tests..... 113

S

s/1..... 103, 106
 s_assrt_body/1..... 44, 45, 46, 47, 51, 54
 scribe..... 26
 section..... 27
 section_prop/2..... 131, 133
 section_select_prop/3..... 131, 133
 sections..... 29
 secttree/1..... 161, 162
 secttree_resolve/3..... 161, 162
 sequence/2..... 63, 72
 sequence_or_list/2..... 63, 72
 setting_value/2..... 139, 140
 setting_value_or_default/2..... 139
 setting_value_or_default/3..... 139, 140
 SETTINGS..... 5, 8, 9, 29
 SETTINGS.pl..... 15, 16, 18, 19, 20, 21, 22, 23
 SETTINGS.pl.generated..... 15
 sh_exec/2..... 167
 sharing pieces of text..... 30
 sharing sets..... 85
 short..... 112
 sideff/2... 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
 74, 75, 76, 77, 78, 107, 109

sideff_hard/1..... 81, 88
 sideff_pure/1..... 81, 88
 sideff_soft/1..... 81, 89
 signal/1..... 81, 89
 signal/2..... 81, 89
 signals/2..... 81, 89
 size/2..... 81, 89
 size/3..... 81, 90
 size_lb/2..... 81, 90
 size_metric/3..... 81, 90
 size_metric/4..... 81, 91
 size_o/2..... 81, 90
 size_ub/2..... 81, 90
 soft side-effects..... 89
 solutions/2..... 81, 87
 sort..... 81
 space, extra lines..... 27
 spcae, horizontal fill..... 27
 special characters..... 31
 specifications..... 41
 standalone_docstr/1..... 137
 steps/2..... 81, 91
 steps_lb/2..... 81, 91, 105
 steps_o/2..... 81, 91
 steps_ub/2..... 81, 92
 streams..... 81
 streams_basic.. 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 string/1.... 63, 73, 74, 120, 124, 135, 151, 160, 162,
 169
 stringcommand/1..... 25, 26, 48, 52, 54, 55, 56
 strings..... 25
 strong face..... 27
 struct/1..... 63, 67
 subsection..... 27
 subsubsection..... 27
 subtarget/1..... 155, 156
 subtitle..... 33
 succeeds/1..... 81, 91
 success assertion..... 44
 success/1..... 42, 44, 46
 success/2..... 42, 44
 supported documentation formats..... 121
 supported_file_format/1..... 155, 156
 supported_option/1..... 123, 124
 synopsis section of the man page..... 16
 syntax of formatting commands..... 26

system 81, 120, 123, 145, 147, 151, 155, 167, 169
 system modules 16
 system_extra 120, 123, 131, 139, 145, 149, 151,
 155, 165, 167, 169
 system_info 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169

T

t/5 103, 106
 tau/1 81, 92
 term/1 33, 63, 71, 121
 term_basic 25, 42, 51, 60, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 term_compare 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 term_typing 25, 42, 51, 63, 81, 95, 103, 112, 114,
 120, 123, 131, 137, 139, 145, 147, 149, 151, 153,
 155, 159, 161, 163, 165, 167, 169
 terminates/1 81, 92
 terms 120, 123, 131, 137, 145, 149, 151, 155, 165,
 169
 terms_check 63, 81
 terms_vars 81
 test assertion 44, 45, 113
 test/1 42, 44, 45
 test/2 42, 44
 test_type/2 64, 65, 66, 67, 68, 81, 92
 tex 24, 115
 TeX 9
 tex/1 139, 140
 texec assertion 43
 texec/1 42, 43
 texec/2 42, 43
 texi2dvi 115
 texindex 115
 texindex/1 139, 141
 texinfo 1, 3, 4, 16, 17, 21, 24, 119
 Texinfo 115
 texinfo files 4
 textual comments 25
 thesis-like style 17
 throws/2 81, 93
 time_struct/1 39
 times(N) 113

title 32, 33
 top_suffix/2 123, 125
 tree_of/2 103, 104
 troubleshooting 15
 true assertion 49
 true/1 42, 49
 trust assertion 48
 trust/1 42, 48
 try_sols(N) 113
 ttyout 120
 two-sided 16
 typeindex/5 159
 types 3
 typewriter-like font 27

U

u/3 103, 105
 unit tests 113
 unittest 113
 Universal Resource Locator 29
 Unix 20
 URL 29
 url references 11
 urls 29
 usage 42
 usage of a command 30
 usage of the application 16
 usage section 11
 usage tips 15
 usage_message/1 11, 16
 use_module/1 22
 use_package/1 22, 38
 user_output/2 81, 93
 using citations 16
 using_mathjax/1 149

V

var/1 53, 104, 105, 106, 108, 109, 132
 variable names 41
 verbatim text 27
 verify_settings/0 139
 version 29
 version maintenance mode for packages 37
 version number 36
 version_date/2 131, 135
 version_descriptor/1 25, 32, 36

version_maintenance_type/1..... 36, 37, 39
 version_number/1 38
 version_numstr/2..... 131, 135
 version_patch/2..... 131, 135
 version_string/2..... 131, 135
 viewer/4 139, 140

W

w/1..... 103, 106
 word-help..... 20
 word-help.el..... 20
 write..... 123, 131

WWW 1, 19
 WWW address 29

X

xdvi 17
 xdvi/1..... 139, 140
 xdvisize/1 139, 140

Y

yes 111, 112
 ymd_date/1..... 39