

IF/Prolog V5.3

Quick Reference

Is there

anything you would like to tell us about this manual?  
Please send us your comments.

Siemens AG Austria  
PSE KB B3  
Gudrunstrasse 11  
A-1100 Vienna  
Austria  
Fax.: +43-1-1707 56992  
email: [prolog@siemens.at](mailto:prolog@siemens.at)

The information in this document is subject to change and does not represent a commitment on the part of Siemens AG Austria. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open and the X device are trademarks of X/Open Company Ltd.

Copyright ©Siemens AG Austria, 1999. All rights reserved.

The reproduction, transmission, translation or exploitation of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. Delivery subject to availability; right of technical modifications reserved.

# Contents

Contents	iii
Prolog command line parameters	1
Environment variables	3
Built-in predicates	4
Arithmetic functions	19
Prolog flags	21
Input/Output	25
Syntax messages	27
Signals	27
Debugger commands	28
Debugger parameters	33
Tracer parameters	35
C Interface	36
Bibliography	43

## Prolog command line parameters

- `-root pathname`  
Specify *pathname* under which the files of the Prolog system are installed.
- `-r file` Load system status from *file* (see `save_system/1`).
- `-st file` Specify which startup file, if any, should be loaded or consulted.
- `-c file` Consult *file* (see `consult/1`).
- `-l file` Load *file* (see `load/1`).

The system parameters `-c` and `-l` are processed in the order of their appearance in the command line. The system parameters `-c` and `-l` are processed after the parameters `-r` and `-st`.

Independently of their position in the command line, the following system parameters are evaluated **before** the parameters `-r`, `-st`, `-c` and `-l`.

- `-sp search_path`  
Set the initial value of the Prolog `search_path` flag to the list specified in *search\_path*.
- `-msg number [k|K]`  
Set the initial size of the global stack to *number* bytes.
- `-msc number [k|K]`  
Set the initial size of the constraint data stack to *number* bytes.
- `-mst number [k|K]`  
Set the initial size of the variable instantiations stack (trail) to *number* bytes.
- `-msl number [k|K]`  
Set the initial size of the local data stack to *number* bytes.
- `-msx number [k|K]`  
Set the initial size of the extended terms stack to *number* bytes.
- `-ms number [k|K]`  
Set the initial size of the total memory of the Prolog system to *number* bytes.
- `-mi number [k|K]`  
Set the minimum size of the memory blocks requested by the Prolog system to *number* bytes.
- `-debug` The initial value of the Prolog flag `debug` is set to `on`.
- `-iso` Only the features required by the ISO standard are available.
- `-nobuf` The standard input streams are not buffered by Prolog.
- `-nocopyright`  
The copyright message is not output when Prolog starts.
- `-nonotify`  
The initial value of the Prolog flag `consult` is set to `nonotify`.

- nosignal** Prevents the Prolog system from managing signals itself, i.e. no signal handling routines are defined.
- notty** The standard streams **screen** and **keyboard** are not linked to the terminal, but are mapped to the standard input and standard output.
- nowarnings** The initial value of the Prolog flag **warnings** is set to **off**.
- prompt** The initial value of the Prolog flag **prompt** is set to **on**.
- stream** Prolog acts as if no terminal is available.
- verbose** A message containing configuration and license information and creation date of IF/Prolog is output when Prolog starts.
- xenv** If a OSF/Motif environment (X server) is available, the Prolog system starts the graphical development environment under OSF/Motif.
- Identifies the end of the system arguments.

## Environment variables

The Prolog system uses the following environment variables:

Variable	Used for
TMPDIR	directory for temporary files
LANG	national character set supported by the Prolog-scanner
TERM	type of the terminal
HOME	name of HOME-Directory of the user
SHELL	name of shell (see <code>system/0</code> )
EDITOR	name of standard editor (see <code>edit/0/1</code> )
PROROOT	path where Prolog is installed
PRORC	name of startup file
PROHELP	name of files with help-information (see <code>help/0/1</code> )
PROPATH	search path for files used by the Prolog-system

## Built-in predicates

Predicates which allow **backtracking** are identified by the character '#' before the functor.

**Metapredicates** are identified by the character '@' before the functor.

### Term classification

<b>array</b> (@TestTerm)	Test for array
<b>array</b> (@Array, ?Dimension)	Query dimension of an array
<b>atom</b> (@TestTerm)	Test for atom
<b>atomic</b> (@TestTerm)	Test for constant
<b>compound</b> (@TestTerm)	Test for structure
<b>cyclic</b> (@TestTerm)	Test for cyclic term
<b>digit</b> (@TestTerm)	Test for digit
<b>float</b> (@TestTerm)	Test for floating-point number
<b>ground</b> (@TestTerm)	Test for ground instantiation
<b>integer</b> (@TestTerm)	Test for integer
<b>letter</b> (@TestTerm)	Test for letter
<b>nonvar</b> (@TestTerm)	Test for instantiation
<b>number</b> (@TestTerm)	Test for number
<b>rational</b> (@TestTerm)	Test for rational number
<b>var</b> (@TestTerm)	Test for variable

### Term comparison

?Term1 = ?Term2	Unify terms
@Term1 == @Term2	Compare terms for identity
@Term1 \== @Term2	Compare terms for unidentity
@Term1 @< @Term2	Term1 less than Term2 (order of terms)
@Term1 @> @Term2	Term1 greater than Term2 (order of terms)
@Term1 @=< @Term2	Term1 less or equal Term2 (order of terms)
@Term1 @>= @Term2	Term1 greater or equal Term2 (order of terms)
@Term1 @= @Term2	Term1 equal Term2 (order of terms)
@Term1 @\= @Term2	Term1 not equal Term2 (order of terms)
@Term1 \= @Term2	Test for non-unifiability
<b>compare</b> (?Op, @Term1, @Term2)	Compare terms
<b>unify_with_occurs_check</b> (?Term1, ?Term2)	Unify terms with occurs check

**Term conversion**

<code>+Structure =.. ?List</code>	Convert a structure into a list
<code>-Structure =.. +List</code>	Convert a list into a structure
<code>arg(+Position, +Structure, ?Argument)</code>	Access individual arguments of a structure
<code>atom_chars(+Atom, ?List)</code>	Decompose an atom into a list of characters
<code>atom_chars(-Atom, @List)</code>	Convert a list of characters into an atom
<code>atom_codes(+Atom, ?List)</code>	Decompose an atom into a list of character codes
<code>atom_codes(-Atom, @List)</code>	Convert a list of character codes into an atom
<code>atom_number(+Atom, ?Number)</code>	Convert an atom into a number
<code>atom_number(-Atom, @Number)</code>	Convert a number into an atom
<code>char_code(+Character, ?CharCode)</code>	Convert a character into a character code
<code>char_code(-Character, +CharCode)</code>	Convert a character code into a character
<code>copy_term(@Term1, ?Term2)</code>	Create a term copy
<code>create_array(?Array, @Dimension)</code>	Create an array
<code>functor(-Structure, +Functor, +Arity)</code>	Generate a structure
<code>functor(@Structure, ?Functor, ?Arity)</code>	Analyze a structure
<code>get_array(@Array, @Index, ?Term)</code>	Query array element
<code>number_chars(+Number, ?List)</code>	Decompose a number into a list of characters
<code>number_chars(-Number, @List)</code>	Convert a list of characters into a number
<code>number_codes(+Number, ?List)</code>	Decompose a number into a list of character codes
<code>number_codes(-Number, @List)</code>	Convert a list of character codes into a number
<code>parse_atom(+String, +StartPosition, ?EndPosition, ?Term, ?VarList, ?Error)</code>	Parse a character string in accordance with Prolog syntax
<code>rational(+Number, ?Numerator, ?Denominator)</code>	Decompose a rational number
<code>reduce(+Level, @Term, ?ReducedTerm)</code>	Reduce the depth of a structure
<code>set_array(@Array, @Index, @Term)</code>	Set array element
<code>write_atom(@Term, ?Atom)</code>	Convert a term into an atom
<code>write_formatted_atom(?Atom, +Format, @TermList)</code>	Formatted output of terms
<code>writeln_atom(@Term, ?Atom)</code>	Convert a term into an atom

**String processing**

<code>atom_chars(+Atom, ?List)</code>	Decompose an atom into a list of characters
<code>atom_chars(-Atom, @List)</code>	Convert a list of characters into an atom
<code>atom_codes(+Atom, ?List)</code>	Decompose an atom into a list of character codes
<code>atom_codes(-Atom, @List)</code>	Convert a list of character codes into an atom
<code>#atom_concat(?Atom1, ?Atom2, +Atom3)</code>	Decompose an atom
<code>#atom_concat(+Atom1, +Atom2, -Atom3)</code>	Compose an atom

<b>atom_length</b> (+Atom, ?Length)	Count the characters in an atom
<b>atom_number</b> (+Atom, ?Number)	Convert an atom into a number
<b>atom_number</b> (-Atom, @Number)	Convert a number into an atom
<b>atom_part</b> (+Atom, +Position, +Length, ?SubAtom)	Determine the subatom of an atom
<b>atom_prefix</b> (+Atom, +Length, ?Prefix)	Determine the prefix of an atom
<b>atom_split</b> (+Atom, +Delimiter, ?Subatoms)	Decompose an atom
<b>atom_suffix</b> (+Atom, +Length, ?Suffix)	Determine the suffix of an atom
<b>byte_length</b> (+Atom, ?Length)	Count the bytes in an atom
<b>concat_atom</b> (@List, ?Atom)	Concatenate individual atoms to form an atom
<b>concat_atom</b> (@List, +Delimiter, ?Atom)	Concatenate individual atoms to form an atom with separators
<b>current_language</b> (?Language)	Query current language
<b>getchar</b> (+Atom, +Position, ?Character)	Access a character in an atom
<b>index</b> (+Atom, +String, ?Position)	Determine the position of a character string in an atom
<b>lower_upper</b> (+Lowercase, ?Uppercase)	Convert lowercase letters into uppercase
<b>lower_upper</b> (-Lowercase, +Uppercase)	Convert uppercase letters into lowercase
<b>match</b> (+Mask, +Atom)	Pattern matching
<b>match</b> (+Mask, +Atom, ?Replacements)	Pattern matching
<b>match_atom</b> (@List, ?Atom)	Concatenate atoms to form an atom
<b>#match_atom</b> (+List, +Atom)	Concatenate matching atoms to form an atom
<b>parse_atom</b> (+String, +StartPosition, ?EndPosition, ?Term, ?VarList, ?Error)	Parse a character string in accordance with Prolog syntax
<b>regexp</b> (+RegExp, +Atom)	Regular expression matching
<b>regexp</b> (+RegExp, +Atom, ?List)	Regular expression matching
<b>#sub_atom</b> (+Atom, ?StartLength, ?Length, ?RestLength, ?SubAtom)	Analyze an atom
<b>write_atom</b> (@Term, ?Atom)	Convert a term into an atom
<b>write_formatted_atom</b> (?Atom, +Format, @TermList)	Formatted output of terms
<b>writeq_atom</b> (@Term, ?Atom)	Convert a term into an atom

## List processing

<b>#append</b> (?Head, ?Tail, ?List)	Append or decompose lists
<b>atom_chars</b> (+Atom, ?List)	Decompose an atom into a list of characters
<b>atom_chars</b> (-Atom, @List)	Convert a list of characters into an atom
<b>atom_codes</b> (+Atom, ?List)	Decompose an atom into a list of character codes
<b>atom_codes</b> (-Atom, @List)	Convert a list of character codes into an atom
<b>concat_atom</b> (@List, ?Atom)	Concatenate individual atoms to form an atom

<b>concat_atom</b> (@List, +Delimiter, ?Atom)	Concatenate individual atoms to form an atom with separators
<b>connect</b> (?List, ?Head, ?Tail)	Connect a head and a tail to form a list
<b>list_last</b> (?Element, +List)	Last element in list
<b>list_length</b> (+List, ?Length)	Count the elements in a list
<b>list_nth</b> (?Position, +List, ?Element)	Access elements of a list
<b>match_atom</b> (@List, ?Atom)	Concatenate atoms to form an atom
<b>#match_atom</b> (+List, +Atom)	Concatenate matching atoms to form an atom
<b>#member</b> (?Element, ?List)	List membership
<b>memberchk</b> (?Element, ?List)	List membership
<b>nonmember</b> (@Element, @List)	List membership
<b>number_chars</b> (+Number, ?List)	Decompose a number into a list of characters
<b>number_chars</b> (-Number, @List)	Convert a list of characters into a number
<b>number_codes</b> (+Number, ?List)	Decompose a number into a list of character codes
<b>number_codes</b> (-Number, @List)	Convert a list of character codes into a number
<b>remove</b> (+Sublist, +List, ?Remainder)	Remove a sublist from a list
<b>reverse</b> (@List, ?ReversedList)	Reverse a list
<b>sort</b> (@List, ?SortedList)	Sort a list

## Arithmetic

@Value < @Value	Value less than Value
@Value > @Value	Value greater than Value
@Value ==< @Value	Value less or equal Value
@Value >= @Value	Value greater or equal Value
@Value == @Value	Value equal Value
@Value =\= @Value	Value not equal Value
<b>atom_number</b> (-Atom, @Number)	Convert a number into an atom
<b>for</b> (+Start, +Counter, +End)	Check range
<b>#for</b> (+Start, ?Counter, +End)	Generate a sequence of integers
?Result <b>is</b> @Expression	Evaluate arithmetic expressions

## Global variables

@ <b>#current_global</b> (?Name)	Query global variables
@ <b>get_global</b> (+Name, ?Value)	Query the value of a global variable
@ <b>pop_global</b> (+Name, ?Value)	Query global variable and remove value
@ <b>push_global</b> (+Name, @Value)	Set value of a global variable
@ <b>set_global</b> (+Name, @Value)	Set value of a global variable
@ <b>unset_global</b> (+Name)	Delete a global variable

## Stream processing

<b>assign_alias</b> (+Alias, @Stream)	Define an alias for an input/output stream
<b>cancel_alias</b> (+Alias)	Cancel an alias for an input/output stream
<b>close</b> (@Stream)	Close an input/output stream
<b>close</b> (@Stream, @Actions)	Close an input/output stream
<b>#current_alias</b> (?Alias, ?Stream)	Query aliases of input/output streams
<b>#current_device_control</b> (+Device, ?Command, ?Arity)	Query information on device drivers
<b>current_error</b> (?Stream)	Query current error output stream
<b>current_input</b> (?Stream)	Query current input stream
<b>current_output</b> (?Stream)	Query current output stream
<b>#current_stream_control</b> (@Stream, ?Command, ?Arity)	Query information on device drivers
<b>device_control</b> (+Device, @Command)	Control an input/output device
<b>@edit</b>	Edit a file
<b>@edit</b> (+Filename)	Edit a file
<b>error_tell</b> (@Stream)	Set current error output stream
<b>error_telling</b> (?Stream)	Query current error output stream
<b>error_told</b>	Reset current error output stream
<b>file_test</b> (+Pathname, +AccessMode)	Check access permission for a file
<b>filepos</b> (@Stream, ?Line)	Position within an input stream
<b>filepos</b> (@Stream, ?Line, ?Column)	Position within an input stream
<b>flush_output</b>	Flush the buffer of an output stream
<b>flush_output</b> (@Stream)	Flush the buffer of an output stream
<b>open</b> (@DeviceAndName, +Mode, -Stream)	Open an input/output stream
<b>open</b> (@DeviceAndName, +Mode, -Stream, @Options)	Open an input/output stream
<b>reset_streams</b>	Restore default values for current streams
<b>see</b> (@Stream)	Set current input stream
<b>seeing</b> (?Stream)	Query current input stream
<b>seen</b>	Reset current input stream
<b>set_error</b> (@Stream)	Set current error output stream
<b>set_input</b> (@Stream)	Set current input stream
<b>set_output</b> (@Stream)	Set current output stream
<b>set_stream_position</b> (@Stream, @Position)	Position within an input/output stream
<b>stream_control</b> (@Stream, @Command)	Control an input/output stream
<b>stream_copy</b> (@Stream1, @Stream2)	Redefine streams
<b>stream_device</b> (@Stream, ?Device)	Determine device name of an input/output stream
<b>#stream_property</b> (?Stream, ?Info)	Query information on input/output streams

---

<b>stream_type</b> (@Stream, ?StreamType)	Query type of an input/output stream
<b>tell</b> (@Stream)	Set current output stream
<b>telling</b> (?Stream)	Query current output stream
<b>told</b>	Reset current output stream
<b>unix_make_pipe</b> (-Pipename)	Create a pipe

## Operators

<b>#current_op</b> (?Priority, ?Assoc, ?Name)	Information on operators
<b>op</b> (+Priority, +Assoc, @Names)	Define and delete operators
<b>:- op</b> (+Priority, +Assoc, @Names)	Define and delete operators (directive)

## Elementary input/output

<b>at_end_of_line</b>	Query end of line
<b>at_end_of_line</b> (@Stream)	Query end of line
<b>at_end_of_stream</b>	Query end of stream
<b>at_end_of_stream</b> (@Stream)	Query end of stream
<b>get_byte</b> (?ByteCode)	Input a byte
<b>get_byte</b> (@Stream, ?ByteCode)	Input a byte
<b>get_char</b> (?Character)	Input a character
<b>get_char</b> (@Stream, ?Character)	Input a character
<b>get_code</b> (?CharCode)	Input a character
<b>get_code</b> (@Stream, ?CharCode)	Input a character
<b>get_until</b> (+SearchChar, ?Text, ?EndChar)	Read up to a specific character
<b>get_until</b> (@Stream, +SearchChar, ?Text, ?EndChar)	Read up to a specific character
<b>nl</b>	Output newline
<b>nl</b> (@Stream)	Output newline
<b>outtab</b> (+Position)	Position the cursor
<b>outtab</b> (@Stream, +Position)	Position the cursor
<b>peek_byte</b> (?ByteCode)	Input a byte
<b>peek_byte</b> (@Stream, ?ByteCode)	Input a byte
<b>peek_char</b> (?Character)	Input a character
<b>peek_char</b> (@Stream, ?Character)	Input a character
<b>peek_code</b> (?CharCode)	Input a character
<b>peek_code</b> (@Stream, ?CharCode)	Input a character
<b>put_byte</b> (+ByteCode)	Output a byte
<b>put_byte</b> (@Stream, +ByteCode)	Output a byte
<b>put_char</b> (+Char)	Output a character
<b>put_char</b> (@Stream, +Char)	Output a character

<b>put_code</b> (+CharCode)	Output a character
<b>put_code</b> (@Stream, +CharCode)	Output a character
<b>skip_line</b>	Skip an input line
<b>skip_line</b> (@Stream)	Skip an input line
<b>tab</b> (+Number)	Move the cursor forward
<b>tab</b> (@Stream, +Number)	Move the cursor forward

### Input/output for terms

<b>char_conversion</b> (+CharacterIn, +CharacterOut)	Define a character conversion
<b>:- char_conversion</b> (+CharacterIn, +CharacterOut)	Define a character conversion (directive)
<b>#current_char_conversion</b> (?CharacterIn, ?CharacterOut)	Information on character conversions
<b>#current_op</b> (?Priority, ?Assoc, ?Name)	Information on operators
<b>float_format</b> (?FormatOld, @FormatNew)	Query and set floating-point format
<b>op</b> (+Priority, +Assoc, @Names)	Define and delete operators
<b>:- op</b> (+Priority, +Assoc, @Names)	Define and delete operators (directive)
<b>portray</b> (@Stream, @Term)	User-defined output predicate
<b>@print</b> (@Term)	Output a term
<b>@print</b> (@Stream, @Term)	Output a term
<b>read</b> (?Term)	Input a term
<b>read</b> (@Stream, ?Term)	Input a term
<b>read_error</b> (?Line, ?Number)	Query position of a syntax error
<b>read_error</b> (?Line, ?Column, ?Number)	Query position of a syntax error
<b>read_term</b> (?Term, +Options)	Input a term
<b>read_term</b> (@Stream, ?Term, +Options)	Input a term
<b>syntax_error</b> (+Number, ?Message)	Assignment of syntax error number to error message
<b>write</b> (@Term)	Output a term
<b>write</b> (@Stream, @Term)	Output a term
<b>write_canonical</b> (@Term)	Output a term
<b>write_canonical</b> (@Stream, @Term)	Output a term
<b>write_formatted</b> (+Format, @TermList)	Formatted output of terms
<b>write_formatted</b> (@Stream, +Format, @TermList)	Formatted output of terms
<b>@write_term</b> (@Term, @Options)	Output a term
<b>@write_term</b> (@Stream, @Term, @Options)	Output a term
<b>writeq</b> (@Term)	Output a term in readable form
<b>writeq</b> (@Stream, @Term)	Output a term in readable form

## Screen control

<b>clear_eol</b>	Clear to end of line
<b>clear_eol</b> (@Stream)	Clear to end of line
<b>clear_eos</b>	Clear to end of screen
<b>clear_eos</b> (@Stream)	Clear to end of screen
<b>clear_screen</b>	Clear screen
<b>clear_screen</b> (@Stream)	Clear screen
<b>get_last</b> (?Line, ?Column)	Determine the cursor position
<b>get_last</b> (@Stream, ?Line, ?Column)	Determine the cursor position
<b>move_cursor</b> (+Line, +Column)	Position the cursor
<b>move_cursor</b> (@Stream, +Line, +Column)	Position the cursor
<b>tty_size</b> (?Lines, ?Columns)	Query the screen size
<b>tty_size</b> (@Stream, ?Lines, ?Columns)	Query the screen size

## Directives

<b>:- begin_module</b> (+Module)	Define the body of a module
<b>:- char_conversion</b> (+CharacterIn, +CharacterOut)	Define a character conversion
<b>:- discontinuous</b> (@PredicateList)	Declare discontinuous predicate definition
<b>:- dynamic</b> (@PredicateList)	Declare predicates as modifiable
<b>:- end_module</b>	Conclude the definition of a module
<b>:- end_module</b> (+Module)	Conclude the definition of a module
<b>:- ensure_loaded</b> (+Filename)	Insert a Prolog file in the source text once
<b>:- export</b> (@PredicateList)	Export predicates
<b>:- import</b> (+Module)	Import predicates
<b>:- import</b> (+Module, @PredicateList)	Import predicates selectively
<b>:- include</b> (+Filename)	Include a Prolog file in the source text
<b>:- init</b> (+Goal)	Initialization goal
<b>:- initialization</b> (+Goal)	Initialization goal
<b>:- meta</b> (@PredicateList)	Declare metapredicates
<b>:- module</b> (+Module)	Define a module
<b>:- multifile</b> (@PredicateList)	Declare predicates as being scattered over several files
<b>:- nonotify</b>	Do not output load message
<b>:- op</b> (+Priority, +Assoc, @Names)	Define and delete operators
<b>:- private</b> (@PredicateList)	Declare predicates as private
<b>:- reexport</b> (+Module)	Reexport predicates
<b>:- reexport</b> (+Module, @PredicateList)	Reexport predicates
<b>:- set_prolog_flag</b> (+Flag, +Value)	Set value of a Prolog flag

## Modules

<b>+Module</b> : +Predicate	Specify the module of the called predicate
<b>+Predicate @ +Module</b>	Specify module context for predicate call
<b>:- begin_module(+Module)</b>	Define the body of a module
<b>calling_context(?Module)</b>	Query the module argument of metapredicates
<b>compile(+Filename)</b>	Compile a Prolog text
<b>current_default_module(?Module)</b>	Query current module
<b>#current_module(?Module)</b>	Query modules
<b>:- end_module</b>	Conclude the definition of a module
<b>:- end_module(+Module)</b>	Conclude the definition of a module
<b>:- export(@PredicateList)</b>	Export predicates
<b>:- import(+Module)</b>	Import predicates
<b>:- import(+Module, @PredicateList)</b>	Import predicates selectively
<b>load(+Filename)</b>	Load compiled Prolog text
<b>:- meta(@PredicateList)</b>	Declare metapredicates
<b>:- module(+Module)</b>	Define a module
<b>:- private(@PredicateList)</b>	Declare predicates as private
<b>:- reexport(+Module)</b>	Reexport predicates
<b>:- reexport(+Module, @PredicateList)</b>	Reexport predicates
<b>@save_module(+Filename)</b>	Store current module in a file
<b>set_default_module(+Module)</b>	Set current module
<b>unload(+Module)</b>	Remove module from the database

## Grammar rules

<b>+Arg1 --&gt; +Arg2</b>	Create a grammar rule
<b>expand_term(+TermIn, ?TermOut)</b>	Transform a grammar rule
<b>#phrase(+Goal, ?List)</b>	Call a grammar rule

## Database manipulation

<b>@[ +Filename, ...]</b>	List notation for consult/1
<b>@[ - +Filename, ...]</b>	List notation for reconsult/1
<b>@abolish(@Predicate)</b>	Remove a predicate from the database
<b>@asserta(@Clause)</b>	Insert a clause in the database
<b>@asserta(@Head, @Body)</b>	Insert a clause in the database
<b>@asserta_with_names(@Clause, @VarList)</b>	Insert a clause in the database
<b>@asserta_with_names(@Head, @Body, @VarList)</b>	Insert a clause in the database
<b>@assertz(@Clause)</b>	Insert a clause in the database

---

<b>@assertz</b> (@Head, @Body)	Insert a clause in the database
<b>@assertz_with_names</b> (@Clause, @VarList)	Insert a clause in the database
<b>@assertz_with_names</b> (@Head, @Body, @VarList)	Insert a clause in the database
<b>compile</b> (+Filename)	Compile a Prolog text
<b>@consult</b> (+Filename)	Read in a Prolog file
<b>:- ensure_loaded</b> (+Filename)	Insert a Prolog file in the source text once
<b>:- include</b> (+Filename)	Include a Prolog file in the source text
<b>load</b> (+Filename)	Load compiled Prolog text
<b>load_system</b> (+Filename)	Restore the old system status
<b>@reconsult</b> (+Filename)	Read in a Prolog file
<b>@#retract</b> (+Clause)	Remove a clause from database
<b>@#retract</b> (+Head, ?Body)	Remove a clause from database
<b>@#retract_with_names</b> (+Clause, ?VarList)	Remove a clause from database
<b>@#retract_with_names</b> (+Head, ?Body, ?VarList)	Remove a clause from database
<b>save_runtime</b> (+Filename)	Save system status for creating applications
<b>save_system</b> (+Filename)	Save current system status

### Query database

<b>@#clause</b> (+Head, ?Body)	Search the database for specific clauses
<b>@#clause_with_names</b> (+Head, ?Body, ?VarList)	Search the database for specific clauses
<b>@#current_predicate</b> (?Predicate)	Query visible predicates
<b>@#current_visible</b> (?DefModule, ?Predicate)	Query visible predicates
<b>@listing</b>	List the clauses in the database
<b>@listing</b> (+Functor)	List the clauses in the database
<b>@listing</b> (@Predicate)	List the clauses in the database
<b>@predicate_type</b> (@Predicate, ?Type)	Query predicate type

### Predicate attributes

<b>@debug_mode</b> (@Predicate, ?Before, +After)	Query and set the debug mode of a predicate
<b>:- discontinuous</b> (@PredicateList)	Declare discontinuous predicate definition
<b>:- dynamic</b> (@PredicateList)	Declare predicates as modifiable
<b>:- export</b> (@PredicateList)	Export predicates
<b>:- import</b> (+Module)	Import predicates
<b>:- import</b> (+Module, @PredicateList)	Import predicates selectively
<b>@list_mode</b> (@Predicate, ?Before, +After)	Query and set the list mode of a predicate

---

<code>:- meta(@PredicateList)</code>	Declare metapredicates
<code>@modify_mode(@Predicate, ?Before, +After)</code>	Query and set the modify mode of a predicate
<code>:- multifile(@PredicateList)</code>	Declare predicates as being scattered over several files
<code>@predicate_mode(@Predicate, ?Before, +After)</code>	Query and set the exception mode of a predicate
<code>@predicate_type(@Predicate, ?Type)</code>	Query predicate type
<code>:- private(@PredicateList)</code>	Declare predicates as private
<code>:- reexport(+Module)</code>	Reexport predicates
<code>:- reexport(+Module, @PredicateList)</code>	Reexport predicates

## Execution control

<code>!</code>	Disable backtracking
<code>+Goal1 , +Goal2</code>	Conjunction of goals
<code>+ConditionGoal -&gt; +ThenGoal</code>	IF-THEN conjunction of goals
<code>+ConditionGoal -&gt; +ThenGoal ; +ElseGoal</code>	IF-THEN-ELSE conjunction of goals
<code>#+Goal1 ; +Goal2</code>	Disjunction of goals
<code>\+(\@Goal)</code>	Negation through failure
<code>abort</code>	Return to the toplevel loop of IF/Prolog
<code>@#bagof(?Term, +Goal, ?TermList)</code>	Find selected solutions of a goal
<code>break</code>	Generate a new input loop
<code>@call(+Goal)</code>	Call a goal
<code>@call(+Name, +ArgList)</code>	Call a goal
<code>fail</code>	Initiate backtracking
<code>@findall(?Term, +Goal, ?TermList)</code>	Find all solutions of a goal
<code>for(+Start, +Counter, +End)</code>	Check range
<code>#for(+Start, ?Counter, +End)</code>	Generate a sequence of integers
<code>halt</code>	Terminate Prolog
<code>halt(+ExitStatus)</code>	Terminate Prolog and set exitstatus
<code>:- init(+Goal)</code>	Initialization goal
<code>:- initialization(+Goal)</code>	Initialization goal
<code>not @Goal</code>	Negation through failure
<code>@once(+Goal)</code>	Activate a goal once only
<code>program</code>	Automatically call a Prolog goal
<code>#repeat</code>	Generate a choice point
<code>@#setof(?Term, +Goal, ?TermList)</code>	Find selected solutions of a goal
<code>true</code>	Goal which is always true

## Contexts/exceptions

<b>alarm</b> (+Seconds)	Schedule alarm
<b>@catch</b> (+Goal, +CatchMask, +SubstituteGoal)	Intercept a context jump
<b>@context</b> (+Goal, +ContextInfo)	Create a context
<b>#current_signal</b> (?Signal, ?Mode)	Query signals and modes
<b>exception</b> (@Error, @Info)	Raise an exception
<b>@exception_handler</b> (+Goal, +Exception, +Handler)	Define local exception handling
<b>signal_control</b> (+Signal, +Command)	Control signal processing
<b>@signal_handler</b> (+Goal, ?Signal, +Handler)	Define local signal handling
<b>throw</b> (@Ball)	Initiate a context jump

## Debugger

<b>@debug</b> (+Goal)	Activate the debugger for a goal
<b>debug_begin</b>	User-defined debugger initialization
<b>debug_config</b> (+Parameter, ?Old, +New)	Configure the debugger
<b>debug_end</b>	User-defined termination of the debugger
<b>@debug_mode</b> (@Predicate, ?Before, +After)	Query and set the debug mode of a predicate

## Tracer

<b>@profile</b> (+Goal)	Activate profiling for a goal
<b>@profile</b> (+Goal, @Ports)	Activate profiling for a goal
<b>profile_reset</b>	Reset profiling statistics
<b>profile_stat</b>	Query profiling statistics
<b>profile_stat</b> (?List)	Query profiling statistics
<b>@trace</b> (+Goal)	Activate tracing for a goal
<b>trace_begin</b>	User-defined tracer initialization
<b>trace_config</b> (+Parameter, ?Old, +New)	Configure the tracer
<b>trace_end</b>	User-defined termination of the tracer
<b>trace_output</b> (@CallNumber, +Port, @Goal, @VariableNames)	User-defined output predicate for the tracer

## Memory management

<b>#current_memory_management</b> (?Area, ?Parameter, ?Value)	Query memory management parameters
--	------------------------------------

---

<b>garbage_collection</b> (+Area)	Activate garbage collection
<b>set_memory_management</b> (+Area, +Parameter, +Value)	Set memory management parameters
<b>statistics</b>	Print system values
<b>statistics</b> (?Type, ?ResultList)	Query system values

## System information

<b>#current_prolog_flag</b> (?Flag, ?Value)	Query Prolog flags
<b>help</b>	Display information about help
<b>help</b> (+Pattern)	Display help information
<b>manual</b>	Output manual pages
<b>manual</b> (+Functor)	Output manual pages
<b>manual</b> (@Predicate)	Output manual pages
<b>program_parameters</b> (?ParameterList)	Query parameters from IF/Prolog call
<b>prolog_serial</b> (?Serial)	Query IF/Prolog serial number
<b>prolog_version</b> (?Version)	Query IF/Prolog version
<b>proroot</b> (?Path)	Query path of IF/Prolog system
<b>set_prolog_flag</b> (+Flag, +Value)	Set value of a Prolog flag
<b>:- set_prolog_flag</b> (+Flag, +Value)	Set value of a Prolog flag (directive)
<b>system_hostid</b> (?Hostid)	Query machine identification
<b>system_name</b> (?SystemName)	Query operating system
<b>system_parameters</b> (?ParameterList)	Query system parameters from IF/Prolog call
<b>user_parameters</b> (?ParameterList)	Query user parameters from IF/Prolog call

## Operating system environment

<b>chdir</b> (+Dir)	Change current directory
<b>default_editor</b> (?EditorOld, +EditorNew)	Query and set the editor
<b>@edit</b>	Edit a file
<b>@edit</b> (+Filename)	Edit a file
<b>exec</b> (+Command)	Start program execution
<b>exec</b> (+Prog, @Args)	
<b>file_test</b> (+Pathname, +AccessMode)	Check access permission for a file
<b>#get_file_info</b> (+Pathname, ?Attribute, ?Value)	Get OS information for file
<b>get_file_info</b> (+Pathname, ?Info)	Get OS information for file
<b>getcwd</b> (?Dir)	Query current working directory
<b>getenv</b> (+Name, ?Value)	Query environment variable
<b>localtime</b> (+Time, ?Year, ?Month, ?Day, ?DoW, ?DoY, ?Hour, ?Min, ?Sec)	Determine the date and time

---

<b>stream_copy</b> (@Stream1, @Stream2)	Redefine streams
<b>system</b>	Start the operating system command interpreter
<b>system</b> (+Command)	Execute an operating system command
<b>system</b> (+Command, ?ExitStatus)	Execute an operating system command
<b>system</b> (+Command, ?Input, ?Output)	Execute an operating system command with input and output specified
<b>system</b> (+Command, ?Input, ?Output, ?Error, ?Pid)	Execute an operating system command with input and output specified
<b>timezone</b> (+Time, ?TimeZone, ?AlternateZone, ?DaylightSaving)	
<b>unix_fork</b> (?Pid)	Spawn process
<b>unix_getpid</b> (?Pid)	Query process ID of current process
<b>unix_kill</b> (+Pid, +Signal)	Send signal to a process
<b>unix_make_pipe</b> (-PipeName)	Create a pipe
<b>unix_wait</b> (?Pid)	Wait for termination of a child process
<b>unix_wait</b> (?Pid, ?ExitStatus)	Wait for termination of a child process
<b>windows_chdrive</b> (+DriveName)	Change drive
<b>windows_getdrive</b> (?DriveName)	Query drive

## Processes

<b>exec</b> (+Command)	Start program execution
<b>exec</b> (+Prog, @Args)	Start program execution
<b>system</b>	Start the operating system command interpreter
<b>system</b> (+Command)	Execute an operating system command
<b>system</b> (+Command, ?ExitStatus)	Execute an operating system command
<b>system</b> (+Command, ?Input, ?Output)	Execute an operating system command with input and output specified
<b>system</b> (+Command, ?Input, ?Output, ?Error, ?Pid)	Execute an operating system command with input and output specified
<b>unix_fork</b> (?Pid)	Spawn process
<b>unix_getpid</b> (?Pid)	Query process ID of current process
<b>unix_kill</b> (+Pid, +Signal)	Send signal to a process
<b>unix_wait</b> (?Pid)	Wait for termination of a child process
<b>unix_wait</b> (?Pid, ?ExitStatus)	Wait for termination of a child process

## Net communication

<b>#current_host</b> (?Host)	Query names of current host
<b>#current_socket</b> (?Domain, ?Type, ?Socket)	Query communication sockets

---

<b>#get_socket_option</b> (@Socket, ?Option, ?Value)	Get socket option
<b>#host_addr</b> (+Host, ?Addr)	Query host address
<b>#host_addr</b> (-Host, +Addr)	Query host name
<b>net_service</b> (+Service, ?Protocol, ?Port)	Query communication service
<b>select</b> (@Streams, +Timeout, -ReadyReadStream)	Synchronous input multiplexing
<b>select</b> (@Streams, +Timeout, -ReadyReadStream, -ReadyWriteStreams)	Synchronous input multiplexing
<b>select</b> (@SocksOrStreams, +Timeout, -ReadyReadSOS, -ReadyWriteSOS, -Timeleft)	Synchronous input multiplexing
<b>set_socket_option</b> (@Socket, +Option, +Value)	Set socket option
<b>socket</b> (+Domain, +Type, -Socket)	Create communication socket
<b>socket_accept</b> (@Socket, ?Addr, -NewSocket)	Accept a connection
<b>socket_bind</b> (@Socket, ?Addr)	Bind a name to a socket
<b>socket_close</b> (@Socket)	Close a socket
<b>socket_connect</b> (@Socket, @Addr)	Connect a socket
<b>socket_listen</b> (@Socket)	Listen for connections
<b>socket_listen</b> (@Socket, +Quelen)	Listen for connections
<b>socket_raw_receive</b> (@Socket, @Size, ?List, ?Length)	Receive data from socket
<b>socket_raw_receive</b> (@Socket, @Size, ?Addr, ?List, ?Length)	Receive data from socket
<b>socket_receive</b> (@Socket, ?List)	Receive data from socket
<b>socket_receive</b> (@Socket, ?Addr, ?List)	Receive data from socket
<b>socket_send</b> (@Socket, @List)	Send data to socket
<b>socket_send</b> (@Socket, @Addr, @List)	Send data to socket
<b>socket_shutdown</b> (@Socket)	Close a socket
<b>socket_shutdown</b> (@Socket, +How)	Close a socket

## Arithmetic functions

**	Exponentiation
*	Multiplication
//	Integer division
&	Bitwise conjunction (AND)
/	Division
\	Bitwise disjunction (OR)
\	Complement
#	Bitwise exclusive or (XOR)
>>	Bitwise right shift
<<	Bitwise left shift
-	Negation
-	Subtraction
+	Addition
+	Neutral function
abs(N)	Absolute value
acos(N)	Arc cosine
asin(N)	Arc sine
atan(N)	Arc tangent
ceiling(N)	Rounding floating-point number up to the next higher integer
cos(N)	Cosine
cosh(N)	Hyperbolic cosine
cputime	Used CPU time in seconds
csize	Total size of the local (control) stack (bytes)
cused	Portion of the local (control) stack used in %
dsize	Total size of the database (bytes)
dused	Portion of the database used in %
esize	Total size of the extended term stack (bytes)
eused	Portion of the extended term stack used in %
exp(N)	Exponential function
float(N)	Conversion of integer into floating-point number
float_fractional_part(N)	Fractional part of a floating-point number
float_integer_part(N)	Integer part of a floating-point number
floor(N)	Rounding floating-point number to the next lower integer
gsize	Total size of the global stack (bytes)
gused	Portion of the global stack used in %
log(N)	Natural logarithm
max(N,N)	Maximum of two numbers
min(N,N)	Minimum of two numbers
maxint	Largest representable short integer
minint	Smallest representable short integer
mod	Modulo function
pi	Constant $\pi$
random	Random number
rdiv	Rational division
rem	Division remainder

---

<code>round(N)</code>	Rounding floating-point number to integer
<code>sign(N)</code>	Sign function
<code>signum(N)</code>	Sign function
<code>sin(N)</code>	Sine
<code>sinh(N)</code>	Hyperbolic sine
<code>sqrt(N)</code>	Square root
<code>ssize</code>	Total size of the stacks (bytes)
<code>sused</code>	Portion of the stacks used in %
<code>tan(N)</code>	Tangent
<code>tanh(N)</code>	Hyperbolic tangent
<code>time</code>	System time in seconds
<code>truncate(N)</code>	Truncating floating-point number to integer
<code>tsize</code>	Total size of the trail stack (bytes)
<code>tused</code>	Portion of the trail stack used in %

## Prolog flags

The following Prolog flags may be queried and modified:

Flag	Value	Description
consult	<b>notify</b>	A message is output when a Prolog text is loaded.
	<b>nonotify</b>	No message is output when a Prolog text is loaded.
search_path	<i>Paths</i>	If input files are not specified using a complete path name, they are searched for in the directories specified in the <i>Paths</i> list. The default value of this Prolog flag is the value of the system parameter <code>-sp</code> , if specified, otherwise it is the value of the environment variable <code>PROPATH</code> or it is empty.
read_error	<b>error</b>	Syntax errors found by analysis with <code>read_term/2/3</code> , etc. raise an exception.
	<b>fail</b>	Syntax errors found by analysis with <code>read_term/2/3</code> , etc. are reported directly by the parser in a message and the predicate fails.
	<b>quiet</b>	Syntax errors found by analysis with <code>read_term/2/3</code> , etc. are not reported and the predicate fails.
report_read_error	<b>on</b>	Syntax errors found by analysis with <code>parse_atom/6</code> are reported directly by the parser in a message.
	<b>off</b>	Syntax errors found by analysis with <code>parse_atom/6</code> are not reported by the parser but can be queried with the <code>read_error/2/3</code> predicates.
warnings	<b>on</b>	For <code>consult/1</code> , etc. warnings are output for possible errors.
	<b>off</b>	For <code>consult/1</code> , etc. no warnings are output.
unknown	<b>error</b>	If a predicate is to be activated which is not defined, an exception is raised. This does not apply to predicates for which the exception mode has been set to <code>on</code> (see <code>predicate_mode/3</code> ).
	<b>fail</b>	If a predicate is to be activated which is not defined, backtracking is initiated, i.e. the predicate is evaluated with <code>fail</code> .
	<b>warning</b>	If a predicate is to be activated which is not defined, a warning is output and backtracking is initiated, i.e. this predicate is evaluated with <code>fail</code> . The warning is not output for predicates for which the exception mode has been set to <code>on</code> (see <code>predicate_mode/3</code> ).

---

debug	on	For <code>consult/1</code> and <code>reconsult/1</code> , <i>all</i> predicates are implicitly declared as dynamic (see <code>dynamic/1</code> ) to support program testing.
	off	For <code>consult/1</code> and <code>reconsult/1</code> , all the predicates which were not explicitly declared with <code>dynamic/1</code> are declared as static. These cannot then be modified, tested or displayed.
prompt	on	Output of the prompt in the Prolog input loop and interaction for backtracking for goals which have been successfully executed in the Prolog input loop and for which variables have been instantiated <i>always</i> take place, i.e. even if the standard input and the standard output are not terminals.
	off	Output of the prompt in the Prolog input loop and interaction for backtracking for goals which have been successfully executed in the Prolog input loop and for which variables have been instantiated take place <i>only</i> if the standard input and the standard output are terminals.
char_conversion	on	The character conversions defined with the predicate <code>char_conversion/2</code> are performed when terms are read.
	off	No character conversion is performed when terms are read.
backtrace	10	The maximal length of the <b>backtrace</b> -list. The list is created when an exception has been raised and contains the calling sequence of goals that has led to the exception. The elements of the list are predicate indicators ( <i>Functor/Arity</i> ). Goals that have been removed due to tail-recursion optimization (normally the last subgoal in a clause body) are not included in the list.
double_quotes	codes	Characters inside double quotes (") are to be interpreted as a list of character codes when reading a term (e.g. with <code>read/1/2</code> ).
	chars	Characters inside double quotes (") are to be interpreted as a list of characters when reading a term (e.g. with <code>read/1/2</code> ).
	atom	Characters inside double quotes (") are to be interpreted as an atom when reading a term (e.g. with <code>read/1/2</code> ).
extended_syntax	on	Cyclic terms can be entered using the notation $\text{Variable} :: \text{Term}.$ When reading such a cyclic term, <i>Variable</i> is unified with <i>Term</i> . The above notation is also used by output of cyclic terms (see <code>=/2</code> ).
	off	Cyclic terms can be entered in the above notation, but <i>Variable</i> is <b>not</b> unified with <i>Term</i> .

---

<code>nested_comments</code>	<b>off</b>	Nested comments are not allowed in a Prolog text. Begin of a comment inside a comment is ignored, which normally leads to a syntax error.
	<b>on</b>	Nested comments are accepted in a Prolog text.
<code>write_depth</code>	<b>10</b>	Maximal depth of terms output in the Prolog toplevel (see <code>break/0</code> ). This applies to variable instantiations of a query and to exception messages. The predicate <code>write_formatted/2/3</code> takes this value in account, if a term is to be output with the format <code>%w</code> .

The following Prolog flags can only be queried but not changed. Their value depends on the command line parameters specified when the Prolog system was started (see `system_parameters/1`).

Flag	Value	Description
<code>signal</code>	<b>on</b>	The system parameter <code>-nosignal</code> was <b>not</b> specified.
	<b>off</b>	The system parameter <code>-nosignal</code> was specified.
<code>notty</code>	<b>on</b>	The system parameter <code>-notty</code> was specified.
	<b>off</b>	The system parameter <code>-notty</code> was <b>not</b> specified.
<code>iso</code>	<b>on</b>	The system parameter <code>-iso</code> was specified.
	<b>off</b>	The system parameter <code>-iso</code> was <b>not</b> specified.

The following Prolog flags supply information on the implementation of the Prolog system or on the operating system (the values cannot be changed):

Flag	Description			
<code>max_arity</code>	The value indicates the maximum arity of predicates and structures.			
<code>min_integer</code>	The value indicates the smallest representable machine-precision (32- or 64-bit) integer.			
<code>max_integer</code>	The value indicates the largest representable machine-precision (32- or 64-bit) integer.			
<code>bounded</code>	This flag can assume the following values:			
	<table> <tr> <td><b>true</b></td> <td>Integer arithmetic returns correct values only if the operands and the mathematically correct result lie within the closed interval (<code>min_integer</code>, <code>max_integer</code>).</td> </tr> <tr> <td><b>false</b></td> <td>Integer arithmetic always returns correct results or an exception is created if this is not possible.</td> </tr> </table>	<b>true</b>	Integer arithmetic returns correct values only if the operands and the mathematically correct result lie within the closed interval ( <code>min_integer</code> , <code>max_integer</code> ).	<b>false</b>
<b>true</b>	Integer arithmetic returns correct values only if the operands and the mathematically correct result lie within the closed interval ( <code>min_integer</code> , <code>max_integer</code> ).			
<b>false</b>	Integer arithmetic always returns correct results or an exception is created if this is not possible.			
<code>integer_rounding_function</code>	This flag can assume the following values:			
	<b>down</b>	The arithmetic functions <code>//</code> and <code>rem</code> round their result down if necessary.		
	<b>toward_zero</b>	The arithmetic functions <code>//</code> and <code>rem</code> round the absolute value of their result down if necessary.		
<code>float_min</code>	The value indicates the smallest representable floating-point number.			

`float_max`      The value indicates the largest representable floating-point number.

## Input/Output

The following aliases are built into the Prolog system:

Alias	Assignment
<code>user_input</code>	Standard input
<code>user_output</code>	Standard output
<code>user_error</code>	Standard error output
<code>keyboard</code>	Controlling terminal (input)
<code>screen</code>	Controlling terminal (output)
<code>current_input</code>	Current input
<code>current_output</code>	Current output
<code>current_error</code>	Current error output
<code>debug_input</code>	Used by the debugger
<code>debug_output</code>	Used by the debugger
<code>trace_output</code>	Used by the tracer

The following types are possible for the built-in device drivers:

Device	Type	Description
file	<code>regular</code>	Regular file
	<code>block_special</code>	Special file
	<code>character_special</code>	Special file
	<code>directory</code>	Directory
	<code>fifo_special</code>	Named pipe
	<code>symbolic_link</code>	Symbolic link
	<code>unknown</code>	Unknown type
standard	<code>regular</code>	Regular file
	<code>block_special</code>	Special file
	<code>character_special</code>	Special file
	<code>directory</code>	Directory
	<code>fifo_special</code>	Named pipe
	<code>symbolic_link</code>	Symbolic link
	<code>unknown</code>	Unknown type
pipe	<code>pipe</code>	Pipe
string	<code>string</code>	String, atom
socket	<code>socket</code>	Socket
null	<code>null</code>	Null device (UNIX: <code>/dev/null</code> )

The following control commands are implemented in the built-in device drivers of the Prolog system:

Device	Command	Meaning	
file	<code>stream_type(Arg)</code>	see <code>stream_type/2</code>	
	<code>stream_size(N)</code>	Unifies <i>N</i> with the size of the file	
	<code>isatty</code>	<code>stream_control/2</code> succeeds if <i>Stream</i> is a terminal	
	<code>fileno(N)</code>	Unifies <i>N</i> with the operating system dependent file number of the file	
	<code>clear_eos</code>	see <code>clear_eos/0/1</code>	
	<code>clear_eol</code>	see <code>clear_eol/0/1</code>	
	<code>clear_screen</code>	see <code>clear_screen/0/1</code>	
	<code>tty_size(Z,S)</code>	see <code>tty_size/2/3</code>	
	<code>move_cursor(Z,S)</code>	see <code>move_cursor/2/3</code>	
	<code>seek(P)</code>	see <code>set_stream_position/2</code>	
	<code>tell(P)</code>	see <code>stream_property/2</code>	
	standard	<code>stream_type(Arg)</code>	see <code>stream_type/2</code>
<code>stream_size(N)</code>		Unifies <i>N</i> with the size of the file	
<code>isatty</code>		<code>stream_control/2</code> succeeds if <i>Stream</i> is a terminal	
<code>fileno(N)</code>		Unifies <i>N</i> with the operating system dependent file number of the file	
<code>clear_eos</code>		see <code>clear_eos/0/1</code>	
<code>clear_eol</code>		see <code>clear_eol/0/1</code>	
<code>clear_screen</code>		see <code>clear_screen/0/1</code>	
<code>tty_size(Z,S)</code>		see <code>tty_size/2/3</code>	
<code>move_cursor(Z,S)</code>		see <code>move_cursor/2/3</code>	
pipe		<code>stream_type(Arg)</code>	see <code>stream_type/2</code>
		<code>fileno(N)</code>	Unifies <i>N</i> with the operating system dependent file number of the file
socket		<code>stream_type(Arg)</code>	see <code>stream_type/2</code>
	<code>fileno(N)</code>	Unifies <i>N</i> with the operating system dependent file number of the file	
string	<code>stream_type(Arg)</code>	see <code>stream_type/2</code>	
	<code>stream_size(N)</code>	Unifies <i>N</i> with the size of the string	
	<code>string(Arg)</code>	Unifies <i>Arg</i> with the contents of the string	
	<code>seek(P)</code>	see <code>set_stream_position/2</code>	
	<code>tell(P)</code>	see <code>stream_property/2</code>	
null	<code>stream_type(Arg)</code>	see <code>stream_type/2</code>	
-	<code>get_last(Z,S)</code>	see <code>get_last/2/3</code>	

## Syntax messages

Number	Message
1	. expected
2	) expected
3	end of comment */ expected
4	illegal character
5	start of term expected
6	unknown stand alone character
7	operator expected
8	in/postfix operator or end of term expected
9	too large constant
10	] or , expected
11	} expected
12	' or " expected
13	, or ) expected
14	illegal number
15	precedence error
16	] expected
17	interrupt
18	i/o error
19	illegal escape sequence
20	illegal character code
21	exceeded max_arity
22	unbound variable expected

## Signals

The following signal names are known:

Prolog name	C name
interrupt	SIGINT
abort	SIGABRT
alarm	SIGALRM
pipe	SIGPIPE
quit	SIGQUIT
termination	SIGTERM
user_1	SIGUSR1
user_2	SIGUSR2

## Debugger commands

A command that is only available in the alpha surface of the debugger is identified by the character #.

### Breakpoints

Command	Short	Functionality
<b>activate_stop</b>	<b>as</b>	Reactivate current breakpoints
<b>activate_stop</b> (Numbers)	<b>as</b> (...)	Reactivate breakpoints
<b>deactivate_stop</b>	<b>ds</b>	Deactivate all breakpoints temporarily
<b>deactivate_stop</b> (Numbers)	<b>ds</b> (...)	Deactivate breakpoint temporarily
<b>remove_stop</b>	<b>rs</b>	Remove all current explicit breakpoints
<b>remove_stop</b> (Numbers)	<b>rs</b> (...)	Remove explicit breakpoints
<b>stop</b> (Breakpoint, Ports, Conditions, Actions)	<b>st</b> (...)	Set breakpoint
<b>stop</b> (MonitoredVariable, BindingType, Conditions, Actions)	<b>st</b> (...)	Set breakpoint to the monitoring of variables
<b>stop_interactive</b> (Breakpoint)	<b>si</b> (...)	Set interactive breakpoint
<b>stop_interactive</b> (Breakpoint, Ports)	<b>si</b> (...)	Set interactive breakpoint
<b>stop_interactive</b> (Breakpoint, Ports, Conditions, Actions)	<b>si</b> (...)	Set interactive breakpoint
<b>stop_-interactive</b> (MonitoredVariable, BindingType, Conditions, Actions)	<b>si</b> (...)	Set interactive breakpoint to the monitoring of variables
<b>activate_stop</b>	<b>as</b>	Reactivate current breakpoints
<b>activate_stop</b> (Numbers)	<b>as</b> (...)	Reactivate breakpoints
<b>deactivate_stop</b>	<b>ds</b>	Deactivate all breakpoints temporarily
<b>deactivate_stop</b> (Numbers)	<b>ds</b> (...)	Deactivate breakpoint temporarily
<b>remove_stop</b>	<b>rs</b>	Remove all current explicit breakpoints
<b>remove_stop</b> (Numbers)	<b>rs</b> (...)	Remove explicit breakpoints
<b>stop</b> (Breakpoint, Ports, Conditions, Actions)	<b>st</b> (...)	Set breakpoint
<b>stop</b> (MonitoredVariable, BindingType, Conditions, Actions)	<b>st</b> (...)	Set breakpoint to the monitoring of variables
<b>stop_interactive</b> (Breakpoint)	<b>si</b> (...)	Set interactive breakpoint
<b>stop_interactive</b> (Breakpoint, Ports)	<b>si</b> (...)	Set interactive breakpoint
<b>stop_interactive</b> (Breakpoint, Ports, Conditions, Actions)	<b>si</b> (...)	Set interactive breakpoint

<b>stop_-</b> <b>interactive</b> (MonitoredVariable, BindingType, Conditions, Actions)	<b>si</b> (...)	Set interactive breakpoint to the monitoring of variables
---	-----------------	---

### Control commands

Command	Short	Functionality
<b>abort</b>	<b>A</b>	Terminate interactive debugger
<b>break</b>	<b>B</b>	Start new input loop
<b>call</b> (Goal) [ @ +Module ]	<b>ca</b> (...)	Execute a Prolog goal
<b>halt</b>	<b>H</b>	Terminate debugger and IF/Prolog
<b>system</b>	<b>S</b>	Interrupt interactive debugger and start the shell
<b>abort</b>	<b>A</b>	Terminate interactive debugger
<b>break</b>	<b>B</b>	Start new input loop
<b>call</b> (Goal) [ @ +Module ]	<b>ca</b> (...)	Execute a Prolog goal
<b>halt</b>	<b>H</b>	Terminate debugger and IF/Prolog
<b>system</b>	<b>S</b>	Interrupt interactive debugger and start the shell

### Configuration



Command	Short	Functionality
<b>port</b> (Ports, Switch)	<b>p</b> (...)	Set debugger ports
<b>set_debug_depth</b> (Depth)	<b>sdd</b> (...)	Set structure depth
# <b>set_history</b> (Number)	<b>sh</b> (...)	Configure history
<b>set_trace_depth</b> (Depth)	<b>std</b> (...)	Set structure depth for trace outputs
# <b>set_trace_length</b> (Length)	<b>stl</b> (...)	Set length of trace field
<b>trace</b> (Switch)	<b>t</b> (...)	Switch trace mode
<b>trace_port</b> (Ports, Switch)	<b>tp</b> (...)	Set trace ports
<b>port</b> (Ports, Switch)	<b>p</b> (...)	Set debugger ports
<b>set_debug_depth</b> (Depth)	<b>sdd</b> (...)	Set structure depth
# <b>set_history</b> (Number)	<b>sh</b> (...)	Configure history
<b>set_trace_depth</b> (Depth)	<b>std</b> (...)	Set structure depth for trace outputs
# <b>set_trace_length</b> (Length)	<b>stl</b> (...)	Set length of trace field
<b>trace</b> (Switch)	<b>t</b> (...)	Switch trace mode
<b>trace_port</b> (Ports, Switch)	<b>tp</b> (...)	Set trace ports

### Data manipulation

Command	Short	Functionality
---------	-------	---------------

**unify**(Variable, Value)      ...=...      Unify variable  
**unify**(Variable, Value)      ...=...      Unify variable

**Execution control**

<b>Command</b>	<b>Short</b>	<b>Functionality</b>
<b>continue</b>	<b>c</b>	Continue to next breakpoint
<b>fast_skip</b>	<b>fs</b>	Accelerate execution to exit of the subsequent subgoal
<b>fast_skip</b> (Number)	<b>fs(...)</b>	Accelerate execution to exit of a subsequent subgoal
<b>nonstop</b>	<b>no</b>	Continue without stopping
<b>skip</b>	<b>s</b>	Execute up to exit of the subgoal
<b>skip</b> (Number)	<b>s(...)</b>	Execute up to exit of a subsequent subgoal
<b>step</b>		Execute up to next monitored port
<b>continue</b>	<b>c</b>	Continue to next breakpoint
<b>fast_skip</b>	<b>fs</b>	Accelerate execution to exit of the subsequent subgoal
<b>fast_skip</b> (Number)	<b>fs(...)</b>	Accelerate execution to exit of a subsequent subgoal
<b>nonstop</b>	<b>no</b>	Continue without stopping
<b>skip</b>	<b>s</b>	Execute up to exit of the subgoal
<b>skip</b> (Number)	<b>s(...)</b>	Execute up to exit of a subsequent subgoal
<b>step</b>		Execute up to next monitored port

**Display information**

<b>Command</b>	<b>Short</b>	<b>Functionality</b>
<b>help</b>	<b>h</b>	Display help information
<b># snapshot</b>	<b>sn</b>	Store or append field in previous used file
<b># snapshot</b> (File)	<b>sn(...)</b>	Store field in file
<b># snapshot</b> (File, Mode)	<b>sn(...)</b>	Store or append field in file
<b>trace</b>	<b>t</b>	Logging passes through ports
<b># view_and_box</b>	<b>&amp;</b>	Display the goal box in the window
<b># view_or_box</b>	<b>#</b>	Display the clause box in the window
<b># view_history</b>	<b>vh</b>	Display the goal or clause box in the history list
<b># view_trace</b>	<b>vt</b>	Display the trace field in the window
<b>view_ancestors</b>	<b>a</b>	Show dynamic call chain
<b>view_ancestors</b> (Number)	<b>a(...)</b>	Show dynamic call chain
<b>view_configuration</b>	<b>vco</b>	Display default values
<b>view_constraints</b>	<b>vc</b>	Display constraints
<b>view_module</b>	<b>vm</b>	Display list of predicates in module

<b>view_module</b> (Module)	<b>vm(...)</b>	Display list of predicates in module
<b>view_predicate</b>	<b>v</b>	Display listing of a predicate
<b>view_predicate</b> (Predicate)	<b>v(...)</b>	Display listing of a predicate
<b>view_stops</b>	<b>vs</b>	Display explicit breakpoints
<b>view_variable</b>	<b>vv</b>	Display variable constraints
<b>view_variable</b> (VariableName)	<b>vv(...)</b>	Display variable constraints
<b>help</b>	<b>h</b>	Display help information
<b># snapshot</b>	<b>sn</b>	Store or append field in previous used file
<b># snapshot</b> (File)	<b>sn(...)</b>	Store field in file
<b># snapshot</b> (File, Mode)	<b>sn(...)</b>	Store or append field in file
<b>trace</b>	<b>t</b>	Logging passes through ports
<b># view_and_box</b>	<b>&amp;</b>	Display the goal box in the window
<b># view_or_box</b>	<b>#</b>	Display the clause box in the window
<b># view_history</b>	<b>vh</b>	Display the goal or clause box in the history list
<b># view_trace</b>	<b>vt</b>	Display the trace field in the window
<b>view_ancestors</b>	<b>a</b>	Show dynamic call chain
<b>view_ancestors</b> (Number)	<b>a(...)</b>	Show dynamic call chain
<b>view_configuration</b>	<b>vco</b>	Display default values
<b>view_constraints</b>	<b>vc</b>	Display constraints
<b>view_module</b>	<b>vm</b>	Display list of predicates in module
<b>view_module</b> (Module)	<b>vm(...)</b>	Display list of predicates in module
<b>view_predicate</b>	<b>v</b>	Display listing of a predicate
<b>view_predicate</b> (Predicate)	<b>v(...)</b>	Display listing of a predicate
<b>view_stops</b>	<b>vs</b>	Display explicit breakpoints
<b>view_variable</b>	<b>vv</b>	Display variable constraints
<b>view_variable</b> (VariableName)	<b>vv(...)</b>	Display variable constraints

## Jump commands

Command	Short	Functionality
<b>back_clause</b>	<b>bc</b>	Branch back to clause head
<b>back_parent</b>	<b>bp</b>	Branch back to CALL port of parent goal
<b>back_subgoal</b>	<b>b</b>	Branch back to CALL port of subgoal
<b>error</b> (Error, Info)	<b>err(...)</b>	Raise an exception
<b>exit</b>	<b>x</b>	Force current subgoal to succeed
<b>fail</b>	<b>f</b>	Force the current subgoal to fail
<b>back_clause</b>	<b>bc</b>	Branch back to clause head
<b>back_parent</b>	<b>bp</b>	Branch back to CALL port of parent goal
<b>back_subgoal</b>	<b>b</b>	Branch back to CALL port of subgoal
<b>error</b> (Error, Info)	<b>err(...)</b>	Raise an exception
<b>exit</b>	<b>x</b>	Force current subgoal to succeed
<b>fail</b>	<b>f</b>	Force the current subgoal to fail

Scrolling and paging

Command	Short	Functionality
# +		Scroll line by line forwards
# -		Scroll line by line backwards
# ++		Scroll page by page forwards
# --		Scroll page by page backwards
# forward		Scroll line by line forwards
# backward		Scroll line by line backwards
# fast_forward		Scroll page by page forwards
# fast_backward		Scroll page by page backwards
# view_previous_screen	<	Display the previous goal or clause box
# view_last_screen	<<	Display the last goal or clause box stored
# view_next_screen	>	Display the next goal or clause box
# view_first_screen	>>	Display the first goal or clause box stored
# +		Scroll line by line forwards
# -		Scroll line by line backwards
# ++		Scroll page by page forwards
# --		Scroll page by page backwards
# forward		Scroll line by line forwards
# backward		Scroll line by line backwards
# fast_forward		Scroll page by page forwards
# fast_backward		Scroll page by page backwards
# view_previous_screen	<	Display the previous goal or clause box
# view_last_screen	<<	Display the last goal or clause box stored
# view_next_screen	>	Display the next goal or clause box
# view_first_screen	>>	Display the first goal or clause box stored

## Debugger parameters

The following debugger parameters are available (default values are printed in boldface):

Parameter	Value	Description
call	<b>on</b>	The interactive debugger stops at each CALL port (implicit breakpoint).
	off	The interactive debugger will not normally stop at the CALL port unless a user-defined breakpoint exists or an execute command requires it to.
exit	<b>on/off</b>	the same applies to the EXIT port
redo	<b>on/off</b>	the same applies to the REDO port
fail	<b>on/off</b>	the same applies to the FAIL port
error	<b>on/off</b>	the same applies to the ERROR port
wakeup	<b>on/off</b>	the same applies to the WAKEUP port
suspend	<b>on/off</b>	the same applies to the SUSPEND port
trymatch	<b>on/off</b>	the same applies to the TRYMATCH port
failmatch	<b>on/off</b>	the same applies to the FAILMATCH port
enterbody	<b>on/off</b>	the same applies to the ENTERBODY port
exitbody	<b>on/off</b>	the same applies to the EXITBODY port
redobody	<b>on/off</b>	the same applies to the REDOBODY port
failbody	<b>on/off</b>	the same applies to the FAILBODY port
display_depth	<b>7</b>	The maximum structure depth is restricted by this parameter when terms are output by the debugger. If 0 is specified, there is no restriction.
history	<b>3</b>	The number of goal and clause boxes to be stored for the alpha and Motif interfaces.
trace	<b>on</b>	Tracing is activated.
	<b>off</b>	Tracing is deactivated.

---

<code>display</code>	<code>automatic</code>	The user interface is set automatically by the system in accordance with the system environment. This value is the default setting unless the system parameter <code>stream</code> was specified when the Prolog system was started (see <code>system_parameters/1</code> ).
	<code>alpha</code>	The most basic interface for alphanumeric screens is set. The I/O device must be a terminal and the device driver must be able to perform terminal control functions. If this is not the case, the value is automatically reset to <code>stream</code> .
	<code>stream</code>	The most basic command-oriented interface is set. It does not require any terminal functions, i.e. it can also be used for redirected input/output. This value is the default setting if the system parameter <code>stream</code> was specified when the Prolog system was started (see <code>system_parameters/1</code> ).
	<code>motif</code>	An attempt is made to start the debugger's Motif interface on the default X display (see Motif documentation). If this attempt is unsuccessful, the value is automatically reset to <code>alpha</code> .
	<code>motif(Display)</code>	<i>Display</i> is regarded as the specification of an X display, and an attempt is made to start the debugger's Motif interface on this X display. If this attempt is not successful, the value is automatically reset to <code>alpha</code> .

## Tracer parameters

The following tracer parameters are available (default values are printed in boldface):

Parameter	Value	Description
call	<b>on</b>	The tracer logs every pass through this port or calls the user-defined predicate <code>trace_output/4</code> .
	<b>off</b>	The tracer does not log passes through this port.
exit	<b>on/off</b>	the same applies to the EXIT port
redo	<b>on/off</b>	the same applies to the REDO port
fail	<b>on/off</b>	the same applies to the FAIL port
error	<b>on/off</b>	the same applies to the ERROR port
wakeup	<b>on/off</b>	the same applies to the WAKEUP port
suspend	<b>on/off</b>	the same applies to the SUSPEND port
trymatch	<b>on/off</b>	the same applies to the TRYMATCH port
failmatch	<b>on/off</b>	the same applies to the FAILMATCH port
enterbody	<b>on/off</b>	the same applies to the ENTERBODY port
exitbody	<b>on/off</b>	the same applies to the EXITBODY port
redobody	<b>on/off</b>	the same applies to the REDOBODY port
failbody	<b>on/off</b>	the same applies to the FAILBODY port
display_depth	<b>7</b>	The maximum structure depth is restricted by this parameter when terms are output by the tracer. If 0 is specified, there is no restriction.
trace_length	<b>100</b>	Number of trace outputs to be stored if the alpha interface is being used.

## C Interface

### Active Prolog goals

#### C Function

void  
**PrologClose**(t\_cursor ActGoal)  
 BOOLEAN  
**PrologError**(t\_cursor ActGoal, TERM \*Term)  
 BOOLEAN  
**PrologFetch**(t\_cursor ActGoal)  
 t\_cursor  
**PrologOpen**(MODULE Module, TERM Goal,  
 TERM VarList)

#### Functionality

Deactivate the current Prolog goal  
 Query error  
 Find solutions for a goal  
 Create current Prolog goal

### Analyze term

#### C Function

TERM  
**TermArg**(ARITY Argno, TERM Term)  
 TERMTYPE  
**TermDecompose**(TERM Term,  
 TERMINFO \*Info)  
 TERMTYPE  
**TermType**(TERM Term)

#### Functionality

Access structure arguments  
 Classify term  
 Classify term

### Test Prolog terms

#### C Function

BOOLEAN  
**TermIsAtom**(TERM Term, STRING \*Name)  
 BOOLEAN  
**TermIsCompound**(TERM Term,  
 STRING Functor, ARITY Arity, ...)  
 BOOLEAN  
**TermIsFloat**(TERM Term, double \*Value)  
 BOOLEAN  
**TermIsFloatExpression**(TERM Term,  
 double \*Value)  
 BOOLEAN  
**TermIsFunctor**(TERM Term, STRING Functor,  
 ARITY Arity)

#### Functionality

Test for atom  
 Test for structure  
 Test for floating-point number  
 Test for floating-point expression  
 Test for structure

BOOLEAN	<b>TermIsInteger</b> (TERM Term, long *Value)	Test for integer
BOOLEAN	<b>TermIsIntegerExpression</b> (TERM Term, long *Value)	Test for integer expression
BOOLEAN	<b>TermIsList</b> (TERM Term, TERM *Head, TERM *Tail)	Test for list
BOOLEAN	<b>TermIsNil</b> (TERM Term)	Test for empty list
BOOLEAN	<b>TermIsUniversal</b> (TERM Term, STRING Functor, ARITY Arity, TERM Args[])	Test for structure
BOOLEAN	<b>TermIsVar</b> (TERM Term)	Test for variable

### Test results (passive C interface)

#### C Function

TERM	<b>PrologGoal</b> (t_cursor ActGoal)
BOOLEAN	<b>PrologIsFloat</b> (t_cursor ActGoal, STRING VarName, double *Value)
BOOLEAN	<b>PrologIsInteger</b> (t_cursor ActGoal, STRING VarName, long *Value)
BOOLEAN	<b>PrologIsString</b> (t_cursor ActGoal, STRING VarName, STRING *Name)
BOOLEAN	<b>PrologIsTerm</b> (t_cursor ActGoal, STRING VarName, TERM *Term)

### Connect to Prolog

#### C Function

void	<b>Cboot</b> (void)
void	<b>CPRED</b> (STRING Functor, ARITY Arity, CPREDFUN Function, size_t Size)

#### Functionality

Prolog goal as term
Get floating-point number from variable
Get integer from variable
Get character string from variable
Get Prolog term from variable

void	<b>CPRIM</b> (STRING Functor, ARITY Arity, CPRIMFUN Function)	Add a simple C function to IF/Prolog
void	<b>Cshutdown</b> (void)	Call a C function on IF/Prolog shutdown
void	<b>MCPRED</b> (STRING Module, STRING Functor, ARITY Arity, CPREDFUN Function, size_t Size)	Add a complex C function to a module
void	<b>MCPRIM</b> (STRING Module, STRING Functor, ARITY Arity, CPRIMFUN Function)	Add a simple C function to a module
TERM	<b>PrologArg</b> (ARITY Argno)	Access predicate arguments

## Control backtracking

### C Function

void	<b>PrologEnableEpilog</b> (void)
void	<b>PrologUndo</b> (void)

### Functionality

Enable epilog
Undo unification

## Construct lists

### C Function

BOOLEAN	<b>TermAddList</b> (TERM *Tail, TERM Term)
BOOLEAN	<b>TermCloseList</b> (TERM Tail)
TERM	<b>TermMakeList</b> (TERM Head, TERM Tail)
TERM	<b>TermOpenList</b> (TERM *Tail)

### Functionality

Add to a Prolog list
Close a Prolog list
Generate a Prolog list
Generate a Prolog list

## Construct Prolog terms

### C Function

TERM	<b>TermMakeAtom</b> (STRING name)
TERM	<b>TermMakeCompound</b> (STRING Functor, ARITY Arity, ...)

### Functionality

Generate a Prolog atom
Generate a Prolog structure

TERM	<b>TermMakeFloat</b> (double Value)	Generate a Prolog floating-point number
TERM	<b>TermMakeFunctor</b> (STRING functor, ARITY arity)	Generate a Prolog structure
TERM	<b>TermMakeInteger</b> (long Value)	Generate a Prolog integer
TERM	<b>TermMakeList</b> (TERM Head, TERM Tail)	Generate a Prolog list
TERM	<b>TermMakeVar</b> (void)	Generate an empty list
TERM	<b>TermMakeUniversal</b> (STRING Functor, ARITY Arity, TERM Args[])	Generate a Prolog structure
TERM	<b>TermMakeVar</b> (void)	Generate a Prolog variable

## Device driver function

### C Function

void	<b>DeviceCreate</b> (t_device *Device)
BOOLEAN	<b>StreamFlush</b> (t_stream Stream, long *Number)
t_iomode	<b>StreamMode</b> (t_stream Stream)

### Functionality

Register device driver
Flush Prolog output buffer
Query stream mode

## Formatted output

### C Function

size_t	<b>fprintt</b> (FILE *FilePointer, STRING Format, ...)
size_t	<b>outputlen</b> (STRING Format, ...)
size_t	<b>printt</b> (STRING Format, ...)
size_t	<b>sprintt</b> (char *Buffer, STRING Format, ...)

### Functionality

Formatted output of Prolog terms and C data objects
Determine length of formatted output
Formatted output of Prolog terms and C data objects
Formatted output of Prolog terms and C data objects

## Initialize Prolog (passive C interface)

### C Function

### Functionality

BOOLEAN

**EndProlog**(void)

BOOLEAN

**InitProlog**(int Number,  
STRING \*ArgumentVector, FILE \*Input,  
FILE \*Output, FILE \*Error)

Release memory space used by  
IF/Prolog

Initialize IF/Prolog and define the  
standard media

## Memory management

### C Function

void

**TermCollect**(TERMCONTEXT context, ...)

void

**TermCollectSequence**(TERM-  
CONTEXT context, size\_t N,  
TERM \*TermList)

TERMCONTEXT

**TermContext**(void)

### Functionality

Release term variables

Release term variables

Current set of term variables

## Prepare Prolog terms

### C Function

STRING

**ParseError**(int Number)

BOOLEAN

**ParseProlog**(STRING GoalAtom,  
t\_parse\_context \*Context)

MODULE

**PrologModule**(STRING Name)

### Functionality

Assignment of syntax error number to  
syntax error message

Parse a goal string and store the  
information

Get Prolog module

## Passive Prolog terms

### C Function

t\_query

**QueryCreate**(MODULE Module, TERM Goal,  
TERM VarList)

void

**QueryDispose**(t\_query Query)

t\_cursor

**QueryOpen**(t\_query Query)

### Functionality

Create passive Prolog goal

Release passive Prolog goal

Activate a passive Prolog goal

**Raise errors****C Function**

```
void
    ErrorContextClear(void)
BOOLEAN
    ErrorContextGet(ERRORCLASS *Errorclass,
    STRING *Parm1, STRING *Parm2,
    ARITY *Argno, TERM *Culprit)
BOOLEAN
    ErrorContextIsSet(void)
void
    ErrorContextSet(ERRORCLASS Errorclass,
    STRING Parm1, STRING Parm2, ARITY Argno,
    TERM Culprit)
```

**Functionality**

```
Clear error context
Query error context
Status of the error context
Set error context
```

**Unification****C Function**

```
BOOLEAN
    TermUnify(TERM Term1, TERM Term2)
BOOLEAN
    TermUnifyAtom(TERM Term, STRING Name)
BOOLEAN
    TermUnifyCompound(TERM Term,
    STRING Functor, ARITY Arity, ...)
BOOLEAN
    TermUnifyFloat(TERM Term, double Value)
BOOLEAN
    TermUnifyFunctor(TERM Term,
    STRING Functor, ARITY Arity)
BOOLEAN
    TermUnifyInteger(TERM Term, long Value)
BOOLEAN
    TermUnifyList(TERM Term, TERM Head,
    TERM Tail)
BOOLEAN
    TermUnifyNil(TERM Term)
BOOLEAN
    TermUnifyUniversal(TERM Term,
    STRING Functor, ARITY Arity, TERM *Args[])
BOOLEAN
    TermUnifyVar(TERM Term1, TERM Term2)
```

**Functionality**

```
Unify terms
Unify term with atom
Unify term with structure
Unify term with floating-point number
Unify term with structure
Unify term with integer
Unify term with list
Unify term with the empty list
Unify term with structure
Unify terms
```



# Bibliography

- [1] IF/Prolog V5.3 Reference Manual
- [2] IF/Prolog V5.3 User's Guide
- [3] IF/Prolog V5.3 OSF/Motif Interface
- [4] IF/Prolog V5.3 Informix Interface
- [5] IF/Prolog V5.3 Constraints Package
- [6] IF/Prolog V5.3 Quick Reference
- [7] IF/Prolog V5.3 Windows Interfaces
- [8] IF/Prolog V5.3 Java Interface
- [9] IF/Prolog V5.3 BDD Package
- [10] X/Open CAE (Common Applications Environment) Specification. System Interfaces and Headers, Issue 4. Prentice Hall 1994.
- [11] International Standard, ISO/IEC IS 13211-1. International Organization for Standardization, 1995
- [12] William F. Clocksin, Chris S. Mellish: Programming in PROLOG. Standard Edition Berlin et al.: Springer 1995.
- [13] Ivan Bratko: PROLOG. Programming for Artificial Intelligence Second Edition, Addison-Wesley 1990.
- [14] Leon Sterling, Ehud Shapiro: The Art of PROLOG. Advanced Programming Techniques. Cambridge Massachusetts: MIT Press, 1986