ECL^{i}PS^e Built-In Predicates

August 7, 1999

Abstract

This manual lists the ECL^{i}PS^e built-in predicates in groups with a one-line description. For the complete descriptions, refer to the online help or follow the links in the hypertext version of this document.

Contents

Chapter 1

The ECL^{i}PS^e Built-In Predicates

1.1 All Solutions

bagof(?Term, +Goal, ?List)
Succeeds if List is the (non–empty) list of all instances of Term such that Goal is provable.

coverof(?Term, +Goal, ?List)
Succeeds if List is the (non–empty) list of all the most general instances of Term such that Goal is provable.

findall(?Term, +Goal, ?List)
List is the (possibly empty) list of all instances of Term such that Goal is provable.

setof(?Term, +Goal, ?List)
Succeeds if List is the (non–empty) ordered list of all instances of Term such that Goal is provable.

1.2 Arithmetic

***(+Number1, +Number2, ?Result)**
Evaluates the product Number1 * Number2 and unifies the resulting value with Result.

+(+Number, ?Result)
Checks if Number is a number and unifies it with Result.

+(+Number1, +Number2, ?Result)
Evaluates the sum Number1 + Number2 and unifies the resulting value with Result.

–(+Number, ?Result)
Unifies the negative of Number with Result.

–(+Number1, +Number2, ?Result)
Evaluates the difference Number1 – Number2 and unifies the resulting value with Result.

/(+Number1, +Number2, ?Result)
Evaluates the quotient Number1 / Number2 and unifies the resulting value with Result.

//(+Number1, +Number2, ?Result)
Evaluates the integer quotient Number1 // Number2 and unifies the resulting value with Result.

/\(+Number1, +Number2, ?Result)
Evaluates the bitwise conjunction Number1 /\ Number2 and unifies the resulting value with Result.

+Expr1 < +Expr2
Succeed if the value of Expr1 is less than the value of Expr2.

<<(+Number1, +Number2, ?Result)
Shifts Number1 left arithmetically by Number2 bits and unifies the resulting value with Result.

+Expr1 =:= +Expr2
Succeed if the value of Expr1 is equal to the value of Expr2.

+Expr1 =< +Expr2
Succeed if the value of Expr1 is less than or equal to the value of Expr2.

+Expr1 =\= +Expr2
Succeed if the value of Expr1 is not equal to the value of Expr2.

+Expr1 > +Expr2
Succeed if the value of Expr1 is greater than the value of Expr2.

+Expr1 >= +Expr2
Succeed if the value of Expr1 is greater than or equal to the value of Expr2.

>>(+Number1, +Number2, ?Result)
Shifts Number1 right arithmetically by Number2 bits and unifies the resulting value with Result.

\(+Number, ?Result)
Evaluates the bitwise complement of Number and unifies the resulting value with Result.

\/(+Number1, +Number2, ?Result)
Evaluates the bitwise disjunction Number1 \/ Number2 and unifies the resulting value with Result.

^(+Number1, +Number2, ?Result)
Evaluates the expression Number1 "to the power of" Number2 and unifies the resulting value with Result.

abs(+Number, ?Result)
Unifies the absolute value of Number with Result.

acos(+Number, ?Result)
Evaluates the trigonometric function acos(Number) and unifies the resulting value with Result.

asin(+Number, ?Result)
Evaluates the trigonometric function asin(Number) and unifies the resulting value with Result.

atan(+Number, ?Result)
Evaluates the trigonometric function atan(Number) and unifies the resulting value with Result.

between(+From, +To, +Step, ?Result)
Generate integer values between From and To with Step increment.

ceiling(+Number, ?Result)
Unifies Result with the least integral value that is greater than or equal to Number and of the same numeric type as Number.

clrbit(+Number, +Index, ?Result)
Result is Number with the Index'th bit cleared.

cos(+Number, ?Result)
Evaluates the trigonometric function cos(Number) and unifies the resulting value with Result.

denominator(+Number, ?Result)
Extracts the denominator of the rational Number and unifies the resulting integer with Result.

eval(+Expression, ?Result)
Used to evaluate eval/1 terms in arithmetic expressions. It is equivalent to Result is Expression, which should be preferred for direct use.

exp(+Number, ?Result)
Evaluates the exponential function exp(Number) ("e to the power of Number") and unifies the resulting value with Result.

fix(+Number, ?Result)
Unifies the integer part of Number with Result (Truncation towards zero).

float(+Number, ?Result)
Converts the integer Number to real and unifies the resulting value with Result.

floor(+Number, ?Result)
Unifies Result with the greatest integral value that is less or equal than Number and of the same numeric type as Number.

frandom(?F)
Generates a random floating–point number F in the range <0, 1>.

getbit(+Number, +Index, ?Result)
Result is the Index'th bit of Number.

?Result is +Expression
Evaluates the arithmetic expression Expression and unifies the resulting value with Result.

ln(+Number, ?Result)
Evaluates the natural logarithm ln(Number) and unifies the resulting value with Result.

max(+Number1, +Number2, ?Maximum)
Unifies the maximum of Number1 and Number2 with Maximum.

min(+Number1, +Number2, ?Minimum)
Unifies the minimum of Number1 and Number2 with Minimum.

mod(+Number1, +Number2, ?Result)
Evaluates the modulus Number1 mod Number2 and unifies the resulting value with Result.

numerator(+Number, ?Result)
Extracts the numerator of the rational Number and unifies the resulting integer with Result.

plus(?Add1, ?Add2, ?Sum)
Succeeds if Sum is the sum of integer arguments Add1 and Add2.

random(?N)
Generates a random integer N.

rational(+Number, ?Result)
Converts Number into a rational number and unifies it with Result.

round(+Number, ?Result)
Rounds Number to the nearest integral value of the same type and unifies the resulting real value with Result.

seed(+Seed)
Sets the initial seed Seed for generating random numbers with random/1 or frandom/1.

setbit(+Number, +Index, ?Result)
Result is Number with the Index'th bit set.

sgn(+Number, ?Result)
Unifies Result with the sign of Number which is either —-1, 0 or 1.

sin(+Number, ?Result)
Evaluates the trigonometric function sin(Number) and unifies the resulting value with Result.

sqrt(+Number, ?Result)
Evaluates the square root sqrt(Number) and unifies the resulting value with Result.

sum(+ExprList, ?Result)
Evaluates the the arithmetic expressions in ExprList and unifies their sum with Result.

tan(+Number, ?Result)
Evaluates the trigonometric function tan(Number) and unifies the resulting value with Result.

times(?Factor1, ?Factor2, ?Product)
Succeeds if Product is the result of multiplying integer arguments Factor1 and Factor2.

xor(+Number1, +Number2, ?Result)
Evaluates the bitwise exclusive disjunction Number1 xor Number2 and unifies the resulting value with Result.

1.3 Arrays and Global Variables

current_array(?Array, ?Options)
Succeeds if there exists an array as denoted by Array and with type and visibility as given in the list Options.

decval(+ElemSpec)
Decrements the contents of the visible array element or global variable ElemSpec by one.

erase_array(+ArraySpec)
Erases existing visible array or global variable specified by ArraySpec.

getval(+ElemSpec, ?Value)
Succeeds if the visible array element or global variable ElemSpec unifies with the value Value.

incval(+ElemSpec)
Increments the contents of the visible array element or global variable ElemSpec by one.

make_array(+Array)
Creates the untyped array or global variable Array.

make_array(+Array, +Type)
Creates the global array or global variable Array of type Type.

make_local_array(+Array)
Creates an array or global variable Array visible only in the caller module.

make_local_array(+Array, +Type)
Creates an array or global variable Array of type Type visible only in the caller module.

setval(+ElemSpec, ?Value)
Sets visible array element or global variable ElemSpec to the value Value.

1.4 Control

!
Cut – succeeds and removes all choice points between cut and parent goal.

+Goal1 , +Goal2
Comma (AND) operator – succeeds if the goals Goal1 and Goal2 both succeed

+Goal1 –> +Goal2
If..Then – succeeds if Goal1 succeeds, and then Goal2 succeeds.

+Goal1 –> +Goal2 ; +Goal3
If..Then..Else – succeeds if either Goal1 succeeds, and then Goal2 succeeds; or else if Goal1 fails, and then Goal3 succeeds.

–?–> ?Body
The matching operator. The head of the clause which contains it will not be unified with the caller, one–way matching will be used instead.

+DefModule : +Goal
Goal is executed using the predicate definition in DefModule rather than the visible one.

+Goal1 ; +Goal2
Semicolon (OR) operator – Succeeds if the goal Goal1 succeeds or if the goal Goal2 succeeds.

Goal @ ContextModule
Goal is executed in the calling context of ContextModule.

\+ +Goal
Succeeds if Goal cannot be satisfied. Uses negation as failure (synonym of not/1 and fail_if/1).

+Vars ∧ +Goal
Succeeds if Goal succeeds.

attach_suspensions(+Trigger, +Susps)
Insert the suspensions Susps into the suspension list of the symbolic trigger Trigger.

block(+Goal, ?Tag, +Recovery)
Similar to call(Goal) if Goal succeeds or fails. If an exit_block/1 is executed inside Goal, whose argument unifies with Tag, then Recovery is executed.

call(+Goal)
Succeeds if Goal succeeds.

call_priority(+Goal, +Priority)
Execute Goal with priority Priority.

current_suspension(–Susp)
Susp is a live (sleeping or scheduled) suspension.

+IterationSpecs do +Goals
Execute Goals iteratively according to IterationSpecs.

exit_block(?TagExit)
Continues the program at the recovery procedure of the block/3 predicate whose Tag argument unifies with TagExit.

fail
Does not succeed. A synonym of false/0.

false
Does not succeed (synonym of fail/0).

+IterationSpecs do +Goals
Execute Goals iteratively according to IterationSpecs.

+IterationSpecs do +Goals
Execute Goals iteratively according to IterationSpecs.

+IterationSpecs do +Goals
Execute Goals iteratively according to IterationSpecs.

+IterationSpecs do +Goals
Execute Goals iteratively according to IterationSpecs.

fork(+Max, ?I)
Succeeds for all integers I between 1 and Max. The solutions are generated in parallel.

+IterationSpecs do +Goals
Execute Goals iteratively according to IterationSpecs.

get_priority(–Priority)
Get the priority of the currently executing goal.

init_suspension_list(+Position, +Attribute)
Initialise the argument position Position within the structure Attribute with an empty suspension list.

kill_suspension(+Susp)
Kill the suspended goal represented by Susp, i.e. treat it as if it had been woken.

make_suspension(+Goal, +Prio, –Susp)
Make Goal a suspended goal with waking priority Prio and return the corresponding suspension object in Susp.

make_suspension(+Goal, +Prio, –Susp, +Module)
Make Goal from module Module a suspended goal with waking priority Prio and return the corresponding suspension object in Susp.

merge_suspension_lists(+Pos1, +Attr1, +Pos2, +Attr2)
Destructively merge the suspension list on Pos1 in structure Attr1 into the suspension list on Pos2 in structure Attr2.

mutex(+MutexId, +Goal)
Equivalent to once(Goal) but with mutual exclusion among parallel workers.

not +Goal
Succeeds if Goal cannot be satisfied (uses negation as failure).

notify_constrained(−AttrVar)
Notify the system that the attributed variable was constrained.

once +Goal
Succeeds if Goal succeeds, and removes all its alternatives —— equivalent to call((Goal, !))

repeat
Succeeds as often as tried.

schedule_suspensions(+Trigger)
Take the suspension list associated with the symbolic trigger Trigger and schedule them for execution.

schedule_suspensions(+Position, +Attribute)
Take the suspension list on argument position Position within Attribute, and schedule them for execution.

set_suspension_data(+Susp, +Name, +Value)
Modify properties of suspended goals.

subcall(+Goal, ?Delayed_goals)
Succeeds iff Goal succeeds and unifies Delayed_goals with a list of remaining delayed goals.

suspend(+Goal, +Prio, +CondList)
Delay the Goal and wake it with priority Prio as soon as one of the specifications in CondList occurs.

suspend(+Goal, +Prio, +CondList, −Susp)
Delay the Goal as suspension Susp and wake it with priority Prio as soon as one of the specifications in CondList occurs.

suspensions(−Susps)
Returns a list of all currently live (sleeping or scheduled) suspensions.

true
Succeeds always.

wake
Execute all scheduled suspensions whose priorities are higher than the current one.

?Goal
The sound negation operator. If Goal is not ground, the predicate delays.

1.5 Coroutining

coroutine
Switches on the coroutine flag, equivalent to set_flag(coroutine, on).

delayed_goals(?GoalList)
Succeeds if GoalList is the list of all goals currently delayed.

delayed_goals(?Var, ?GoalList)
Succeeds if GoalList is the list of all goals delayed by the variable Var.

delayed_goals_number(?Var, ?Number)
Succeeds if Number is the number of goals delayed by the variable Var.

1.6 Predicate Database and Compiler

[+File_1, ..., +File_N]
Compile file or list of files —— shorthand for compile/1.

abolish +SpecList
Remove the definition and/or declaration of the predicates specified in SpecList.

als(+PredSpec)
Outputs the abstract code for the compiled predicate PredSpec.

assert(+Clause)
Add specified clause at the end of the dynamic procedure to the database.

asserta(+Clause)
Add specified clause for a dynamic procedure to the database before any existing clauses.

clause(+Clause)
Succeeds if Clause unifies with a clause of a dynamic procedure.

clause(+Head, ?Body)
Succeeds if Head :– Body is an existing dynamic clause.

compile(+File)
Compile specified file or list of files File.

compile(+File,+Module)
Compiles specified file or list of files File into the specified module Module.

compile_stream(+Stream)
Compile the given stream Stream.

compile_term(+Clause)
Compile specified clause or list of clauses Clause.

compiled_stream(?Stream)
Succeeds if the I/O stream currently being compiled is Stream.

current_built_in(?PredSpec)
Succeeds if the predicate defined by PredSpec is a visible built–in predicate.

current_predicate(?PredSpec)
Succeeds if PredSpec is a visible predicate defined by the user, or a visible library predicate.

demon +SpecList
Declares the procedure(s) specified by SpecList to be demons.

dump(+File)
Dump the object form of the source program in the file File into a file File.sd.

dynamic +SpecList
Declares the procedures specified by SpecList as dynamic.

ensure_loaded(+Files)
Compile the specified Files if necessary.

get_flag(+PredSpec, ?Flag, ?Value)
Succeeds if the flag Flag of the procedure specified by PredSpec has the value Value.

inline(+Pred, +TransPred)
Declares TransPred as the predicate to be used to do compile–time transformation (e.g. inlining) of calls to Pred.

is_built_in(+PredSpec)
Succeeds if PredSpec is a system built–in predicate.

is_dynamic(+PredSpec)
Succeeds if the procedure specified by PredSpec has been declared as dynamic.

is_predicate(+PredSpec)
Succeeds if PredSpec is a defined predicate.

lib(+LibraryName)
Makes the library LibraryName available in the current module if not loaded already.

lib(+LibraryName,+ModuleName)
The library LibraryName is loaded into the module ModuleName if not loaded already.

listing
Outputs the definition of all dynamic predicates in the database.

listing +SpecList
Outputs the definition of all dynamic predicates indicated by SpecList.

mode +PredModes
Specifies the mode (calling pattern) for the given predicates.

parallel +SpecList
Declares the procedure(s) specified by SpecList as parallel.

retract(+Clause)
Succeeds if a clause that unifies with Clause can be removed from the database.

retract_all(+Head)
Succeeds if all the clauses whose heads match Head are successfully removed from the database.

set_flag(+PredSpec, +Flag, +Value)
Sets the flag Flag of the procedure specified by PredSpec to the value Value.

1.7 Debugger

dbgcomp
Tells the compiler to generate code with debug instructions. Equivalent to the call to set_flag(debug_compile, on).

debug
Execute subsequent top–level queries with the debugger on.

debug(+Goal)
Execute the goal Goal with the debugger in leap mode.

debugging
Information about the current top–level tracing mode and spied procedures is printed on the current output.

get_leash(?Port, ?Leash)
Currently not supported.

kill_display_matrix(+Name)
Destroys an existing display matrix.

make_display_matrix(+Terms, +Name)
Creates a display matrix of terms that can be monitored with the graphical ECLiPSe (the predicate will succeed siliently in tty ECLiPSe).

make_display_matrix(+Terms, +Prio, +Type, +CondList, +Name)
Creates a display matrix of terms that can be monitored with the graphical ECLiPSe (the predicate will succeed siliently in tty ECLiPSe).

nodbgcomp
Tells the compiler to generate code without debug instructions and not to retain the source variable names.

nodebug
Switch the debugger off for subsequent top–level queries.

nospy ?Speclist
All spypoints are removed from all the procedures given by SpecList. If SpecList is a variable, remove all spypoints.

notrace
Switch the debugger off for subsequent top–level queries.

set_leash(?Ports, +Leash)
Currently not supported.

skipped SpecList
Declares all the procedures given in SpecList as skipped.

spy SpecList
Sets a spypoint for the procedure(s) in SpecList.

spy_term(?Term, +Cond)
Create a SPYTERM–port in the debugger and prepare for tracing modifications to Term as MODIFY–ports.

spy_var(?Var)
Create a SPYTERM–port in the debugger and prepare for tracing modifications to Var as MODIFY–ports.

trace
Execute subsequent top–level queries with the debugger on.

trace(+Goal)
Execute the goal Goal with the debugger in creep mode.

trace_port(?Invoc, +Port, ?Term)
Create a user–defined debugger port.

traceable SpecList
Sets the procedures in SpecList to be traceable.

unskipped SpecList
Sets the procedures specified in SpecList to be unskipped or not skipped.

untraceable +SpecList
Declares the procedure(s) in SpecList to be untraceable.

1.8 Directives

pragma(Option)
Enable or disable compiler options. Can only be used as a directive in source files.

1.9 Environment

abolish_op(+Name, +Associativity)
Remove the declaration of the visible operator +Name of associativity Associativity.

abort
The current computation is aborted and control is returned to the top level.

break
A new invocation of the top level loop is called as a subgoal.

current_atom(?Atom)
Succeeds if Atom is an atom in the system.

current_compiled_file(?File, ?Time, ?Module)
Succeeds if File is a file that has been compiled into the system.

current_functor(?PredSpec)
Succeeds if PredSpec is a functor known to the system.

current_macro(?TermClass, ?TransPred, ?Options, ?Module)
Succeeds if TermClass is a macro with the transformation predicate TransPred defined in module Module and flags Options.

current_op(?Precedence, ?Associativity, ?Name)
Succeeds if Name is a visible operator with precedence Precedence and associativity Associativity.

current_struct(?Struct)
Succeeds if Struct is a currently visible structure specification.

define_macro(+TermClass, +TransPred, +Options)
Defines a macro transformation for the functor or type specified by TermClass. The transformation predicate is TransPred and Options is a list of options.

env
Prints information about the current ECLiPSe environment on the current output.

erase_macro(+TransTerm)
Erases the macro definition for TransTerm done in the current module

erase_macro(+TransTerm, +Options)
Erases the macro definition for TransTerm done in the current module

garbage_collect
Request an immediate garbage collection.

get_chtab(+Char, ?Class)
Succeeds if the lexical class of character Char is Class.

get_flag(?Flag, ?Value)
Succeeds if the flag Flag has the value Value.

help
Prints general help information on the current output.

help(+PredSpec)
Prints help information on the specified built–ins in PredSpec on the current output.

make
Recompile all files that have been modified since their last compilation.

op(+Precedence, +Associativity, +Name)
Defines the operator(s) Name with precedence Precedence and associativity Associativity locally in the current module. Defining a local operator of precedence 0 hides a global operator of the same associativity group (postfix, infix or prefix).

phrase(+Grammar, ?List)
Succeeds if List unifies with a list from the specified grammar Grammar.

phrase(+Grammar, ?Tokens, ?Remainder)
Succeeds if Tokens can be parsed as part of the grammar defined in Grammar and Remainder contains any remaining terms in Tokens.

pred(+PredSpec)
The flags of the predicate described by PredSpec are printed out on the current output.

profile(+Goal)
Profile the goal and print the times spent in each predicate.

profile(+Goal, +Flags)
Profile the goal and print the times spent in each predicate.

set_chtab(+Char, +Class)
Sets the lexical class of character Char to class Class, this provides an interface to ECLiPSe 's lexical analyser.

set_flag(+Flag, +Value)
Succeeds if the flag Flag is successfully set to the value Value.

statistics
Prints information about resource usage and garbage collection.

statistics(?Keyword, ?Value)
Succeeds if the statistics item Keyword has value Value.

local struct(++Prototype), export struct(++Prototype), global struct(++Prototype)

Declare a structure according to Prototype.

1.10 Event Handling

cancel_after_event(+Event)
Cancel all pending instances of after event Event.

current_after_event(+Event)
Check if Event is a currently pending after event.

current_error(?N)
Succeeds if N unifies with a valid error number.

current_interrupt(?N, ?IntID)
Succeeds if N unifies with the number and IntID unifies with the mnemonic name of a valid interrupt.

error(+Number, ?Culprit)
An error number Number is raised and the corresponding error handler is executed.

error(+Number, ?Culprit, +Module)
An error Number is raised with Culprit (in module Module) the culprit, and the corresponding error handler is executed.

error_id(+N, ?Message)
Succeeds if Message unifies with the error message string defined for error number N.

event(+EventName)
The event EventName is raised and the corresponding error handler is executed.

event_after(+Even, +Time)
Set up an event Event which is triggered after Time seconds have elapsed.

event_after_every(+Even, +Time)
Set up an event Event which is triggered after every Time seconds have elapsed.

get_error_handler(+Number, ?PredSpec, ?Module)
Returns the error_handler for error number Number and its home module Module.

get_interrupt_flag(?Int_Id, ?Mode)
Suceed if Int_id unifies with a valid interrupt identifier and Mode unifies with its debugging mode.

get_interrupt_handler(+Number, ?PredSpec, ?Module)
Succeeds if PredSpec unifies with the specification of the current handler for interrupt Number and Module unifies with its home module.

reset_error_handlers
All error handlers are reset, cancelling any redefinition.

reset_event_handler(+EventName)
Resets the handler for error number Number to its default value.

reset_interrupt_handler(+Number)
Resets an interrupt handler for interrupt Number to its default value, cancelling a previous redefinition.

reset_interrupt_handlers
All interrupt handlers are reset, cancelling any redefinition.

set_event_handler(+EventName, +PredSpec)
Set an event handler PredSpec for the event EventName.

set_interrupt_flag(?Int_Id, +Mode)
Set the debugger mode Mode for the interrupt Int_Id

set_interrupt_handler(+Number, +PredSpec)
Sets an interrupt handler PredSpec for the interrupt with number Number.

1.11 External Interface

b_external(+PredSpec)
Declares PredSpec to be a non–deterministic external predicate.

b_external(+PredSpec, +CName)
Defines PredSpec to be a nondeterministic external predicate linked to the C function whose system name is CName.

call_c(+Function, ?Code)
Invoke the C function Function and unify its return code with Code.

external(+PredSpec)
Declares PredSpec to be a deterministic external predicate.

external(+PredSpec, +CName)
Defines PredSpec to be a deterministic external predicate linked to the C function whose system name is CName.

load(+File)
The object code or loadable library File is loaded into the running system.

xget(+Handle, +Index, –Value)
Get the Index–th field of an external data structure (referenced by Handle).

xset(+Handle, +Index, +Value)
Set the Index–th field of an external data structure (referenced by Handle) to Value.

yield(+ToC, –FromC)
Yield control to the C/C++ main program that has invoked ECLiPSe. The arguments are used for passing data in and out.

1.12 Character I/O

get(?Ascii)
Reads the next character from the current input stream and unifies its ASCII code with Ascii.

get(+Stream, ?Ascii)
Reads the next character from the input stream Stream and unifies its ASCII code with Ascii.

get_char(?Char)
Reads the next character from the current input and unifies it with a single character string Char.

get_char(+Stream, ?Char)
Reads the next character from the input stream Stream and unifies it with a single character string Char.

put(+Ascii)
The character represented by the ascii integer Ascii is put onto the buffered current output.

put(+Stream, +Ascii)
The character represented by the ascii integer code Ascii is put onto the buffered output stream Stream.

put_char(+Char)
Puts the single character string Char onto the buffered current output.

put_char(+Stream, +Char)
Puts the single character string Char onto the buffered output stream Stream.

read_string(+Delimiters, ?Length, ?String)
Reads a string from the input stream up to a delimiter or up to a specified length.

read_string(+Stream, +Delimiters, ?Length, ?String)
Reads a string from the stream Stream up to a delimiter or up to a specified length.

tyi(?Ascii)
Succeeds if the ascii code of the next character read in raw mode from the current input is successfully unified with Ascii.

tyi(+Stream, ?Ascii)
Succeeds if the ascii code of the next character read in raw mode from the input stream Stream is successfully unified with Ascii.

tyo(+Ascii)
The character represented by the ascii integer Ascii is put onto the current output in raw mode.

tyo(+Stream, +Ascii)
The character represented by the ascii integer Ascii is put onto the output stream Stream in raw mode.

1.13 Stream I/O

accept(+Stream, ?From, ?NewStream)
Accepts a connection for a stream socket and creates a new socket which can be used for I/O.

at(+Stream, ?Pointer)
Succeeds if Pointer is the pointer position of the stream Stream.

at_eof(+Stream)
Succeeds if the position of the pointer to stream Stream is at the end of file.

bind(+Stream, ?Address)
Associates an address with a given socket stream.

close(+Stream)
Closes the stream specified by Stream.

connect(+Stream, +Address)
Connects a socket with the given address.

current_stream(?Stream)
Succeeds if Stream is a currently open stream.

current_stream(?Name, ?Mode, ?Stream)
Succeeds if there is currently an open stream Stream open to the file Name in the mode Mode. This predicate is obsolete, use current_stream/1 and get_stream_info/3 instead.

flush(+Stream)
Flushes the output stream Stream.

get_prompt(+InStream, ?Prompt, ?OutStream)
Succeeds if the prompt for the stream InStream is Prompt and is written to the stream OutStream.

get_stream(+StreamId, ?Stream)
Succeeds if Stream is the stream to which the stream StreamId is assigned.

get_stream_info(+Stream, ?Attr, ?Value)
Succeeds if the attribute Attr of the open stream Stream has the value Value.

listen(+Stream, +Queue)
Specifies how many connections are accepted for a socket and makes connections available.

open(+SourceSink, +Mode, ?Stream)
Opens the I/O source or sink SourceSink in mode Mode and associates it with the stream identifier Stream.

open(+SourceSink, +Mode, ?Stream, +Options)
Opens the I/O source or sink SourceSink in mode Mode and associates it with the stream identifier Stream.

pipe(?StreamIn, ?StreamOut)
Creates a pipe and two streams StreamIn and StreamOut to its read and write ends.

seek(+Stream, +Offset)
The pointer in stream Stream is offset Offset from the start of the file.

select(+StreamList, +Timeout, ?ReadyStreams)
Returns streams from StreamList which are ready for I/O, blocking at most Timeout seconds.

set_prompt(+InStream, +Prompt, +OutStream)
The prompt Prompt is output on the stream OutStream for input to the input stream InStream.

set_stream(+StreamId, +Stream)
The symbolic stream name StreamId is associated with the stream Stream.

socket(+Domain, +Type, ?Stream)
Creates a socket of a given type and domain and associates a stream with it.

1.14 Term I/O

display(+Term)
Term is displayed on the current output —— without considering operator definitions.

display(+Stream, +Term)
Term is displayed on the output stream Stream —— without considering operator definitions.

nl
A newline is printed on the output stream.

nl(+Stream)
A newline is printed on the output stream Stream.

print(?Term)
The term Term is written on the output stream according to the current operator declarations, using the predicate portray/2 or portray/1 if it exists.

print(+Stream, ?Term)
The term Term is written on the output stream Stream according to the current operator declarations, using the predicate portray/2 or portray/1 if it exists.

printf(+Format, ?ArgList)
The arguments in the argument list ArgList are interpreted according to the Format string and the result is printed to the output stream.

printf(+Stream, +Format, ?ArgList)
The arguments in the argument list ArgList are interpreted according to the Format string and the result is printed on the output Stream.

read(?Term)
Succeeds if the next term from the input stream is successfully read and unified with Term.

read(+Stream, ?Term)
Succeeds if the next term from the input stream Stream is successfully read and unified with Term.

read_exdr(+Stream, –Term)
A term in EXDR–format is read from the input stream Stream and converted to the corresponding ECLiPSe term Term.

read_token(?Token, ?Class)
Succeeds if the next token from the current input stream is successfully read and unified with Token and its token class with Class.

read_token(+Stream, ?Token, ?Class)
Succeeds if the next token from the input stream Stream is successfully read and unified with Token and its token class with Class.

readvar(+Stream, ?Term, –VarList)
Succeeds if the next Prolog term from the input stream Stream is successfully read and unified with Term, and any variables in Term are collected in the list VarList, together with their names.

write(?Term)
The term Term is written on output stream according to the current operator declarations.

write(+Stream, ?Term)
The term Term is written on the output stream Stream according to the current operator declarations.

write_canonical(?Term)
The term Term is written on the stream output in a form that ignores operator declarations and can be read in.

write_canonical(+Stream, ?Term)
The term Term is written on the output stream Stream in a form that ignores operator declarations and can be read in.

write_exdr(+Stream, +Term)
The term Term is written onto the output stream Stream in EXDR–format (a format for communication with agents in other programming languages).

writeclause(+Clause)
The clause Clause is pretty printed on the current output .

writeclause(+Stream, +Clause)
The clause Clause is pretty printed on the output stream Stream .

writeln(?Term)
The term Term is written on the current output according to the current operator declarations. Equivalent to write(Term),nl.

writeln(+Stream, ?Term)
The term Term is written on the output stream Stream according to the current operator declarations. Equivalent to write(Stream,Term), nl(Stream).

writeq(?Term)
The term Term is written on the current output in a form that can be read in.

writeq(+Stream, ?Term)
The term Term is written on the output stream Stream in a form that can be read in.

1.15 List Utilities Library

append(?List1, ?List2, ?List3)
Succeeds if List3 is the result of appending List2 to List1.

checklist(+Pred, +List)
Succeeds if Pred(Elem) succeeds for every element of List.

delete(?Element, ?List1, ?List2)
Succeeds if List2 is List1 less an occurence of Element in List1.

flatten(+NestedList, ?FlatList)
Succeeds if FlatList is the list of all elements in NestedList, as found in a left–to–right, depth–first traversal of NestedList.

intersection(+List1, +List2, ?Common)
Succeeds if Common unifies with the list which contains the common elements of List1 and List2.

length(?List, ?N)
Succeeds if the length of list List is N.

maplist(+Pred, ?OldList, ?NewList)
Succeeds if Pred(Old, New) succeeds for corresponding pairs of elements from OldList and NewList.

member(?Term, ?List)
Succeeds if Term unifies with a member of the list List.

memberchk(+Term, ?List)
Succeeds if Term is a member of the list List.

nonmember(+Element, +List)
Succeeds if Element is not an element of the list List.

reverse(+List, ?Reversed)
Succeeds if Reversed is the reversed list List.

subset(?SubList, +List)
Succeeds if List is the list which contains all elements from SubList in the same order as in SubList.

subtract(+List1, +List2, ?Remainder)
Succeeds if Remainder is the list which contains those elements of List1 which are not in List2.

union(+List1, +List2, ?Union)
Succeeds if Union is the list which contains the union of elements in List1 and those in List2.

1.16 Sorting

keysort(+List1, ?List2)
Succeeds if List2 is a sorted list version of List1, whose elements are of the form Key–Value. The sort is done according to the value of the key Key.

merge(+List1, +List2, ?List3)
Succeeds if List3 is a merged list of List1 and List2. If both lists are sorted, List3 will be sorted.

merge(+Key, +Order, +List1, +List2, ?List3)
Succeeds if List3 is a merged list of List1 and List2. If both lists are sorted, List3 will be sorted. The sort is done according to the Key and Order specifications.

msort(+List1, ?List2)
Succeeds if List2 has the same elements as List1 and is sorted.

prune_instances(+List, ?PrunedList)
Succeeds if PrunedList is the smallest list that subsumes the list List.

sort(+List1, ?List2)
Succeeds if List2 is the strictly ordered, no duplicates version of List1.

sort(+Key, +Order, +Random, ?Sorted)
Succeeds if Sorted is the sorted list version of Random. The sort is done according to the Key and Order specifications.

1.17 Modules

autoload(+Library, +ListOfPredSpec)
Declares the predicates in ListOfPredSpec to be autoloading from the module Library, which is in the file Library.pl in one of the library directories.

autoload_tool(+Library, +ListOfPredSpec)
Declares the predicates in ListOfPredSpec to be autoloading tools from the module (file) Library.pl.

begin_module(+Module)
Start the definition of the body of the Module.

create_module(+Module)
Create the given module Module.

current_module(?Module)
Succeeds if Module is an existing module.

erase_module(+Module)
Erase the given module Module.

export +SpecList
Exports all procedures specified by SpecList. These are then visible to modules that import them.

global +SpecList
Declares the procedure(s) and other modular items specified by SpecList to be global.

import +Module
Declare all the exported procedure of Module as being imported by default to the caller module.

import +SpecList from +Module
Declares the procedure(s) specified by SpecList to be imported from the module Module.

is_locked(+Module)
Succeeds if the module Module is locked.

local +SpecList
Declares the procedure(s) and other modular items specified by SpecList to be local to the caller module. If a global procedure of the same name is defined in another module it is made invisible.

lock(+Module)
Locks the access to the module Module.

lock(+Module, +Password)
Locks the access to the module Module, but allow unlocking with the password Password

module(+Module)
Open and enter a module, ie. set Module as the home module for the top–level loop and for subsequent compiled predicates.

module_interface(+Module)
Create the module Module and start defining its interface.

tool(+PredSpec)
Declares PredSpec as a tool interface procedure.

tool(+PredSpecI, +PredSpecB)
Declares PredSpecI as a tool interface procedure and PredSpecB as its body procedure.

tool_body(+PredSpecI, ?PredSpecB, ?Module)
Succeeds if PredSpecI is a tool interface procedure, PredSpecB is its body procedure, and Module the module where it is defined.

unlock(+Module, +Password)
Unlocks the access to the module Module, if the password given in Password is correct

use_module(+Module)
Make available the interface of Module.

1.18 Operating System

absolute_file_name(+RelName, ?FullName)
Expands a relative or symbolic file name into a full file path name.

alarm(+Clock)
The operating system will send the signal 14 to the ECLiPSe system Clock seconds after the command was entered.

argc(?Number)
Succeeds if Number is the number of arguments given on the command line to invoke ECLiPSe .

argv(+N, ?Argument)
Succeeds if the Nth argument given on the command line when invoking ECLiPSe is the string Argument.

cd(+Directory)
Directory is made the current working directory. Equivalent to set_flag(cwd, Directory).

cputime(?Time)
Succeeds if Time is the elapsed user cpu time in seconds.

date(?Date)
Date is bound to the string giving the date and time.

delete(+File)
The file File (with absolute or relative pathname) is deleted.

errno_id(–Message)
Message is bound to the message string that corresponds to most recent operating system error that occurred during the execution.

exec(+Command, ?Streams)
A child process Command is forked, its standard streams are connected to Streams and the ECLiPSe process waits until it terminates.

exec(+Command, ?Streams, ?Pid)
A child process Command is forked, its standard streams are connected to Streams and its process ID is Pid.

exec_group(+Command, ?Streams, ?Pid)
A child process Command is forked in a new process group, its standard streams are connected to Streams and its process ID is Pid.

exists(+File)
Succeeds if the file File (with absolute or relative pathname) exists. File can be a directory.

exit(+Status)
The Prolog session is ended and the specified status returned to the operating system.

get_file_info(+File, ?Attr, ?Value)
Succeeds if the file File (with absolute or relative pathname) exists and the value of its attribute Attr unfies with Value.

getcwd(?Cwd)
Unifies the current working directory with Cwd.

getenv(+EnvVar, ?Value)
Succeeds if the string EnvVar corresponds to a variable set in the UNIX environment, and it unifies Value with the string which is the value of that variable.

halt
The Prolog session is ended (the same as Control–D at the top–level loop).

kill(+Pid, +Signal)
Send the signal Signal to the process number Pid

mkdir(+Path)
Create a directory/folder with the given name.

os_file_name(?InternalName, ?ExternalName)
Conversion between internal ECLiPSe file name and Operating System file name syntax.

pathname(+FilePath, ?Path)
Succeeds if the pathname FilePath, if stripped of its file name, gives Path, the path up to the parent directory of the file.

pathname(+FilePath, ?Path, ?Name)
If FilePath is a pathname of a file, succeed if Path is the pathname of its parent directory and Name is the local file name.

pause
The ECLiPSe session is suspended (the same as Control–Z at the top–level loop).

read_directory(+Directory, +Pattern, ?SubdirList, ?FileList)
Unifies SubdirList with a list of subdirectories and FileList with a list of matching files in the specified directory.

rename(+OldName, +NewName)
Renames the file OldName to NewName.

sh(+ShellCommand)
The string or atom ShellCommand is passed as a command to the operating system, and the command is executed there (it is a synonym for system/1).

sleep(+Seconds)
The execution of ECLiPSe is suspended for Seconds number of seconds.

suffix(+FileName, ?Suffix)
Succeeds if the string Suffix is the extension part of the input string FileName.

system(+ShellCommand)
The string or atom ShellCommand is passed as a command to the operating system, and the command is executed there (it is a synonym for sh/1).

wait(?Pid, ?Status)
Wait for a child process to terminate and report its process ID and status.

1.19 Recorded Database

abolish_record(+Key)
Remove the local record Key and all its recorded values.

current_record(?Key)
Succeeds if Key is the key of a recorded item.

erase(+DBRef)
Succeeds if the database reference DBRef designates a term recorded in the indexed database and this term is successfully erased.

erase(+Key, ?Value)
Succeeds if the term Value associated with key Key is removed as an entry in the indexed database.

erase_all(+Key)
All the the values associated with key Key are removed from the indexed database.

is_record(+Key)
Succeeds if Key is a key of a recorded item.

local_record(+Key)
Declare the record with key Key to be local to the caller module

record(+Key, ?Value)
Records the term Value at the end of key Key in the indexed database.

recorda(+Key, ?Value)
Records the term Value at the beginning of key Key in the indexed database.

recorda(+Key, ?Value, –DBRef)
Records the term Value at the beginning of key Key in the indexed database.

recorded(+Key, ?Value)
Succeeds if the term Value has been recorded in the indexed database under the key Key.

recorded(+Key, ?Value, –DBRef)
Succeeds if the term Value has been recorded in the indexed database under the key Key and DBRef is its unique reference.

recorded_list(+Key, ?List)
Succeeds if the List is the list of all terms that are currently recorded in the indexed database under the key Key.

recordz(+Key, ?Value)
Records the term Value at the end of key Key in the indexed database.

recordz(+Key, ?Value, –DBRef)
Records the term Value at the end of key Key in the indexed database.

referenced_record(+DBRef, ?Value)
Succeeds if DBRef is the database reference of a currently recorded term and this term unifies with Value

rerecord(+Key, ?Value)
Erases all entries recorded under the key Key and replaces them with the given value Value.

1.20 Strings and Atoms

append_strings(?String1, ?String2, ?String3)
Succeeds if String3 is the concatenation of String1 and String2.

atom_length(+Atom, ?Length)
Succeeds if Length is the length of Atom.

concat_atom(+List, ?Dest)
Succeeds if Dest is the concatenation of the atomic terms contained in List. It is more efficient to use concat_string/2 whenever possible.

concat_atoms(+Src1, +Src2, ?Dest)
Succeeds if Dest is the concatenation of Src1 and Src2. It is more efficient to use concat_strings/3 whenever possible.

concat_string(+List, ?Dest)
Succeeds if Dest is the concatenation of the atomic terms contained in List.

concat_strings(+Src1, +Src2, ?Dest)
Succeeds if Dest is the concatenation of Src1 and Src2.

join_string(+List, +Glue, ?String)
String is the string formed by concatenating the elements of List with an instance of Glue beween each of them.

split_string(+String, +SepChars, +PadChars, ?SubStrings)
Decompose String into SubStrings according to separators SepChars and padding characters PadChars.

string_length(+String, ?Length)
Succeeds if Length is the length of the string String.

substring(+String1, +String2, ?Position)
Succeeds if String2 is a substring of String1 beginning at position Position.

substring(+String1, ?Position, ?Length, ?String2)
Succeeds if String2 is the substring of String1 starting at position Position and of length Length.

1.21 Term Comparisons

?Term1 = ?Term2
Succeeds if Term1 and Term2 unify.

?Term1 == ?Term2
Succeeds if Term1 and Term2 are identical terms.

?Term1 @< ?Term2
Succeeds if term Term1 is before term Term2 in the standard ordering.

?Term1 @=< ?Term2
Succeeds if term Term1 is before or equal to Term2 in the standard ordering.

?Term1 @> ?Term2
Succeeds if term Term1 is after term Term2 in the standard ordering.

?Term1 @>= ?Term2
Succeeds if term Term1 is after or equal to Term2 in the standard ordering.

?Term1 \= ?Term2
Succeeds if Term1 and Term2 are not unifiable.

?Term1 \== ?Term2
Succeeds if Term1 and Term2 are not identical terms.

compare(?Ordering, ?Term1, ?Term2)
Succeeds if Ordering is a special atom which describes the ordering between Term1 and Term2.

compare_instances(?Relationship, ?Term1, ?Term2)
Succeeds if Relationship is an atom describing the instance relationship between Term1 and Term2.

instance(?Instance, ?Term)
Succeeds if Instance is an instance of Term.

not_unify(?Term1, ?Term2)
Succeeds if Term1 and Term2 are not unifiable.

occurs(?Simple, ?Term)
Succeeds if Simple is a variable or an atomic type that occurs in the term Term.

variant(?Term1, ?Term2)
Succeeds if Term1 is a variant of Term2.

?Term1 = ?Term2
The sound difference operator. Succeeds if the two terms cannot be unified, fails if they are identical, otherwise it delays.

1.22 Term Manipulation

'C'(?Input, ?Token, ?Rest)
Specifies how DCG grammar rules get a token from their input.

?Term =.. ?List
Univ —— Succeeds if List is the list which has Term's functor as its first element and Term's arguments, if any, as its successive elements.

add_attribute(?Var, ?Attribute)
Add dynamically an attribute to a variable.

add_attribute(?Var, ?Attribute, +Module)
Add dynamically an attribute to a variable.

arg(+N, +Term, ?Arg)
Succeeds if Arg is the Nth argument of the compound term Term.

atom_string(?Atom, ?String)
Conversion between an atom and a string.

bytes_to_term(+String, –Term)
Converts String, which is supposed to be an encoding of a term, into Term.

char_code(?Char, ?Code)
Succeeds if Code is the numeric character code of the character Char.

char_int(?Char, ?Integer)
Succeeds if Integer is the ASCII code of the one–character string Char.

copy_term(+OldTerm, ?NewTerm)
A copy of OldTerm with new variables is created and unified with NewTerm.

copy_term(+OldTerm, ?NewTerm, ?MetaTerms)
A copy of OldTerm with new variables is created and unified with NewTerm. MetaTerms is a list mapping the metaterms in OldTerm to the corresponding variables in NewTerm.

copy_term_vars(+Vars, +OldTerm, ?NewTerm)
NewTerm gets unified with a variant of OldTerm where all occurrences of variables in Vars are replaced by fresh variables.

dim(?Term, ?Dimensions)
Creates a multi–dimensional array in the form of nested structures, or computes the dimensions of an existing matrix.

functor(?Term, ?Functor, ?Arity)
Succeeds if the compound term Term has functor Functor and arity Arity or if Term and Functor are atomic and equal, and Arity is 0.

insert_suspension(?Term, +Susp, +Index)
Insert the suspension Susp into the Index'th suspension list of the current module's attribute for all metaterms that occur in Term.

insert_suspension(?Term, +Susp, +Index, +Module)
Insert the suspension Susp into the Index'th suspension list of the attribute Module for all metaterms that occur in Term.

integer_atom(?Integer, ?Atom)
Conversion between an integer and an atom. It is more efficient to use number_string/2 wherever possible.

meta_attribute(+Name, +Handlers)
Declares the variable attribute Name with the corresponding handlers.

name(?Atomnumber, ?List)
Succeeds if List is the corresponding list of ASCII codes for the atom or number Atomnumber.

nonground(?Term, –Variable)
Succeeds if Term is not a ground term and binds Variable to one of the variables inside Term.

nonground(+N, ?Term, ?VarList)
Succeeds if Term contains at least N different variables, and returns N of them in the list VarList.

number_string(?Integer, ?String)
Conversion between any number and a string.

setarg(+N, +Term, ?Arg)
Destructively replaces the Nth argument of the compound term Term with the term Arg.

string_list(?String, ?List)
Succeeds if List is a list whose elements are the ascii codes of the characters of String.

subscript(+Term, +Subscript, ?Elem)
Accesses the subterm Elem of Term, as specified by Subscript.

suspension_to_goal(+Susp, ?Goal, ?Module)
Succeeds for an unwoken suspension and returns the corresponding Goal structure and caller module.

term_string(?Term, ?String)
Conversion between a Prolog term and a string.

term_to_bytes(?Term, –String)
String is a ground encoding of Term, suitable for writing to a file, transmitting over a network etc.

term_variables(?Term, ?VarList)
Succeeds if VarList is the list of all variables in Term.

1.23 Type Testing

atom(?Atom)
Succeeds if Atom is a Prolog atom.

atomic(?Atomicterm)
Succeeds if Atomicterm is an atom, a number, or a string.

compound(?Term)
Succeeds if Term is of type compound, i.e. a structure or a list.

free(?Var)
Succeeds if Var is a free variable, not a metaterm.

get_suspension_data(+Susp, +Name, -Value)
Access properties of suspended goals.

get_var_info(?Var, ?InfoName, ?Value)
Succeeds if Var is an uninstantiated variable, InfoName is a valid information name and the information value Value unifies with the value of the information.

integer(?Integer)
Succeeds if Integer is an integer number.

is_handle(?Term)
Succeeds if Term is an external data handle.

is_suspension(?Term)
Succeeds if Term is a sleeping suspension.

meta(?Var)
Succeeds if Var is a metaterm.

nonground(?VarTerm)
Succeeds if VarTerm is not ground.

nonvar(?Term)
Succeeds if Term is instantiated.

number(?Number)
Succeeds if Number is a number.

rational(?Rational)
Succeeds if Rational is a rational number.

real(?Real)
Succeeds if Real is a real number.

string(?String)
Succeeds if String is a string.

type_of(?Term, ?Type)
Succeeds if Type is the data type of the term Term.

var(?Var)

Succeeds if Var is a variable or a metaterm.

Index