Welcome and Update Notes

WIN-PROLOG 7.0

Brian D Steel
WIN-PROLOG Welcome and Update Notes

The contents of this manual describe the product, BDS-PROLOG for Windows (hereinafter referred to as WIN-PROLOG), version 7.0, and are believed correct at time of going to press. They do not embody a commitment on the part of Brian D Steel (BDS), who may from time to time make changes to the specification of the product, in line with his policy of continual improvement. No part of this manual may be reproduced or transmitted in any form, electronic or mechanical, for any purpose without the prior written agreement of BDS.

Copyright (c) 1989-2019 Brian D Steel

Designed and Written by Brian D Steel

The "wallpaper" used in the screen shots in this publication is based on the Willow Boughs design by William Morris (1834-96)

Brian D Steel
Fairbanks
Alfold Road
Cranleigh
Surrey
GU6 8NB
England

phone: +44 (0) 1483 274838
mobile: +44 (0) 7774 211281
email: prolog@solanum.org
web: http://www.solanum.org

BDS-PROLOG and WIN-PROLOG are trademarks of Brian D Steel, Surrey, UK, and are marketed and supported by Logic Programming Associates (LPA); for info, sales and technical support, please use the following contact points:

phone: +44 (0) 20 8871 2016
info: info_team@lpa.co.uk
sales: sale_team@lpa.co.uk
support: tech_team@lpa.co.uk
web: http://www.lpa.co.uk

Welcome to WIN-PROLOG 7.0!
## Contents

WIN-PROLOG Welcome and Update Notes ................................................................. 2
Welcome to WIN-PROLOG 7.0 .............................................................................. 6
  New Features and System Predicates ................................................................. 6
  New and Updated Toolkits .................................................................................. 6
  Enjoy the Software .............................................................................................. 6
New Features in WIN-PROLOG 7.000 ..................................................................... 7
  Full 64-bit Implementation ................................................................................ 7
  Why 64-bit? ........................................................................................................ 7
  Distinguishing the 32-bit and 64-bit Versions .................................................... 7
  Compatibility of X86 vs X64 Prologs ............................................................... 8
  Programming X86 vs X64 .................................................................................. 9
  Windows API Predicate ..................................................................................... 10
  API Return Values ............................................................................................ 10
  API Structures .................................................................................................. 11
  Installation of 32-bit and 64-bit Versions .......................................................... 13
  Other Versions .................................................................................................. 13
New Features in WIN-PROLOG 6.100 ..................................................................... 14
  TreeView Controls ............................................................................................ 14
  ListBox and ComboBox Bug Fix ....................................................................... 14
  Optimise Files Bug Fix ...................................................................................... 14
  New Pi Function ................................................................................................ 14
  Improved Help Pages ......................................................................................... 15
New Features in WIN-PROLOG 6.000 ..................................................................... 16
  Randomise List .................................................................................................. 16
  Parameterised Repeat Predicates ...................................................................... 16
  Support for Huge, “Fat” Files .......................................................................... 17
  Restored Mouse Move Message Filtering ......................................................... 17
  Including “Nameless” Files in Directory Listings .............................................. 18
  Redefinition of Call Predicate with Port .......................................................... 18
  Scanning for White Space .................................................................................. 19
  Setting of Scroll Bar “Page” Value ................................................................. 19
  Development Environment Window Names .................................................... 19
  Rebranding of the 386-PROLOG Family and Revised Version Predicate .......... 20
New Features in WIN-PROLOG 5.000

Co-Processor Maths Functions ................................................................. 21
Relaxation of Exponent Width ................................................................. 22
Cacheing of Window and Other API Handles ........................................... 23
Input and Output Flush Routines ............................................................. 23
Improved Goal Display in Aborting Errors .............................................. 24
Support for Truncated Output ................................................................. 24
Pause for a Given Elapsed Time ............................................................... 25
Revised Options Menu ............................................................................... 27
Changes to Program Abort Messages ....................................................... 27
Rich Colour Predicate ............................................................................... 27
Simplified Recursive Directory Lists ......................................................... 28
Default Character Set for Fonts ............................................................... 28
Floating Global Search & Replace ............................................................ 28
Replacement of Check Syntax Alert ......................................................... 29
Experimental Touch Input ........................................................................ 29
Command Keyboard Shortcut Changes .................................................... 29
Legacy Object File Support ...................................................................... 30
Bug Fixes .................................................................................................... 30

New Features in WIN-PROLOG 4.900

Relaxed Numeric and Formatted Syntax ................................................... 32
Unicode Conversion Predicate .................................................................. 33
Sublist and Toteall Predicates ................................................................... 33
Enhanced Help Subsystem ......................................................................... 34
Word Wrap to Window ............................................................................... 34
Optimising Compiler Support for Meta-Arguments ................................... 35
Redefinition of Mod Operator .................................................................... 35
Soft Meta-Predicate Declarations ............................................................. 35
JSON Features .......................................................................................... 36
RGB, HSL and HSV Colours ...................................................................... 36

New Features in WIN-PROLOG 4.800

Parallel Text Search .................................................................................. 37
Resizable Icon Resources ......................................................................... 38
Extended File Attributes .......................................................................... 38
Recursive File Directory Predicate .......................................................... 39
XML Features ............................................................................................ 39
New Features in WIN-PROLOG 4.700
- Windows Vista and Help
- About Time
- The arg/3, functor/3 and =./2 (univ) Predicates
- Modifications to LZSS and MZSS Checksums
- Removal of Self and Parent from Directory Listings
- Recursive Directory Listings
- Improvements to Windows Sockets (Winsock)
- GraFiX Window Default Font
- Multi-Character Text Scan Predicate
- Increase in Maximum Atom Length
- Single Key Shortcuts
- Plain Text Command History File
- VisiRule 1.5: Greatly Improved Graphics and Interaction
- Documentation Files

New Features in WIN-PROLOG 4.600
- Message Digest Five (MD5) Support
- Extended End-of-Line Parsing
- Adjustable Windows API Buffer Size
- Further Improved Mouse Handling
- Reduced Wait Loop Latency
- Maximised MDI Windows at Startup
- Single-Key Console Window Focus
- Features Removed from WIN-PROLOG 4.600
- Chimera - Agents for WIN-PROLOG
- VisiRule 1.1 - Improved Visual Programming
- Flint - Modifications and Additions

Technical Support
Contact Details
Welcome to WIN-PROLOG 7.0

Welcome to the latest version of WIN-PROLOG, a mature and distinguished 32-bit Prolog compiler and runtime support system for Microsoft Windows, joined for the first time, by a brand-new, hopefully soon to be equally distinguished, 64-bit version. Both, as ever, are accompanied by many optional toolkits and plug-ins to suit many types of application. Built upon the innovative X86-PROLOG (formerly 386-PROLOG) and X64-PROLOG engines, designed and written since 1989 and 2018 respectively by Brian D Steel, WIN-PROLOG has evolved into one of the most widely respected implementations of the Prolog language.

New Features and System Predicates

With every new release of WIN-PROLOG, features are added, and many of these are reflected in additional system predicates; documentation for these can be found in the Technical Reference, provided you know they’re there: one of the primary purposes of these Update Notes is to let you know what to look for.

New and Updated Toolkits

As well as extending the basic vocabulary of built-in predicates and features within WIN-PROLOG itself, new releases often extend the library of toolkits that can be used with the system; again, each such toolkit has an associated manual, which is great if you notice it’s presence: these Update Notes will let you know about them.

Enjoy the Software

We hope you will find much of interest in the new release of WIN-PROLOG, and will take a few moments to read this document before getting underway with your new software package.

Brian D Steel, 01 Mar 19
New Features in WIN-PROLOG 7.000

Full 64-bit Implementation

By far the most significant feature of version 7.000, is that it is now available both as a 32-bit version and a genuine, full-bore 64-bit version. As before, the 32-bit version runs on all versions of Windows from 98SE through 10; the new 64-bit version will run on all versions of Windows X64 from 7 to 10.

Why 64-bit?

In a nutshell, MEMORY! The X86 architecture gave users a theoretical 4Gb address space into which to cram all heaps, stacks and program code; in Windows this was halved to 2Gb, because Windows reserves half of the memory address space for itself.

The X64 architecture smashes through this limitation, allowing applications to access as much memory as can be installed into any given computer. 128Gb of program clauses? No problem any more!

Unfortunately, everything comes with a price, and the 64-bit memory pointers mean that the average Prolog memory structure is about 80% larger than before; where an 8Mb heap once sufficed for a given application in X86-PROLOG, it might be better to allocate 16Mb for X64.

Increasing the amount of data being processed from 32-bit chunks to 64-bit also impacts on speed, not least, because the number of cache hits will be reduced. In testing, X64-PROLOG has been found to be about 15% slower than X86-PROLOG (formerly, 386-PROLOG).

Distinguishing the 32-bit and 64-bit Versions

The two versions of WIN-PROLOG have slightly modified welcome banners, which include the label, “X86” for the 32-bit version, and “X64” for the 64-bit version, for example:

BDS WIN-PROLOG 7.000 X64 S/N 0020426400 02 Nov 2018

For additional visual recognition of version, early (test) releases of X64-PROLOG will include a red application icon and splash screen, rather than the usual blue ones.
Compatibility of X86 vs X64 Prologs

The two versions of the engine maintain maximum mutual compatibility; in practice this means that about 99% of code “just works”. This source code compatibility is extended to the object file (.PC) format: you can optimise files in X86, then load and run them in X64, or vice versa.

The only places where a programmer might need to know which version of Prolog is in use, is when handling large integers, such as in use of the rpn/2 predicate, when processing formatted integers with the fread/4 and fwrite/4 predicates, and when handling values in association with winapi/4 (see below). Simply, X86-PROLOG uses 32-bit integers, in the range -(2^31)..(2^31)-1, while X64-PROLOG uses 64-bit integers, in the range -(2^63)..(2^63)-1. So:

?- fread( r, 8, 16, X ) <= `ffffffff` .

will, in X86-PROLOG, return:

X = -1

but will, in X64-PROLOG, return:

X = 4294967295

Issues can occur with output, where (for example) negative or large positive integers require 16 digits of output in X64, rather than just 8 as in X86:

?- fwrite( r, 0, 16, -1 ), nl.

will, in X86-PROLOG, display:

FFFFFFFF

but will, in X64-PROLOG, display:

FFFFFFFFFFFFFFFF

Another place where the larger native integer size in X64-PROLOG might affect programs compared to X86-PROLOG, is in use of the rpn/2 integer arithmetic predicate, so that:
?- rpn( [16’80000000,1,1], X ), fwrite( r, 0, 16, X ), nl.

will, in **X86-PROLOG**, return and display:

```
1
X = 1
```

but will, in **X64-PROLOG**, return and display:

```
100000000
X = 4294967295
```

In pretty much all code except that handling large integers, both the X86 and X64 versions of WIN-PROLOG will behave identically.

**Programming X86 vs X64**

Just so that programs have the option of treating the 32-bit (X86) and 64-bit (X64) versions separately, for example when doing integer operations as highlighted in the previous section, two more cases have been added to the ver/1 predicate:

?- ver( 32 ).

returns:

<table>
<thead>
<tr>
<th></th>
<th>(in X86-PROLOG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

while:

?- ver( 64 ).

returns:

<table>
<thead>
<tr>
<th></th>
<th>(in X64-PROLOG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

Both calls will fail in earlier versions of **386-PROLOG**, so this allows special cases to be created if necessary, with code constructs such as:
foo( ... ) :-
  (  ver( 32 )
  -> <do 32-bit specific X86 code>
  ;  ver( 64 )
  -> <do 64-bit specific X64 code>
  ;  <do code for earlier version>
).

Again, it is stressed that you will need the above only in rare cases when dealing with large integers, or possibly in advanced uses of the winapi/4 predicate (see below).

**Windows API Predicate**

The X64 winapi/4 predicate retains the syntax and semantics of the X86 version, such that nearly all calls out to Windows APIs will work directly, with no further tweaking. There are, however, two cases to consider. Firstly, the size of the return value, as described in the next two sections.

**API Return Values**

Most integer X64 Windows API functions actually return 32-bit signed values, rather than 64-bit ones, so that, for example, “-1” is returned as “0xFFFFFFFF” (4294967295), rather than “0xFFFFFFFFFFFFFFFF” (-1).

Two new functions have been added to the rpn/2 predicate to help handle transitions between 32-bit and 64-bit values:

```
$  - sign extension, which copies bit 31 into bits 63..32
&  - word truncation, which zeroes bits 63..32
```

As examples, see these calls:

```
?- rpn( [4294967295,$], X ).
X = -1

?- rpn( [-1,$], X ).
X = -1

?- rpn( [-1,&], X ).
X = 4294967295
```
?- rpn([4294967295,&], X).
X = 4294967295

So you would use "$" if (say) you wanted to compare the return value from a winapi/4 call against the value "-1"; previously, code might look like this:

```prolog
winapi( (module,some_32_bit_function), [], 0, Result ),
( Result = -1
-> fail
; true
),
```

but this will not work correctly in X64. Instead, replace "=/2" with an call to rpn/2:

```prolog
winapi( (module,some_32_bit_function), [], 0, Result ),
( rpn([Result,$], -1)
-> fail
; true
),
```

which will work both in X64 and X86.

In **X86-PROLOG**, both functions are present, but essentially do nothing.

### API Structures

Passing simple values to winapi/4, such as integers, text strings, memory file buffer names, and so on, works identically on the X86 and X64 versions of Prolog, but care must be taken when calling APIs that expect, or return, structures. Simply, X86 structures tend to be packed with 32-bit values and pointers, while the equivalent X64 structures will contain a mixture of 32-bit or 64-bit values, and 64-bit pointers. To complicate matters further, X64 structures always align 64-bit values on 8-byte boundaries. So imagine you have a "C" structure containing the following fields:

```c
{
   INT size ;
   HWND handle ;
   INT mode ;
}
```
previous code to load this into a memory buffer might have looked like:

\[
\text{\begin{verbatim}
( \text{putx}( 4, \text{Size} ),
   \text{putx}( 4, \text{Handle} ),
   \text{putx}( 4, \text{Mode} )
) \Rightarrow \text{Struct},
\end{verbatim}}
\]

This would work for X86, if “Struct” was then passed into winapi/4, but not in X64. Thanks to the new extension to ver/1 (see above), we can now write something like this:

\[
\text{\begin{verbatim}
( \text{ver}( 32 )
   \Rightarrow ( \text{putx}( 4, \text{Size} ),
   \text{putx}( 4, \text{Handle} ),
   \text{putx}( 4, \text{Mode} )
) \Rightarrow \text{Struct}
; \text{ver}( 64 )
   \Rightarrow ( \text{putx}( 4, \text{Size} ),
   \text{putx}( 4, 0 ), \% \text{padding to align handle}
   \text{putx}( 8, \text{Handle} ),
   \text{putx}( 4, \text{Mode} )
) \Rightarrow \text{Struct}
),
\end{verbatim}}
\]

Alternatively, to cut down on code duplication, this would also work on both platforms:

\[
\text{\begin{verbatim}
( \text{ver}( 32 )
   \Rightarrow Q = 4
; \text{ver}( 64 )
   \Rightarrow Q = 8
),
( \text{putx}( Q, \text{Size} ), \% \text{includes alignment padding}
   \text{putx}( Q, \text{Handle} ),
   \text{putx}( 4, \text{Mode} )
) \Rightarrow \text{Struct}
\end{verbatim}}
\]
The same tricks work with getx/2, when reading structures after calling a Windows API function.

**Installation of 32-bit and 64-bit Versions**

The two versions of the Prolog engine, 32-bit **X86-PROLOG** and 64-bit **X64-PROLOG**, are so compatible, that you can run them side by side, simultaneously, loading the same toolkits and source files. Being aware only of the issues mentioned above, the only difference you should find is that on large-memory machines, the latter system gives you effectively unlimited sizes for your various heaps, stacks, etc.

To run the 32-bit system, do:

1) Copy P32.EXE to PRO386W.EXE  
2) Run the system

To run the 64-bit system, do:

1) Copy P64.EXE to PRO386W.EXE  
2) Run the system

If you want to run both versions at once, for side-by-side testing, then rather than copying files, use the “@” option:

To run the 32-bit without copying, enter this command:

```
P32 @PRO386W
```

To run the 64-bit without copying, enter this command:

```
P64 @PRO386W
```

This way, both 32-bit and 64-bit versions can be run simultaneously, which is very helpful for testing.

**Other Versions**

As well as X64 WIN-PROLOG, there are X64 versions of both CON-PROLOG and DLL-PROLOG, which again retain near-total compatibility with their X86 counterparts. Contact LPA if you are interested in checking these out.
New Features in WIN-PROLOG 6.100

TreeView Controls

The TreeView control, a feature often requested by users, has now been fully built into WIN-PROLOG. This useful window type allows for the display, browsing and editing of hierarchy data, for example nested file folders, or Prolog call trees, and a number of new dedicated messages have been added to simplify the tracking of edits, traverses, collapses and expansions of tree branches.

Along with a set of new predicates (wtvw??/n), several dozen new control messages (tvm.*) for use with sndmsg/5, and a number of new window styles (tvs.*) for wccreate/8, TreeView has also introduced no fewer than eight new Prolog notification messages, msg_tv*. The latter report changes to the TreeView in real time, as the user navigates or edits the tree, allowing for fully interactive trees to be implemented with nothing more than a regular WIN-PROLOG message handler.

ListBox and ComboBox Bug Fix

A long term bug was discovered in both wlstget/4 and wcmbget/4, where the predicate would fail of the data item (4th argument) had previously been set to -1 during wlstadd/4 or wcmbadd/4 respectively. The “-1” return value was being interpreted as the Windows error, LB_ERR or CB_ERR, respectively, which is defined as -1. By removing a fail test, it is now possible to return any value previously set.

Optimise Files Bug Fix

Another long term bug was discovered, which resulted in syntax errors in the file-to-file optimising compiler, when the source file contained Unicode characters. This version of the optimiser has now been updated to recognise and process Unicode files correctly.

New Pi Function

A new function has been defined both in the is/2 family of predicates, and fpn/2, to compute multiples of the mathematical constant, Pi. The function takes a single argument, which is multiplied by Pi to give the return value. For example:

?- X is pi(1).
X = 3.141592653589793

?- Radius is 0.25, Area is pi(sq(Radius)).
Radius = 0.25
Area = 1.963495408493621e-1
Improved Help Pages

The context-sensitive help system has had a major revamp, greatly improving the appearance and lay-out of pages. Code examples and sample calls now use a fixed-width font, with consistent spacing; tables are more neatly laid out with colour-coded rows to make it easier to reference across the table, and all hyperlinks to other pages now include a comment about what is on that page. A lighter style, with more readable fonts, completes a much needed and very welcome make-over.
New Features in WIN-PROLOG 6.000

Randomise List

A new predicate, `stir/2`, lets you thoroughly randomise any list of terms, using the Fisher/Yates Shuffle Algorithm, powered by the 1185-bit M255 PRANG (Marsaglia Zaman Pseudo Random Number Generator, with Steel Comb Filtering).

For lists of up to 192 elements, every possible permutation is equally likely to be generated. For larger lists, only a subset of possible permutations can be generated. This is not a software defect; rather a result of the properties of the Factorial function: `fact(192)` is smaller than $2^{\text{1185}}$, but `fact(193)` and beyond is bigger, meaning there are more potential permutations than there are numbers in the random sequence!

Paramaterised Repeat Predicates

Additional repeat/n predicates have been included, which return the current repeat iteration. Previous versions of WIN-PROLOG already had:

- `repeat` - repeat forever on backtracking
- `repeat(N)` - repeat given number N times on backtracking

Now we also have:

- `repeat(N,I)` - repeat given number N times on backtracking, returning current repeat number in I
- `repeat(F,T,I)` - repeat for each integer between F and T inclusive, returning current repeat index in I

So, for example, you could type:

```
?- repeat(5, I), write( I ), nl, fail.
1
2
3
4
5
no
```
For a different numerical range, you could type:

```prolog
?- repeat( 21, 25, I ), write( I ), nl, fail.
21
22
23
24
25
no
```

Note that `repeat( X, Y )` is equivalent to calling `repeat( 1, X, Y ).`

These new `repeat/2` and `repeat/3` predicates will be handy in simulating simple iterative loops in Prolog programs.

Please note that the latter predicate, `repeat/3`, has replaced the quirkily-named `integer_bound/3` that existed in versions of WIN-PROLOG up to and including 5.000; a library file, `50_INTBD.PL`, contains a utility which can update your source files automatically, replacing calls to the defunct predicate with calls to the new one.

**Support for Huge, “Fat” Files**

Extensive internal changes have been made to support larger files than were previously handled. Up till version 5.000, WIN-PROLOG has been able to read or write files of up to $2^{32}-1$ (4,294,967,295) bytes in size. While adequate for most purposes, it has meant that Prolog applications struggled to support the processing of increasingly common video file formats, or large databases.

In version 6.000, files of up to $2^{53}-1$ (9,007,199,254,740,991) bytes can be created, read and written. With the new file size limit at just over 9 Petabytes, or 9,000 Terabytes, all types of file should will be processable by Prolog for the foreseeable future.

A further advantage of the new processing, is that once a file size or offset exceeds $2^{31}-1$ (2,147,483,647) bytes, the size will no longer be reported as a negative integer as previously, avoiding the need for modulo arithmetic to compute the correct size.

**Restored Mouse Move Message Filtering**

Up till and including WIN-PROLOG 4.920, filtering was applied to “msg_mousetove” messages, which occur with great frequency, so that only the most recent such message is ever reported to a Prolog application; in version 5.000, this filtering was removed to enable more accurate tracking of the mouse in drawing applications. Unfortunately, two side effects made the decision to remove such filtering unhelpful: first, the message queue could quickly fill up to the point of overflowing, and second, applications which redraw images as the mouse moves around, were working overtime doing multiple unnecessary
redraws. In **WIN-PROLOG** 6.000, the original filtering has been reintroduced.

**Including “Nameless” Files in Directory Listings**

It was discovered that files containing no “name” portion, but only an extension, were not showing up in directory listings. For example, if a folder contained two files:

```
  .hello
  there.world
```

a call to `dir/3` would only return the file with a “name” portion:

```
?- dir( .*.*, 0, X ).
X = ['there.world']
```

This bug dates back to version 4.620, in late 2005, when code was added to remove the virtual “self” (“.”) and “parent” (“..”) directories from every listing below the root folder. In version 6.000, these two virtual folders are still omitted, but other files and folders whose names begin with “.” are no longer being masked out.

**Redefinition of Call Predicate with Port**

The `call/2` predicate, which is used in debugging and similar places, was redefined as part of some internal optimisations. Formerly, it returned one of three atoms, depending on the state of the call: two of these atoms have been renamed better to describe the situation, and to allow direct meta-calls of the flag where needed. Here are the details:

<table>
<thead>
<tr>
<th>State</th>
<th>5.000</th>
<th>6.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call succeeds, further solutions possible</td>
<td>exit</td>
<td>true</td>
</tr>
<tr>
<td>Call succeeds, no further solutions</td>
<td>done</td>
<td>!</td>
</tr>
<tr>
<td>Call fails</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>
Scanning for White Space

The `scan/3` predicate has been extended to provide a special option to scan for blocks of white space on the input file: by specifying an empty list as the first argument, this predicate locate and return the whole of the next sequence white space characters in a file; for example:

```prolog
?- scan([], 0, X) <~ `abc~M~Jdef`
X = `~M~J`
```

Setting of Scroll Bar “Page” Value

The `wrange/4` predicate has been replaced with `wrange/5`, whose new, fifth argument specifies that “page” size, which specifies how far to move the scroll bar when the “page” area is clicked upon.

Previously, `WIN-PROLOG` always scrolled by an amount equal height (vertical scroll) or width (horizontal scroll) of a window’s client area, and this amount could not be changed. While this was a useful default for graphics windows, it made no sense for other purposes, for example scrolling a customised text box or somesuch.

Because the `WIN-PROLOG` Kernel now supports the scroll bar page value, there is no 100% exact equivalent of the old behaviour. The closest would be to implement the old `wrange/4` something like this:

```prolog
wrange( Wind, Bar, Lo, Hi ) :-
  warrea( Wind, _, _, Wide, High ),
  ( Bar = 1
  -> Page = Wide
  ; Bar = 2
  -> Page = High
  ; Page = 0 ),
  wrange( Wind, Bar, Lo, Hi, Page ).
```

Development Environment Window Names

Unlike most other Multiple Document Interface (MDI) programs, `WIN-PROLOG` has traditionally shown the entire pathname of any given file as the title of its window. With the increasing tendency for deeply nested folders and long files names, this became unwieldy, especially in the “More Windows” menu option, where the pop-up “Select Window” dialog was too narrow to show the actual file name. In `WIN-PROLOG` version 6.000, window names have been changed only to show the file name and extension.
Rebranding of the 386-PROLOG Family and Revised Version Predicate

Early in 2014, the IPR to the 386-PROLOG family (formerly known as “LPA PROLOG”) was transferred from LPA to Brian D Steel. Brian has been solely responsible for the design, implementation and maintenance of the 386-PROLOG engine throughout its lifetime, and as the software approaches its 25th anniversary, he has taken on personal responsibility for its continued support and development.

The only changes that will be visible following the software’s change of ownership, will be new wording in the “welcome” banner (see below), and newly designed icons and bitmaps.

In all practical respects, “BDS PROLOG” will be marketed and supported exactly as before; LPA will continue to develop the Toolkits, while Brian will develop, extend and maintain the core Prolog engine, just as has been the case through the past quarter century.

With the above in mind, the ver/4 predicate, whose first argument used to return a 14-character atom of the form, “LPA ???-PROLOG”, now omits the “LPA “ (or latterly, “BDS “) prefix, and returns the 10-character product name, “???-PROLOG”, where “???” will be one of “BAT”, “CON”, “DOS”, “WIN” and “DLL”, or one of a handful of specialist names.
New Features in WIN-PROLOG 5.000

Co-Processor Maths Functions

From its earliest days, the 386-PROLOG engine that powers WIN-PROLOG and its sibling products, has used an externally-sourced floating point package to perform all of its non-integer arithmetic functionality. This source code assembler library, called DPAC, had been adapted and maintained by LPA long after its originators went out of business.

From time to time, strange and esoteric bugs would arise, and increasingly, these became harder to track down and fix in the scantily commented code. So for the 5.000 release, it was decided to start over.

A brand new library of advanced mathematical and arithmetical functions had already been designed by Brian D Steel, independently of the 386-PROLOG engine, for inclusion in some of his utility programs, and after considerable analysis and extensive testing, it was felt the time had come to replace DPAC with his new x87 library.

A change of this type had repercussions throughout the entire 386-PROLOG engine, and the decision was made not to release 5.000 until such time as it had been in regular, internal use, for many months, and any emerging bugs had been fixed. That time has now arrived, and WIN-PROLOG is now presented with a full x87 co-processor hosted arithmetic engine.

Compared to the old DPAC, the new x87 engine has a number of advantages:

1) x87 is smaller, and faster than DPAC

2) x87 is more accurate than DPAC; for example, in WIN-PROLOG 4.920 (DPAC):

   \[ \text{?- fnpr([29,sin,cos,tan,sqrt,ln,aln,2,\^,atan,acos,asin], A ).} \]
   \[ A = 28.999999993105 \]

   whereas in WIN-PROLOG 5.000 (x87):

   \[ \text{?- fnpr([29,sin,cos,tan,sqrt,ln,aln,2,\^,atan,acos,asin], A ).} \]
   \[ A = 28.99999999999988 \]

3) x87 exponentiation yields integer results, unlike DPAC; for example in WIN-PROLOG 4.920 (DPAC):
?- X is 2 ^ 3, Y is 8 - X.
X = 8.0
Y = 2.66453525910038E-015

whereas in \texttt{WIN-PROLOG 5.000 (x87)}:

?- X is 2 ^ 3, Y is 8 - X.
X = 8
Y = 0

Note that in the accuracy example above, using the floating point reverse polish notation predicate, \texttt{fpn/2}, the 5.000/x87 case could have replaced, "2, ^" with a call to the new "square" function, "sq". Like all functions in \texttt{fpn/2}, this is also available in \texttt{is/2}:

?- X is sq(2).
X = 4

Of course, \texttt{386-PROLOG} is no longer an appropriate name for the underlying engine, since it now requires a 486DX processor as a minimum platform (or a 386 with 387 coprocessor board).

\textbf{Relaxation of Exponent Width}

The old DPAC routine for converting ASCII strings to numbers and vice versa, had a limitation on the written size of the exponent: it was required to be a maximum of three digits, and moreover always displayed with 3 digits on output; the x87 routines allow any width of exponent on input, and display output with just the significant digits; for example, in \texttt{WIN-PROLOG 4.920 (DPAC)}:

?- X = 1.23e0045.
Error 42, Syntax Error, Trying ered/2

?- X = 1.23e45.
X = 1.23E045

whereas in \texttt{WIN-PROLOG 5.000 (x87)}:

?- X = 1.23e0045.
X = 1.23e45

?- X = 1.23e45.
X = 1.23e45

Notice also a small cosmetic change, in the use of a lowercase "e" to mark the exponent, rather than "E"; the smaller character height makes the join between the mantissa and exponent easier to see at a glance.

Cacheing of Window and Other API Handles

A performance issue was identified in WIN-PROLOG, which caused the conversion of raw GUI handles into their logical, Prolog equivalents, became increasingly slow as the size of the text heap increased and filled up with atoms. In extreme cases, this resulted in the whole of the WIN-PROLOG Development Environment becoming very sluggish, especially when syntax colouring was enabled.

An advanced cacheing scheme was introduced, to maintain an independent linked list of known GUI resources, resulting in a massive increase of performance of handle reconciliation, and making it independent of the size or population of the text heap.

This cacheing significantly enhances not just window message handling, but also TCP/IP (socket) and MIDI (music) applications.

Input and Output Flush Routines

In WIN-PROLOG, it has long been the case that outputting the character, <ctrl-Z> (Unicode Character 001B), the console window, will force a "flush" of all buffered characters to the physical display surface, but there has been no equivalent function for other output streams, such as disk files. With 5.000, a new predicate has been added to provide this functionality in a device-independent form.

The only problem was, the obvious name for this predicate, flush/0, was already in use, as a misnomer for an error-handling predicate that could be used to empty the current keyboard buffer or input line from a file. It was decided to rename the existing flush/0, "empty/0", better to reflect what it does, allowing the new flush/0 predicate to be defined.

In the unlikely event that user code used the old flush/0 predicate, this will result in a slight incompatibility. Please therefore rewrite any calls to the old flush/0, so that they call empty/0 instead.
Improved Goal Display in Aborting Errors

While most errors in WIN-PROLOG are caught and handled either by the Development Environment or by user code, some errors cannot be processed in this fashion. Dubbed "aborting" errors, they always result in code stopping running, and control being returned to the console "?-" prompt with the output of a terse message; the same happens to all errors if the system error handlers have been deleted. Up until version 4.920, these messages contained a brief predicate/arity element to show which predicate had reported the error; with 5.000, the first portion of the actual goal is shown, complete with the arguments; for example, WIN-PROLOG 4.920 (with no error handler):

```
?- fwrite( f, 10, 3, hello ).
Error 23, Type Error, Trying fwrite/4
```

whereas in WIN-PROLOG 5.000 (with no error handler):

```
?- fwrite( f, 10, 3, hello ).
Error 23 - Type Error - (fwrite f 3 10 hello)
```

The same goal output is also provided now, when a "system" error occurs, either because of an internal software malfunction, or because the user has forced the state by pressing <ctrl><right-shift><break>.

Support for Truncated Output

In conjunction with the improved handling of aborting, unhandled programmble, and system errors, support was added to allow automatically truncated term output. This is useful when displaying very long terms, where typically only the starting section is of interest, and especially when displaying infinite ones; indeed, in the standard console, the latter was previously handled very poorly; for example, in WIN-PROLOG 4.920:

```
?- X = [foo|X].
Term too deep
```

whereas in WIN-PROLOG 5.000:

```
?- X = [foo|X].
X = [foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,foo,... (stops at 1024 chars)
```

Two new calls, one to eprint/3, and one to sprint/3, were added to allow user code to perform truncated output, by specifying the maximum field width as a negative number in the third argument. So that the calling program can detect whether the output was completed or truncated, the print predicates fail in the latter case:
?- eprint( pi(3.14159,`blackberry & apple`), [], -40 ).
pi(3.14159,`blackberry & apple`)yes

?- eprint( pi(3.14159,`blackberry & apple`), [], -20 ).
pi(3.14159,`blackberry & apple`)no

?- sprint( pi(3.14159,`blackberry & apple`), [], -40 ).
(pi 3.14159 `blackberry & apple`)yes

?- sprint( pi(3.14159,`blackberry & apple`), [], -20 ).
(pi 3.14159 `blackberry & apple`)no

As illustrated earlier, the truncated output is now used in the main console supervisor, to limit the displayed portion of any given solution to 1024 characters. It is also used in the new error goal display (see above) to ensure that the displaying of a solution involving an infinite term cannot in itself be an infinite process.

**Pause for a Given Elapsed Time**

An incredibly common requirement, especially in animated user interface elements, is to pause execution for a specified period. Up until now, it has been necessary to concoct a delay timer using a trick like:

```prolog
pause( Time ) :-
  ( ms( repeat, Test ),
    Time > Test
  -> true

).```
or somesuch. The main problem with this approach, is that, while working, it hogged the CPU and caused regular garbage collections during which the pause might exceed the originally intended interval. A new predicate, `pause/1`, has now been implemented in assembler, to provide a more versatile and predictable delay:

```prolog
?- pause( 5000 ).
yes (after 5 seconds)
```

Please note: if the message flag is set, or if timer events occur, they safely interrupt `pause/1`, which will continue with its remaining elapsed time after the appropriate event has been processed.
A new list membership predicate, member/4, has been added, to enable lists to be joined or split at a specific, known element:

\[
\text{member( Item, List, Left, Rite )}
\]

--------------------------------
\[
\text{Item - an element of List}
\text{List - a List}
\text{Left - left sublist relative to Item}
\text{Rite - right sublist relative to Item}
\]

For example:

?- member( X, [the,quick,brown,fox], L, R ).
X = the
L = []
R = [quick,brown,fox]

X = quick
L = [the]
R = [brown,fox]

X = brown
L = [the,quick]
R = [fox]

X = fox
L = [the,quick,brown]
R = []

no

?- member( a, [b,c,d,a,e,f,g,a,h,i,j], L, R ).
L = [b,c,d]
R = [e,f,g,a,h,i,j]

L = [b,c,d,a,e,f,g]
R = [h,i,j]
Revised Options Menu

The contents of the "Options" menu were regrouped, and the "toggle" cases, including "Status Bars" and the new "Word Wrap", were simplified so as not to require a dialog window to switch them on or off.

Changes to Program Abort Messages

In version 4.920, whenever a program evaluation was aborted, as the end result of an unhandled error, a user "break", or because the abort/0 predicate was called by the program, the console displayed the message, "Aborted", before returning to the prompt; for example:

?- abort.
Aborted

This message, which could not be overridden, has been removed. However, when a user forces a break, by pressing <ctrl><break> and confirming the dialog, while 4.920 again displayed "Aborted", version 5.000 will now display the message, "Break".

Rich Colour Predicate

A new predicate, rich_colour/5, was defined to enable quick recolouring of selected areas of text in a rich edit window:

rich_colour( Wind, Tops, Ends, Fore, Back )
-------------------------------------------
Wind - handle of a rich edit window
Tops - the top offset of the selected area to be coloured
Ends - the end offset of the selected area to be coloured
Fore - an integer containing the RGB value for the foreground
Back - an integer containing the RGB value for the background

For example:

?- rich_colour( (1,1), 52, 104, 16'ff0000, 16'ffff00 ).
yes
will colour the console window between offsets 52 and 104, with a light blue foreground and light green background. If the console window contained the standard 'LPA WIN-PROLOG' welcome banner at its top, then the whole of the line starting with the above text, will be what has been coloured.

Please note that because Intel processors use a "little endian" integer storage scheme, the RGB values passed into rich_colour/5 might appear to be back to front! For the least significant six hexadecimal digits of the integer, the first two specify Blue, the next two, Green, and the final two, Red, i.e. "16'bbgrr". This can be seen in the above example.

Simplified Recursive Directory Lists

The dir/4 predicate was reverted to the format originally implemented but not documented in version 4.7, rather than the modified version documented in 4.8, in which the first argument is an atom, specifying a single root folder, rather than a list of atoms as roots. This rewind has been made because of issues with trying to handle multiple volumes, network paths and plug-in devices in a single call, when it is invariably simpler to treat each such volume or device in turn.

Default Character Set for Fonts

Both the wfcreate/4 and gfx_font_create/4 predicates were modified to call the Windows API with a value of "DEFAULT_CHARSET" rather than "ANSI_CHARSET", to enable graphics fonts and others with non-standard glyph mapping to be created for decorative purposes. Previously, only fonts which contained the standard ANSI character set could be created by these predicates; now any font can be used, including "graphics" or "pi" fonts like CHEQ (for chess pieces) or SONATA (for musical symbols).

Floating Global Search & Replace

The clunky old "Find" and "Change" boxes, together with their supporting predicates, fndbox/2 and chgbox/3, have been retired, and replaced with an all-new floating search & replace dialog.

The new dialog is smaller than either of its predecessors, and can be toggled between "search" and "search/replace" modes by the simple click of a button. It can also be turned on and off programmatically by a single new predicate, seabox/1:

    seabox( Mode )
    --------------
    Mode - an integer in the domain, [0,1,2]

For example:

    ?- seabox( 0 ).       (hides the dialog)
A key feature of the new search & replace dialog, is that it highlights available selections in the current rich edit window in real time, giving instantaneous feedback to your search. You can jump between the dialog and your text or code as often as you want, and also jump directly to other windows that contain your current search text. All in all, the new "seabox" is a massive improvement in usability compared to the old find and change boxes.

Replacement of Check Syntax Alert

A small but welcome change has been made to the syntax error dialog that pops up during the "Check Syntax" option in the "Run" menu: up to 4.920, it used a "Yes/No" message box to display an error, which could only be dismissed with a mouse click; in 5.000, this has been replaced with an "OK/Cancel" box, which can also be dismissed with the <escape> key.

Experimental Touch Input

A new message, "msg_touch", has been implemented, together with two predicates, touch/0 and touch/3, to support multi-touch input on Windows 7 and 8 tablet computers. At this stage, support is only intended for experimental uses and some example programs, but the intention is to integrate it firmly into future graphical applications and development environments, not least including an all-new version of VisiRule.

Command Keyboard Shortcut Changes

The keyboard shortcut used to "Break" into a Prolog evaluation has been changed from <ctrl><break> to <ctrl><del>, and the keys used to control the console "command history" option have changed from <ctrl><pgup> and <ctrl><pgdn> to <ctrl><up> and <ctrl><down> respectively.

These changes have been made to support the Windows On-Screen (Touch) Keyboard, as well as the increasingly common reduced-size keyboards, such as those found in smaller laptops and bluetooth devices for tablet computers, all of which include delete and cursor keys, but which often omit the break, page up and page down keys.

The following table summarises these changes:
<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>New Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break!</td>
<td>&lt;ctrl&gt;&lt;break&gt;</td>
<td>&lt;ctrl&gt;&lt;del&gt;</td>
</tr>
<tr>
<td>History Up</td>
<td>&lt;ctrl&gt;&lt;pgup&gt;</td>
<td>&lt;ctrl&gt;&lt;up&gt;</td>
</tr>
<tr>
<td>History Down</td>
<td>&lt;ctrl&gt;&lt;pgdn&gt;</td>
<td>&lt;ctrl&gt;&lt;down&gt;</td>
</tr>
</tbody>
</table>

### Legacy Object File Support

**WIN-PROLOG** 5.000 includes support for "legacy" object files, compiled with versions of **386-PROLOG** back to 4.640. A change was made to the Sigma2 virtual machine during the development of version 4.630, which renders object files from this and earlier versions, incompatible with the current system.

### New Icons

To signify the substantial changes "under the hood" in **WIN-PROLOG** 5.000, a new set of icons and splash screen was designed. The ten 7358-byte ICON*.ICO files, originally created for version 4.720 at 48*48 pixels and 24-bit (16.7M-colour) depth, were replaced with a new set of contemporary "iPhone" style 9662-byte 48*48 icons with 32-bit (16.7M-colour+alpha) depth. The main application icon, ICOMAPL, was saved at 10806 bytes, with a hand-drawn 16*16*32 image included for optimised display on later Windows task bars.

### Bug Fixes

As ever, extensive use together with background maintenance and an ongoing code review, revealed some bugs, which have been fixed for 5.000. These include:

1) A problem in the integer reverse polish notation predicate, rpn/2, which could allow some two-argument functions to be called when only one argument remained on the stack, with unpredictable results, was identified and fixed.

2) An ancient unification bug was found in the optimized Sigma2 instruction set which could result in floating point numbers being garbage collected midway through unification: this bug had been hiding in the system for around 20 years, but has been fixed now.
3) A very old bug was found which resulted in the occasional failure of decompilation of the supplementary terms which are used to store the variable names displayed by listing/0, with the result that occasional a clause would fail to appear in program listings: this bug had been present but undetected for around 16 years, but is now fixed.

4) The UTF-8 input routines were capable of reading illegally encoded values, and have been tightened up to be strict in the processing of multibyte Unicode characters, with calls to generate an error if an incorrect encoding is found.

5) Code was added to the compression, decompression, encoding and decoding predicates, to ensure that they only worked with ISO/IEC 8859-1 streams for their binary data, to prevent the creation of invalid files.

6) The main application message loop was adjusted to give dialog windows precedence when processing incoming messages; this fixed a minor problem that previously meant that dialogs embedded within other windows, for example within the MDI, did not respond correctly to standard Windows keystrokes.

7) A minor bug existing in catch/3 predicate, when a non-executable term was executed; it was returning an error term with an arbitrary arity of 3, rather than 0; it now returns the error term, []/0.

8) The help/2 predicate was fixed to prevent errors occurring when it was called with its second argument bound; the new argument type testing regularises the error handling.

9) Automatic dialog resizing was corrected to size dialogs in proportion to the width and height of the "M" character in the ANSI Var Font, rather than scaling both to just the height of "O" in the System Font, to reflect the use of the ANSI Var Font in all system dialogs.

10) The listing/1 predicate was fixed to avoid errors being generated when a non-existent predicate was in its argument.

11) The syntax colouring mechanism was modified to use an explicit colour of "white" behind terms being coloured, rather than "default", to help terms show up properly in rich edit windows that use tinted backgrounds.

12) The internal programming of opnbox/5 and savbox/5 was reworked to change a number of parameters which are passed through to the Windows API, in order to avoid occasional failures and errors that had been reported by some users.

13) The display/1 predicate was corrected to use quoted canonical output, rather than unquoted output, to make it conform to the standard Edinburgh behaviour.

14) The code which enables and disables menus in the Development Environment could cause a recurring "Program Space Full" when the program heap is full; it could be difficult to escape from this state; this has been fixed in version 5.000.
New Features in WIN-PROLOG 4.900

Relaxed Numeric and Formatted Syntax

Partly in support for JSON (see below), but also for increased compatibility with most other computer languages, some optional relaxations have been made to the number syntax used by the term read predicates. Firstly, it is no long necessary to have an explicit decimal portion in exponential formats:

?- X = 123e-4 .
X = 0.0123

is now identical in meaning to:

?- X = 123.0e-4 .
X = 0.0123

Previously, the first example would have resulted in a syntax error. This change is sensitive to the Edinburgh Language Extensions flag, as set or tested by elex/1, and so can be reversed temporarily or permanently if required.

A related change was made to the formatted input predicate, fread/4: previously, its Radix input format ("r") required alphabetic digits in the range A..Z to be in upper case; now, either case can be used interchangeably, again for greater compatibility with JSON and other computer languages:

?- fread( r, 4, 16, Hex ). abcd
Hex = 43981

is now identical in meaning to:

?- fread( r, 4, 16, Hex ). ABCD
Hex = 43981

Previously, the first case would have resulted in failure of the call to fread/4.
Unicode Conversion Predicate

While **WIN-PROLOG** has included comprehensive, automatic and transparent Unicode support for many years, making most Prolog programs entirely unaware of the dozen or so character encodings commonly in use today, there nonetheless remains the occasional application where it is necessary to perform explicit conversions of strings, especially where those applications are web or network based.

Rather than force users to read and write from disk or memory files in order to perform Unicode conversions, 4.900 includes a direct string-to-string converter call **strutf/3**. It takes as its first two arguments respectively, a binary, **WIN-PROLOG** format string and a variable, or a variable and an 8-bit Unicode UTF-encoded string. The third argument specifies the encoding to use, offering all the same options as **fcreate/5**, as shown in this example, which converts a binary string into UTF-8:

```
?- strutf( `This is a TM: ™`, UTF8, 1).
UTF8 = `This is a TM: â?¢`
```

Here is a similar call, which this time returns the UTF-16LE encoding for the same string:

```
?- strutf( `This is a TM: ™`, UTF16LE, -2).
UTF16LE = `T~@h~@i~@s~@ ~@i~@s~@ ~@a~@ ~@T~@M~@:~@ ~@"!
```

Sublist and Toteall Predicates

A new predicate, **sublist/2**, has been introduced to help list processing programs that need to work with ordered sublists, of known or unknown length, of a parent list. Similar in use to **member/2**, extends the functionality to return 1, 2 or more elements at a time:

```
?- sublist( [One,Two], [cat,and,dog] ).
One = cat ,
Two = and ;

One = cat ,
Two = dog ;

One = and ,
Two = dog ;
```

A second new predicate, **toteall/3**, provides a powerful way to count and/or tote up numerical solutions, using arbitrary Prolog logic to compute the components. This predicate can be very useful in the preparation of certain types of statistics, such as the mean and standard deviation, which require
the counting of solutions, as well as the toting up of their values and the squares of values. Consider the following database relation:

```prolog
foo( 9 ).
foo( 2 ).
foo( 27 ).
foo( 81 ).
```

The following call counts up the total number of solutions and the total of those solutions in order to calculate the arithmetic mean:

```prolog
?- toteall( (One,Val), (One=1,foo(Val)), (Count,Total) ), Mean is Total/Count.
One = _ ,
Val = _ ,
Count = 4 ,
Total = 119 ,
Mean = 29.75
```

**Enhanced Help Subsystem**

The help files in WIN-PROLOG 4.900 have been considerably extended compared to previous releases, and in addition to the original commentary, now include example calls and application notes, along with increased cross-referencing between topics.

**Word Wrap to Window**

Until now, the WIN-PROLOG console window, as well as all and any program windows, have used a virtual line width equivalent to around 4,192 standard Courier 12 characters, meaning that long lines of output (or program data) have required horizontal scrolling of the window in order to be viewed. New in version 4.900 is the ability to wrap text to the window, much as is done in word processing programs. This can be achieved either through a new "Word Wrap ..." item in the "Options" menu, or through a simple predicate, `word_wrap/1`, which can pick up or set the "Word Wrap to Windows" mode:

```prolog
?- word_wrap( Wrap ).
Wrap = 0
```

The above call confirms that word wrap is currently off ("0", zero); the following call sets it to on ("1", one):

```prolog
?- word_wrap( 1 ).
yes
```
Optimising Compiler Support for Meta-Arguments

One of the unique features of the **386-PROLOG** engine, is its support for *meta-arguments*: predicate calls whose number of arguments are not known until runtime. More than just a notational convenience, meta-arguments allow for some very efficient and sophisticated systems code. For example, consider this example:

```prolog
apply( Pred, Args ) :-
    Pred(Args).
```

With "Pred" bound to an atom, for example "sort", and "Args" bound to a list of arguments, the predicate can be applied directly to the arguments without the added step of converting all the data into a call using (=..)/2 (univ):

```prolog
?- apply( sort, [[q,w,e,r,t,y],Sort] ).
Sort = [e,q,r,t,w,y]
```

Unfortunately, the above program used not to work when optimised, owing to a historical shortcoming of the optimising compiler. In version 4.900, the optimising compiler has been modified to provide full support for this useful and efficient feature.

Redefinition of Mod Operator

A small change, but one which may affect some programs: the "mod" operator, used to compute modulo arithmetic in the `is/2` predicate and its family, has been redefined from "300 xfx" to "400 yfx", for consistency with the other multiplication and division operators; meanwhile, a new binary operator, "/" was added to the table with the same priority, as this is an alternative notation for modulus in **WIN-PROLOG**.

Soft Meta-Predicate Declarations

There was a fixed data relation in **WIN-PROLOG**, called `meta_system/2`, which told various system predicates and environment functions, such as `listing/0`, `listing/1` and the debuggers, how to recurse into the meta-call arguments of predicates like `findall/3`, `setof/3` and so on. It has long been wished to make this definition "soft", or user-redefinable, and 4.900, `meta_system/2` is present as a dynamic, multifile user predicate, enabling any predicate with arguments which are metacalls, to be listed with the same full indentation or debugged with the same detail as the original "meta system" predicates, simply by asserting new cases to `meta_system/2`. Similarly, it is now possible to prevent the indentation of listings or nesting of the debugger selectively by retracting one or more of the pre-defined cases.
JSON Features

Support for JSON as been added to WIN-PROLOG 4.900; this data exchange language is used in many applications on the Internet, where the formality and verbosity of XML is considered overkill. Provided as a source code file in the EXAMPLES folder, JSON.PL defines two main predicates, jread/1 and jwrite/1, which let you read and write JSON terms respectively. A third predicate, jprint/1, displays the Prolog equivalent of any depth of nested JSON structure, in a clear indented format.

RGB, HSL and HSV Colours

A new set of example/utility files, to be found in the EXAMPLES folder, provide support for manipulating RGB colours, converting between them and their HSL and HSV equivalents, as well as displaying them in the console and matching for similarity.
New Features in WIN-PROLOG 4.800

Parallel Text Search

The text scan predicate, scan/3, that was introduced in version 4.700, has been substantially improved with the introduction of a parallel text search technology based on a special data structure, called the "Trie" (pronounced "Tree"). Whereas the old version of scan/3 could only search for single-character strings, the new version can search for any number of strings in parallel, including those which share common prefixes, returning the first best match every time.

The new scan/3 predicate is very powerful, performing a parallel search for any number of strings simultaneously, and returning the first, best match. For example, consider an input file containing the nonsense text:

the fool was not fooled by his food fooler

We can search from the start of the file for the first and best match out of the strings, "foo", "fool", and "fooled":

?- scan([`foo`,`fool`,`fooled`], 0, S).
S = `fool`

At first, this might look like the wrong answer: after all, "fooled" was in the list alongside "fool"; however, as described earlier, scan/3 searches for the first and best match: as it reads the file from the beginning, it first encounters the match, "foo"; however, when it looks further ahead, it finds that the next character, "l", allows for the better match, "fool". One more character is read, but this time it is found to be a space, so the first best match, "fool", is returned. Let's make the same call a second time:

?- scan([`foo`,`fool`,`fooled`], 0, S).
S = `fooled`

Much as before, "foo" is quickly found; a peek ahead finds another matching character, "l", suggesting that "fool" is a better match than "foo"; two more looks ahead find "e" and "d" respectively, so the even better match, "fooled", is discovered, and it is this one which is returned. Let's make yet another call:

?- scan([`foo`,`fool`,`fooled`], 0, S).
S = `foo`

The scan/3 predicate's performs its apparent magic using tries, a special form of tree structure which allows an arbitrary number of strings to be represented from their common roots. Its implementation was inspired by the specific requirements of the efficient parsing of Extensible Markup Language (XML).
Resizable Icon Resources

The `gfx_icon_load/3` predicate has been modified to allow the size of icons to be specified at the time of loading, rather than always defaulting to a fixed 32*32 pixel size: doing this required the use of a different Windows API than before, which in turn means icons can only be loaded from icon resource files (.ICO), and no longer from executable (.EXE), dynamic link library (.DLL) and other similar files.

With the advent of later versions of Windows, it became desirable to support smaller and larger icons, as well as icons with different bit depths, and the archaic index parameter was dropped in favour of one which allows the preferred display size to be specified. Many icon files contain multiple images, and `gfx_icon_load/3` will automatically load the best fit for the linear pixel count specified in `Size`, and will further interpolate the image as necessary to obtain the exact size specified.

In conjunction with this change, and the corresponding ability of the "icon()" function to display these icons at their native size, WIN-PROLOG's own icon set has been updated in version 4.800: previously, all icons were drawn in Windows 3.1's icon editor, at 32*32 pixels in 4-bit colour (just 16 indexed colours), and it was felt that the time had arrived to switch to 48*48 pixel icons in full 24-bit colour.

Extended File Attributes

The file attribute predicate, `attrib/2`, has been extended to handle more cases than before. Previously, this predicate could only set or check two file attributes, namely the read-only and hidden bits, and any attempt to set or check the system bit resulted in an error; at the same time, the archive bit was simply masked, and always reset when `attrib/2` was called to change a file's attributes. In WIN-PROLOG 4.800, this predicate can now test, set or clear all relevant attribute bits:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>16'0001</td>
<td>Read-only</td>
</tr>
<tr>
<td>16'0002</td>
<td>Hidden</td>
</tr>
<tr>
<td>16'0004</td>
<td>System</td>
</tr>
<tr>
<td>16'0020</td>
<td>Archive</td>
</tr>
</tbody>
</table>
Recursive File Directory Predicate

A new predicate, \texttt{dir/4}, joins the set of file handling and data predicates, allowing multiple volumes and subfolders to be listed simultaneously, rather than individually, as with the existing \texttt{dir/3} (which is retained unchanged). In fact, \texttt{dir/4} has been present, but undocumented, in a different form for some time, and is used by several of the programming environment menu items, but it has proven so useful, that the decision has been made to formally document it and bring it into the mainstream.

XML Features

Some experimental XML features have been provided with this release of \texttt{WIN-PROLOG}, although they will not officially be integrated until their usefulness and correctness have been tested in the field. Three predicates, \texttt{xml_token/2}, \texttt{xml_entity/4} and \texttt{xml_reference/3} have been defined which, respectively, read an XML file a "token" a time, recursively substitute any entities in that token with supplied definitions, and finally, replace character references with their Unicode values. Using these three predicates, it is simple to write sophisticated readers that can create nested terms from entire XML files, or read them sequentially or even mix the two, reading subterms within a larger file. Such a reader, as well as a corresponding XML writer has been, has been included among the examples.

New Intelligence Server Implementation

The LPA Intelligence Server (IS), which enables Prolog functionality to be embedded in other applications, has been re-implemented to overcome its one main problem, which was a latency of typically 1ms per transaction, corresponding to around 3ms per call. This performance bottleneck was a result of the IS loading Prolog into a separate process space, and using inter-process communication to transfer queries and results between Prolog and the host application.

In the new implementation, Prolog is loaded in-process, meaning the speed of transaction is no longer limited by Windows timeslicing, and is literally as fast as any given processor allows. Speed improvements of hundreds to thousands of times are easily achieved in even the simplest benchmark IS applications.

The application interface (API) of the new IS is identical to that of the old, with one restriction: it is now only possible to attach one Prolog instance to any given process, whereas previously it was possible to load more than one. Meanwhile, the "tickle" parameter, which was used to fine-tune timeslicing performance, is no longer required, since all interactions are now done as direct procedure calls; however, it has been left as an argument to \texttt{LoadProlog()} for reasons of backwards compatibility.

Apart from the vastly increased speed of the new IS, those applications which wish to share data structures between their native language and Prolog have another huge advantage: because everything is now in-process, Prolog can directly access arrays and other structures whose addresses are passed in, without having to go through the rigmarole of setting up mutexes and memory mapped files.
New Features in WIN-PROLOG 4.700

The most significant new feature in WIN-PROLOG 4.700 is built-in support for the Musical Instrument Digital Interface (MIDI), which opens up a whole new world of experimental research and analysis of music, something to which Prolog's inherent pattern matching is well suited. The predicates are modelled on those used for Windows Sockets (Winsock), and include the following:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mclose/1</td>
<td>close a MIDI device</td>
</tr>
<tr>
<td>mcreate/4</td>
<td>create a MIDI device</td>
</tr>
<tr>
<td>mdata/4</td>
<td>return information about a MIDI device</td>
</tr>
<tr>
<td>mdict/2</td>
<td>return a dictionary of MIDI devices</td>
</tr>
<tr>
<td>midhdl/2</td>
<td>convert between a MIDI device and handle</td>
</tr>
<tr>
<td>mrecv/3</td>
<td>receive data from a MIDI device</td>
</tr>
<tr>
<td>msend/3</td>
<td>send data to a MIDI device</td>
</tr>
<tr>
<td>mstat/3</td>
<td>return the status of a MIDI device</td>
</tr>
<tr>
<td>mtime/2</td>
<td>return or test time of a MIDI device</td>
</tr>
<tr>
<td>midi_handler/2</td>
<td>set or get the handler for a MIDI device</td>
</tr>
<tr>
<td>midi_handler/3</td>
<td>default midi handler</td>
</tr>
</tbody>
</table>

A series of new error numbers has been added to support MIDI:

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>MIDI Handling Error - usually caused by calling a MIDI device predicate with an invalid MIDI device name</td>
</tr>
<tr>
<td>1000 upwards</td>
<td>MIDI errors signalled by WINMM.DLL</td>
</tr>
</tbody>
</table>

Windows Vista and Help

With the advent of Microsoft's latest operating system, Windows Vista, system-level support for traditional 32-bit help files has finally been withdrawn, meaning that applications relying on this method of providing context sensitive help now generate an unhelpful dialog box instead! Although a version of WINHLP32.EXE can be downloaded from Microsoft to support legacy applications, this approach is deprecated by the software giant, who further prohibi
independent vendors such as LPA from distributing this file themselves.

The result of the above is that **WIN-PROLOG** 4.700 now supports context sensitive help using your web browser: each help topic is now held within a single self-contained HTML file, and the collection of these help files is held in a new directory, "HELP", within the **WIN-PROLOG** home directory. The files have apparently random names, although these are actually 8-character hashes of the help topic title.

In order to obtain context sensitive help, simply highlight a predicate call or definition, and press <F1>, just as before. Alternatively, you can call up help, if present, with a new `help/1` predicate; for example:

```prolog
| ?- help( fwrite/4 ).
```

will display help on the `fwrite/4` predicate. In principle, any Prolog term can have help associated with it, although the precompiled HTML files relate mainly to the **WIN-PROLOG** predicates, messages, window styles and errors. You can even create your own help files, simply by writing some HTML (or even plain text) about a predicate, and saving the resulting file in the HELP folder. In order to name the file, you should use the new `help/2` predicate:

```prolog
| ?- help( foo/1, File).
File = 'c:\program files\win-prolog 4.700\help\gpw114it.htm'
```

This takes as its first argument, any non-variable term, and returns a fully-qualified file name for help about that term. If the term is a predicate description (Name/Arity), then it will automatically be included subsequently in <F1> context help.

While there is a possibility that the name returned by `help/2` will clash with an existing file, the probability is minuscule: the hashing used generates names using eight base-36 digits, so there are $36^8$ possible names, which is quite a large number:

```prolog
| ?- X is 36 ^ 8 .
X = 2821109907456
```

The "birthday attack" theory states that the number of entities which must coexist before the chances of two of them having clashing values, is the square root of the number of possible values; here this means that well over a million help topics could exist before there was an odds-on chance of a clash of file names:

```prolog
| ?- X is 36 ^ 4 .
X = 1679616
```

If, by some extreme chance, you do happen to stumble across a predicate description of your own which turns out to clash with one of the existing help file, please let us know!
Finally, the help/3 predicate, which relied upon the functionality of the old WINHLP32.EXE program, has been removed. If, for some reason, you were using this predicate, don't despair: a new source code library file, 46_HELP.PL, has been written to perform this function. Just remember, though, you will have issues with Windows Vista if you use it!

About Time

As computers' central processing units (CPUs) continue to leap forwards in terms of speed, roughly doubling in throughput every 18 months or so (a side-effect of the so-called "Moore's Law"), the one-millisecond minimum time increment of WIN-PROLOG's time structures and internal timers had become insufficient for fine-resolution timing. While it would have proven far too disruptive to change this data structure and the predicates to which it relates, an independent, new time feature has been added to WIN-PROLOG 4.700 to allow for maximum-resolution timings.

The time/2 predicate has two new calls, as illustrated in this code fragment:

```prolog
time( Goal ) :-
    time( Base, Freq ),
    Goal,
    time( Base, Tick ),
    Time is Tick/Freq,
    writeq( Goal ),
    write( ` took ` ),
    write( Time ),
    write( ` seconds~M~J` ).
```

The first call to time/2 passes in two variables; the first, "Base", is bound to a data structure containing a 64-bit, extremely high resolution time reference, while the second, "Freq", returns the frequency of the timer in Hertz (Hz, counts per second). In this example, a specimen "Goal" is executed, after which a second call is made to time/2. This time, with "Base" already bound, the predicate returns an elapsed tick count, binding it to "Tick". The simple arithmetic expression, "Tick/Freq", is then used to calculate time to extreme accuracy, as shown in this transcript:

```
---------------------------------------------------
LPA WIN-PROLOG 4.700 - S/N 0014378400 - 04 May 2007
Copyright (c) 2007 Logic Programming Associates Ltd
Licensed To: LPA Development and Documentation Team
B=64 L=64 R=64 H=256 T=2048 P=8192 S=64 I=256 O=256
---------------------------------------------------
| ?- time( beep(440,1234) ).
beep(440,1234) took 1.23308637422686 seconds
```
Typically, even on old hardware, the timer resolution is at least 1,193,180 Hz, which was based on the frequency of an oscillator chip in the original IBM PC, used to derive serial port baud rates and beeper tones; modern machines have timers running many times faster than this.

**The arg/3, functor/3 and =../2 (univ) Predicates**

These three predicates are used to support programmable access to compound terms, which in many traditional Prolog implementations, are stored in array-like tuple. In **WIN-PROLOG**, direct pattern-matching access is supported, making them somewhat redundant other than for compatibility with other implementations.

One aspect of this compatibility relates to the handling of certain functor names, such as ‘.’, which is used as a list constructor in some Prolog implementations. For example:

```
.(Head,Tail)
```

is considered, by some implementations, to be identical in every respect to the term:

```
[Head|Tail]
```

In **WIN-PROLOG**, list are represented by a highly optimised, independent data type, so the following call will fail:

```
| ?- .(Head,Tail) = [Head|Tail].
no
```

However, up to version 4.600, **WIN-PROLOG**’s arg/3, functor/3 and =../2 predicates tried to "pretend" that this "dot" notation existed, and this led to numerous confusing inconsistencies. For example, although the above call failed (and still does), in **WIN-PROLOG** 4.600 and earlier:

```
| ?- functor( .(_,_), Pred, Arty ).
Pred = '.',
Arty = 2
```

and yet:

```
| ?- functor( Term, ., 2 ).
Term = [_123|_456]
```
In **WIN-PROLOG** 4.700, special handling of "dot" notation has been removed, so that we get as before:

```
| ?- functor( .(_,_), Pred, Arty ).
Pred = '.',
Arty = 2
```

but now, more consistently:

```
| ?- functor( Term, ., 2 ).
Term = '.(_123,_456)
```

It is not envisaged that this change in behaviour will cause problems to the majority of modern Prolog code, and although it might result in errors with some very early academic examples, it was felt that the benefits of consistency of behaviour between this predicates and the underlying unification code and **WIN-PROLOG** architecture far outweighs any such worries.

### Modifications to LZSS and MZSS Checksums

With a view to improving security and file integrity, two independent modifications were made to the MZSS encryption routine, and one of these was also made to the LZSS compression routine. Both routines now use CRC32, rather than a simple rotating checksum, to check for file integrity, and this will result in an error if compressed or encrypted files from **WIN-PROLOG** 4.600 or earlier are processed in **WIN-PROLOG** 4.700. Our recommendation is to convert your old compressed and encrypted files back to plaintext using **WIN-PROLOG** 4.600, and then compress and encrypt them afresh in **WIN-PROLOG** 4.700. If this proves to be a problem, please let us know: we will develop and furnish a simple tool to perform this operation if required.

The second modification, which applies to MZSS encryption only, provides better apparent randomization of files created in close temporal proximity. As documented, MZSS combines your secret password with a public, time-derived string, and uses the combined data to initialise the comb-filtered MZ random number generator. Although the algorithm was always and is still very secure, if two files were encrypted within a short space of time, the time-derived component would be similar, and anyone comparing two files would be able to spot the similarities in this 8-byte strings, and furthermore, to deduce the time and date at which the file was encrypted.

Starting with **WIN-PROLOG** 4.700, a much more random 256-bit component is combined with the secret password, resulting in files which appear totally random with no predictable components: this has no effect on the ability to encode or decode older files, but does explain why newly encoded files are 24 bytes longer than previously, and breaks any direct link between system time and the public string.
Removal of Self and Parent from Directory Listings

One long-term bugbear of Windows programming, which owes its existence to MS-DOS, is the presence of two "ghost" directories in every folder, namely "." for "self", and "." for parent. In WIN-PROLOG 4.600 and earlier, these could be seen as the first two elements in the returned list in a simple call such as:

```
| ?- dir( *,.*, 0, Dirs ).
Dirs = ['.',..,'SYSTEM','LIBRARY','EXAMPLES','PRO386W.EXE' etc.
```

Starting in WIN-PROLOG 4.700, these two ghost directories are automatically filtered out of all listings, so that:

```
| ?- dir( *,.*, 0, Dirs ).
Dirs = ['SYSTEM','LIBRARY','EXAMPLES','PRO386W.EXE' etc.
```

This considerably simplifies programs which handle nested directories, which formerly had to be aware of the ghosts, which were present in all directories apart from the root: now, everything returned by \texttt{dir/3} comprises a genuine directory or file name.

Recursive Directory Listings

A new predicate, \texttt{dir/4}, was added to WIN-PROLOG 4.700, to allow for recursive directory listings. Taking a pair of arguments, the root directory and a file mask pattern, as well as an attribute list, it returns the list of all matching directories and/or files both within and beneath the directory specified in the first argument. As with \texttt{dir/3}, the "self" and "parent" directories are filtered out.

Improvements to Windows Sockets (Winsock)

Some improvements were made to the Windows Sockets (Winsock) predicates, and to message and event handling, in order to simplify TCP/IP programming, and to remove the need for frequent "safety" calls to the socket status predicate.

The Winsock predicates now have a universal "fail it not ready" behaviour: previously, some predicates worked like this, but others generated an error condition.

Events now occur automatically, as required, after every single send or receive operation, if more data may be sent or is available to read, as appropriate. This means it is no longer necessary to loop until failure with successive calls to \texttt{ssend/2} or \texttt{srecv/2}: a single call will do, generating another event if more data can be processed.

These improvements do not undermine existing Winsock code, but rather make certain calls redundant in situations where they were formerly required;
the Winsock example programs have been updated accordingly.

**GraFiX Window Default Font**

The default font used in GraFiX Windows has been changed, from the "ansi_var_font", to the "prolog_fixed_font", as used, by default, in the console window. This will affect the default appearance of text displayed in calls to gfx/1, but has no effect on graphics where explicit fonts have been specified.

**Multi-Character Text Scan Predicate**

A new predicate, scan/3, has been added to WIN-PROLOG 4.700 to complement the existing find/3, and provide an alternative way to scan for text within a file or other input stream. While find/3 searches for a specific string, with optional case sensitivity and/or output of mismatched characters, scan/3 looks for any one of the characters in a specified string, again with case sensitivity and/or output options. For example, consider the call:

```
| ?- scan( `abc`, 2, Scan ) <- `the quick brown fox` ~> Text.
  Scan = `c`,
  Text = `the qui`
```

This call scans for each of the letters, "a", "b" and "c", returning whichever of these was found first within the string, "the quick brown fox", bound to "Scan". In addition, the flag, "2", indicates that unmatched characters should be output, and these have accumulated in the variable, "Text".

The scan/3 predicate is designed to help with the scanning of any type of structured file, including HTML, XML, RTF and plain text.

**Increase in Maximum Atom Length**

Although the previous limit of 1024 bytes was considered enough for any single atom, with the advent of Unicode, and more especially, 32-bit characters, this meant that a worst-case atom, in which all characters were large 32-bit values, would be limited to a mere 204 characters' length. In WIN-PROLOG 4.700, the maximum length of an atom has been increased to 8192 bytes, which can handle, at worst, 1638 characters, and, of course, at best, up to 8192 characters.

A curious side-effect of this change is that formatted input and output (fread/4 and fwrite/4) can now handle fields of up to 8192 characters, up from their previous limit of 1024, and the tab/1 predicate and others which use formatted I/O, have similarly expanded capabilities.

Furthermore, the console input buffer, which is used to contain the text of partially read terms in between timer and message interrupts, has been increased in size from its former 4096 bytes to 8192 bytes, to match the maximum size of an atom.
Single Key Shortcuts

As a small but helpful new feature, the WIN-PROLOG 4.700 Development Environment includes a number of commonly-used single-key shortcuts, which are based on those used by other well-known applications. Now, for example, you can create a new window by pressing <CTRL-N>, or save the file you're working on with <CTRL-S>. The shortcuts are listed in the menus, so they can be learned gradually as you work with the system.

Plain Text Command History File

While WIN-PROLOG 4.600 introduced the concept of a persistent command history, which would retain your console commands between sessions, this was stored in an encrypted file which made it impossible to edit manually, or indeed to vet for sensitive information.

In WIN-PROLOG 4.700, this has been replaced by a plain-text file, PRO386W.HST, which coexists with PRO386W.EXE in the WIN-PROLOG home directory. The consists of a simple, formatted list of strings, each one relating to a single command. Individual lines can be added, removed or simply edited, using a standard text editor, and it is easy to see if sensitive information is present within the file.

VisiRule 1.5: Greatly Improved Graphics and Interaction

A lot of work has been put into LPA's visual programming tool for Business Rules and Decision Support, with enhanced graphics and simplified interaction. Existing VisiRule charts can be imported into the new version, which more than halves the number of mouse clicks and/or tool changes required to draw and edit charts.

Documentation Files

The documentation for WIN-PROLOG 4.700 and its toolkits is spread across a large number of Adobe Acrobat .PDF files; the following table gives the names of these files and describes their contents:

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>WELCOME.PDF</td>
<td>Welcome and Update Notes</td>
</tr>
<tr>
<td>CBR_REF.PDF</td>
<td>Case-based Reasoning</td>
</tr>
<tr>
<td>CHI_REF.PDF</td>
<td>Chimera Programming Guide and Tech Ref</td>
</tr>
<tr>
<td>DTM_API.PDF</td>
<td>Data Mining Toolkit</td>
</tr>
<tr>
<td>FLN_REF.PDF</td>
<td>Flint Reference</td>
</tr>
<tr>
<td>FLX_EGS.PDF</td>
<td>flex Examples</td>
</tr>
<tr>
<td>FLX_REF.PDF</td>
<td>flex Reference</td>
</tr>
</tbody>
</table>
New Features in WIN-PROLOG 4.600

The most significant new feature in WIN-PROLOG 4.600 is built-in support for Windows Sockets (Winsock), which opens up a whole new world of TCP/IP based programming, including Internet and local network resource access, agent-based distributed applications, and much more. The predicates which support sockets at a low level include:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>sclose/1</td>
<td>close a named socket</td>
</tr>
<tr>
<td>sckhdl/2</td>
<td>convert between a named socket and its raw numerical handle</td>
</tr>
<tr>
<td>screate/2</td>
<td>create a named socket</td>
</tr>
<tr>
<td>sdict/2</td>
<td>return a dictionary of named sockets</td>
</tr>
<tr>
<td>srecv/2</td>
<td>receive data from a named socket</td>
</tr>
<tr>
<td>ssend/2</td>
<td>send data to a named socket</td>
</tr>
<tr>
<td>sstat/2</td>
<td>return the status of a named socket</td>
</tr>
<tr>
<td>socket_handler/2</td>
<td>set or get the handler for a named socket</td>
</tr>
<tr>
<td>socket_handler/3</td>
<td>default socket handler</td>
</tr>
</tbody>
</table>

A series of new error numbers has been added to support Winsock:

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Socket Handling Error - usually caused by calling a socket predicate with an invalid socket name</td>
</tr>
<tr>
<td>10000-11000</td>
<td>Winsock errors signalled by WSOCK32.DLL</td>
</tr>
</tbody>
</table>

Message Digest Five (MD5) Support

Partly with a view to supporting certain Internet protocols, and to round out the set of data checksum and hashing predicates, the MD5 algorithm is now supported. These are the three data checking predicates, two of which were introduced in WIN-PROLOG 4.320:
Extended End-of-Line Parsing

Also with a view to TCP/IP processing, and the increased likelihood of encountering text files using the Unix end-of-line convention (<LF>, rather than the more familiar <CR/> of Windows), the formatted read predicate, fread/4, was modified to recognise this case in addition to the Windows convention.

Adjustable Windows API Buffer Size

Some extreme applications of the Windows API predicates, winapi/4 and wintxt/4, were failing because of insufficient space to pass very large text strings or data structures, because previous versions of WIN-PROLOG used a fixed 64kb buffer for this purpose. In version 4.600, the size of this buffer can be queried and changed at any time:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>winsze/1</td>
<td>get or set the Windows API buffer size</td>
</tr>
</tbody>
</table>

Further Improved Mouse Handling

Further improvements to mouse handling, which had already been extended in WIN-PROLOG 4.500, were introduced in 4.600, including the ability to detect mouse clicks in all window types (previously, they were limited to GraFiX windows):

<table>
<thead>
<tr>
<th>Message</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_leftdown</td>
<td>left button down</td>
</tr>
<tr>
<td>msg_leftup</td>
<td>left button up</td>
</tr>
<tr>
<td>msg_leftdouble</td>
<td>left button double click</td>
</tr>
<tr>
<td>msg_rightdown</td>
<td>right button down</td>
</tr>
<tr>
<td>msg_rightup</td>
<td>right button up</td>
</tr>
<tr>
<td>msg_rightdouble</td>
<td>right button double click</td>
</tr>
</tbody>
</table>
msg_wheeldown  wheel button down
msg_wheelup    wheel button up
msg_wheeldouble wheel button double click

Reduced Wait Loop Latency

Some applications which relied on a "repeat/wait/fail" loop to process events ran slightly slowly, because the wait/1 predicate would pause for between 10ms and 55ms depending upon the version of Windows. In WIN-PROLOG 4.600, some low level work has enabled this delay to be reduced to 1ms on all versions of Windows, without imposing any noticeable CPU overhead. This improvement also speeds up output to the console window, especially from asynchronous processes such as message handlers, timers and so on.

Maximised MDI Windows at Startup

As a small visual improvement, WIN-PROLOG 4.600 now opens its main and MDI windows to make maximum use of the available desktop space: previously, these windows opened in a "default" mode which meant they could be of various sizes and in various locations in successive runs of the system.

Single-Key Console Window Focus

Finally, WIN-PROLOG 4.600 extends the functionality of the <ctrl-Q> key, which has been used to highlight the console window's Input Zone since version 4.300: now, this key combination will automatically bring the console window into focus, providing a quick shortcut for getting back to the command prompt while working in the WIN-PROLOG development environment.

Features Removed from WIN-PROLOG 4.600

Two predicates, which were really only needed by the WIN-PROLOG development environment, have been removed, although in the interests of compatibility, both are supported through library files:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>abtbox/4</td>
<td>display the WIN-PROLOG about box (see LIBRARY\45_ABTBX.PL)</td>
</tr>
<tr>
<td>sttbox/2</td>
<td>display the WIN-PROLOG status box (see LIBRARY\45_STTBX.PL)</td>
</tr>
</tbody>
</table>
Chimera - Agents for WIN-PROLOG

A brand new toolkit, Chimera builds on the Winsock functions in WIN-PROLOG 4.600 to provide a highly flexible environment for developing distributed agent applications. Chimera replaces the previous TCP/IP and Agent Toolkits, and ships with many of the previous example programs, as well as some new ones. Chimera’s design is very clean and simple, and involves none of the complex setting up of the previous toolkits; moreover, its use of named agents means that any number of agents can run, simultaneously, in a single instance of WIN-PROLOG, further simplifying the development and testing of agent applications.

VisiRule 1.1 - Improved Visual Programming

LPA’s visual programming tool for Business Rules and Decision Support has just got better, with a considerably streamlined interface and even better code generation for flex. Existing VisiRule charts can be imported into the new version, which more than halves the number of mouse clicks and/or tool changes required to draw and edit charts.

Flint - Modifications and Additions

Fuzzy logic has become even more interesting, with assorted modifications and additions to the Flint toolkit.
Technical Support

Please check the Support section of the LPA website for latest updates and reports on bug fixes, and direct all email correspondence to the appropriate address as shown in table below, making sure you include the following details:

1) A cut-and-pasted copy of your WIN-PROLOG "welcome banner", for example:

---------------------------------------------------
BDS WIN-PROLOG 7.000 X86 S/N 0000000000 01 Mar 2019
Copyright 1989-2019 Brian D Steel (www.solanum.org)
This Special Version Is For The Personal Use Of BDS
B=64 L=64 R=64 H=256 T=2048 P=8192 S=64 I=256 O=256
---------------------------------------------------

You can create this banner at any time during a Prolog session simply by typing the command:

?- ver(1).

2) Brief details about your:

<table>
<thead>
<tr>
<th>Subject</th>
<th>For Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>98SE, ME, 2000, XP, Vista, 7, 8, 10, 32/64 bit</td>
</tr>
<tr>
<td>Service Pack</td>
<td>None, SP1, SP2</td>
</tr>
<tr>
<td>Processor</td>
<td>Pentium, Core Solo/Duo/Quad, Xeon, i7</td>
</tr>
<tr>
<td>Memory/Disk Size</td>
<td>RAM, Cache, Hard Disk</td>
</tr>
<tr>
<td>Browser</td>
<td>Firefox, Safari, Chrome, Opera, Edge</td>
</tr>
<tr>
<td>Computer Make</td>
<td>HP, Lenovo, Dell, Apple</td>
</tr>
</tbody>
</table>

3) Description of the problem, including the exact text of any error message or error code that is displayed

Please DON'T send email attachments (.ZIP, .EXE, .DOC, .BMP, etc) unless we have specifically asked for them: we simply delete unsolicited files without opening them.
## Contact Details

<table>
<thead>
<tr>
<th>Information:</th>
<th><a href="mailto:info_team@lpa.co.uk">info_team@lpa.co.uk</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales:</td>
<td><a href="mailto:sale_team@lpa.co.uk">sale_team@lpa.co.uk</a></td>
</tr>
<tr>
<td>Support:</td>
<td><a href="mailto:tech_team@lpa.co.uk">tech_team@lpa.co.uk</a></td>
</tr>
<tr>
<td>Phone:</td>
<td>+44 (0) 20 8871 2016</td>
</tr>
<tr>
<td>Web:</td>
<td><a href="http://www.lpa.co.uk">http://www.lpa.co.uk</a></td>
</tr>
</tbody>
</table>

Logic Programming Associates Ltd  
PO Box 226, Cranleigh, Surrey, GU6 9DL, England

Thank you for purchasing **WIN-PROLOG 7.000** - please enjoy your software!