4.900

Welcome and Update Notes

Brian D Steel

WIN-PROLOG Welcome and Update Notes

The contents of this manual describe the product, **WIN-PROLOG**, version 4.9, and are believed correct at time of going to press. They do not embody a commitment on the part of Logic Programming Associates Ltd (LPA), who may from time to time make changes to the specification of the product, in line with their 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 LPA.

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)

Logic Programming Associates Ltd
Studio 30
The Royal Victoria Patriotic Building
Trinity Road
London SW18 3SX England

phone: +44 (0) 20 8871 2016
fax: +44 (0) 20 8874 0449
email: support@lpa.co.uk
web: http://www.lpa.co.uk

Welcome to WIN-PROLOG 4.9!

Contents

Welcome to *WIN*-PROLOG 4.900

Welcome to the latest version of ***WIN*-PROLOG**, a mature and distinguished Prolog compiler and runtime support system for Microsoft Windows, which is accompanied by many optional toolkits and plug-ins to suit many types of application. Built upon the innovative ***386*-PROLOG** engine, designed and written in 1989 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, 12 May 10

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 enconding 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 retuns the UTF-16LE enconding 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 funcionality 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:

```
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:

```
?- 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:

```
?- 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):

```
?- 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:

```
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):

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

Unfortuntely, 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 beed wished to make this definition "soft", or user-redefinable, and 4.900, *meta_system/2* is present as a dynanic, 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 to 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.

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.

*Old 32*32 pixel, 4-bit colour icons*

*New 48*48 pixel, 24-bit colour icons*

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 releavant attribute bits:

Value	Meaning
16'0001	Read-only
16'0002	Hidden
16'0004	System
16'0020	Archive

Recursive File Directory Predicate

A new predicate, *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 *dir/3* (which is retained unchanged). In fact, *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 ***WIN-PROLOG***, although they will not officially be integrated until their usefulness and correctness have been tested in the field. Three predicates, *xml_token/2*, *xml_entity/4* and *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 LoadProlog() for reasons of backwards compatibility.

Apart from the vastly increased speed of the new IS, those applications which wish to share data strcutures 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:

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

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

Error	Meaning
16	MIDI Handling Error - usually caused by calling a MIDI device predicate with an invalid MIDI device name
1000 upwards	MIDI errors signalled by WINMM.DLL

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 prohibit

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:

```
| ?- 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:

```
| ?- 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:

```
| ?- 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:

```
| ?- 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 strucures 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:

```
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
```

```
yes
```

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 compatibity 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 *dir/3* comprises a genuine directory or file name.

Recursive Directory Listings

A new predicate, *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 *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 *ssend/2* or *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:

File	Contents
WELCOME.PDF	Welcome and Update Notes
CBR_REF.PDF	Case-based Reasoning
CHI_REF.PDF	Chimera Programming Guide and Tech Ref
DTM_API.PDF	Data Mining Toolkit
FLN_REF.PDF	Flint Reference
FLX_EGS.PDF	flex Examples
FLX_REF.PDF	flex Reference

FLX_TUT.PDF	flex Tutorial
INT_REF.PDF	Intelligence Server
PDI_REF.PDF	ProData Interface
PPP_REF.PDF	Prolog++ Reference
PWS_REF.PDF	ProWeb Server User Guide
VSR_REF.PDF	VisiRule User Guide
VSR_TUT.PDF	VisiRule Tutorial
WFS_REF.PDF	WebFlex Server User Guide
WIN_EGS.PDF	***WIN-PROLOG*** Prolog Examples
WIN_PRG.PDF	***WIN-PROLOG*** Programming Guide
WIN_REF.PDF	***WIN-PROLOG*** Technical Reference
WIN_TUT1.PDF	Prolog Tutorial 1
WIN_TUT2.PDF	Prolog Tutorial 2
WIN_TUT3.PDF	Prolog Tutorial 3
WIN_TUT4.PDF	Prolog Tutorial 4
WIN_USR.PDF	***WIN-PROLOG*** User Guide

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:

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

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

Error	Meaning
15	Socket Handling Error - usually caused by calling a socket predicate with an invalid socket name
10000-11000	Winsock errors signalled by *WSOCK32.DLL*

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:

Predicate	Function
mdf/3	perform an MD5 message digest
crc/3	perform a CRC-32 cyclic redundancy check (4.320)
sha/3	perform an SHA-256 secure hash algorithm (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/LF> 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:

Predicate	Function
winsze/1	get or set the Windows API buffer size

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:

Message	Function
msg_leftdown	left button down
msg_leftup	left button up
msg_leftdouble	left button double click
msg_rightdown	right button down
msg_rightup	right button up
msg_rightdouble	right button double click

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:

Predicate	Function
abtbox/4	display the ***WIN-PROLOG*** about box (see *LIBRARY\45_ABTBX.PL*)
sttbox/2	display the ***WIN-PROLOG*** status box (see *LIBRARY\45_STTBX.PL*)

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.

Chimera Agents for WIN-PROLOG

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:

```
-------------------------------------------------
LPA WIN-PROLOG 4.800 - S/N 0015043680 - 08 Aug 2008
Copyright (c) 2008 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
-------------------------------------------------
```

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

```
?- ver(1).
```

2) Brief details about your:

Subject	For Example
System	Windows 98, ME, NT, 2000, 2003, XP, Vista 32, Vista 64
Service Pack	None, SP1, SP2
Processor	386, PII, PIII, P4, Core Solo/Duo/Quad, Xeon, Athlon
Memory/Disk Size	RAM, Cache, Hard Disk
Browser	NetScape 7, Internet Explorer 6, Opera 9
Computer Make	Compaq, HP, Lenovo

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

Information:	info_team@lpa.co.uk
Sales:	sale_team@lpa.co.uk
Support:	tech_team@lpa.co.uk
Phone:	+44 (0) 20 8871 2016
Fax:	+44 (0) 20 8874 0449
Web:	http://www.lpa.co.uk
Logic Programming Associates Ltd Studio 30, Royal Victoria Patriotic Building Trinity Road, London, SW18 3SX, England	

Thank you for purchasing *WIN-PROLOG* 4.800 - please enjoy your software!