Subject: IDLWAVE 4.13 Release (idlwave.org)
Posted by JD Smith on Wed, 15 May 2002 00:41:51 GMT
View Forum Message <> Reply to Message

After a long hiatus, I'm happy to announce a substantial new release of IDLWAVE, version 4.13, with a host of great new features, bug fixes and improvements:

1. Security improvements for temporary file creation.

2. New convenience routines for defining your own abbreviations, e.g.:

```
  (idlwave-define-abbrev "wb" "widget_base()"
                         (idlwave-keyword-abbrev 1))
  (idlwave-define-abbrev "on" "obj_new()"
                         (idlwave-keyword-abbrev 1))))
```

   and several new handy preset abbreviations, including:

```
    \iap    "if arg_present() then"
    \ap     "arg_present()"
    \ik     "if keyword_set() then"
```

3. Bug fixes:

   i) When splitting strings, the point was ending up on the initial apostrophe. Fixed, saving one keypress.

   ii) Fixed keyword completion in continued function statements, when the entire function is part of a larger continued statement. E.g., in:

```
        if a then $
          b=test(c,d, $
               f,g) $
        else return
```

   near `f', IDLWAVE was completing functions, and not keywords to `test()'.

   iii) Procedures on multi-statement lines (with `&') are now correctly recognized as such for completion and help.

   iv) Completion of Class::Init keywords inside of obj_new("Class"...) statements failed for routine completion-case set to anything other than 'preserve or 'upcase. Fixed, for the rare user of 'capitalize.

4. Improved mult-line comment handling. Multiple line comments are often a crucial part of large structure definitions, or complex routine calls. They are now properly counted as continuation lines, relieving the need to insert a placeholder `$' character on blank or comment-only lines just to get things to line up, e.g.:

```
      st={element1:b, $    ;this important element
                           ;requires multiple lines of
                           ;explanation
          element2:c, $    ;this one requires one

          element3:d, $    ;and this one is separated
                           ;by a blank line.
         }
```

This should make long continued statements much more attractive, and perhaps even encourage more people to document their info structures and class definitions!

5. Improved continuation statement indentation, which attempts to line up on the opening punctuation in routine definitions and procedure calls, lists, structures, function calls and other parenthesis pairs, as well as continued assignment statements. The following admittedly contrived example demonstrates intelligent alignment of multiple nested continuation types, and also highlights multi-line and in-statement comments:

```
  var=myfunc(a,b,$
             c,mysubfunc(x,y, $
                         z), $
             d,p,d,q,[12,13,14, $
                      15,16,17]) + !PI + $
      ;; Another very nice function
      ;; to add to var
      myfunc2( $
              {SOMESTRUCT, $
               a:1, $           ;a is a very nice tag
                                ;which has an unfortunately
                                ;long comment
               b:2} $
             ) + $
      ;; The old value, and answer to everything
      oldvar + 42
```

Recognizing the age-old dilemma of prettiness vs.

dwindling-space, you can opt to revert to basic, fixed-offset
indentation if the intelligent indent moves over too far, by
setting the configurable limit in variable
`idlwave-max-extra-continuation-indent'. This variable can
be set to 0 to recover fixed-offset only indentation, or set
to a large number (like 100) to use intelligent indents only.
Note that the variable `idlwave-indent-to-open-paren', if set
(as it is by default), overrides this limit for paren groups.

N.B. THE CONTINUATION INDENTATION DEFAULT BEHAVIOR HAS
CHANGED. TO RECOVER THE OLD BEHAVIOR, SET
`idlwave-max-extra-continuation-indent' TO 0.

Also, since the intelligent indentation update is more
dynamic and changes on one line can impact the entire
statement, a new command on [C-u TAB] is available to
re-indent the entire multi-line statement at once. This is
very useful even without dynamic indentation, e.g. for long
structure assignments.

6. New macros are available for creating custom "examine"
commands, similar to the print [C-c C-d C-p], mouse-print
[S-mouse-2], help [C-c C-d C-?] and mouse-help (now bound on
[M-C-mouse2]) already available. There is one command for
creating custom mouse-based examinations, and one for
keyboard-based. They have all the nice properties of the
built-in "print," and "help," examine routines, including the
ability to pull expressions from higher up on the calling
stack (one of my favorite debugging features), finding the
nearby or enclosing relevant expression (on key press or
single click), or examining the highlighted expression (on
mouse drag). They are simple to create in the user's
`idlwave-shell-hook', e.g.:

```
   (idlwave-shell-define-key-both [s-down-mouse-2]
                  (idlwave-shell-mouse-examine
                   "print, size(___,/DIMENSIONS)"))
```

makes a mouse binding for Super with the middle mouse button
in both the shell and the buffer, which prints the dimensions
of the expression. (The "___" is replaced by the chosen
expression). The possibilities are endless, even if your
keyboard isn't.

7. A new super examine command (bound on [C-S-mouse2], which was
previously mouse-help) expands this concept even further. It
works just like the regular mouse examine commands, but
instead of sending a hard-coded function to the shell, it

pops-up a customizable list of examine functions to choose among. Though it comes with a decent default set of commands, you can add to or change this list to get simple tailored pop-up examine commands, without writing a single line in your .emacs file! See customize item "Idlwave Shell General Setup"->"Idlwave Shell Examine Alist".

8. All examine commands now work equally well in both the buffer and the shell. Traditionally they have been used in the buffer for debugging, to inspect variables while at a break point or error; however, they are also extremely handy in other contexts. For instance, when composing long statements in the shell, you often have to reconfirm the size or type of one of the variables or function outputs being used. With the examine commands, you can do this without leaving the statement or interrupting the process of building it (point is left in place). These commands are also useful for printing variables from prior input or output further back in the shell buffer.

9. Both the examined expression and the output of the examination command are now simultaneously highlighted, and you can configure the highlight faces separately. In addition, by default the output of all examine commands is sent to a special *Examine* buffer, which keeps the shell from getting cluttered and stores old examine output in a convenient location. If you prefer the older behavior (examine commands and output visible in the shell), you can set the new custom variable `idlwave-shell-separate-examine-output'.

10. Several corrections and updates were made to the online help and routine information (including two found by none other than Richard M. Stallman).

Thanks to Dick Jackson and for his suggestions and patient testing, and of course Carsten for his continued assistance and insight. As always, you can find the latest version of IDLWAVE at http://idlwave.org.

Happy coding!

JD

P.S. If you don't find the new version on the site, try again in a few hours: the server has been switched, and the DNS updates may still be propagating.