Subject: IDLWAVE 4.7/Tutorial
Posted by dominik on Fri, 08 Dec 2000 13:18:49 GMT
View Forum Message <> Reply to Message

Hi, I have release IDLWAVE 4.7. Sorry that this happens so quickly after 4.6, but the recent discussion here has prompted a new way of assigning keys to debugging commands which I would like to get out now.

Also, JD and I have been cooking up a tutorial which was requested by several contributions here. While it may not quite be simple enough for David ;-), I hope it will be OK for most people who have used Emacs before. Feedback on the Tutorial is welcome. The Tutorial should work with 4.6 except in one or two details. The keybinding methods described only work with 4.7.

- Carsten

Getting Started (Tutorial)
****

Lession I: Development Cycle
==========

The purpose of this tutorial is to guide you through a very basic development cycle with IDLWAVE. We will type a simple program into a buffer and use the shell to compile, debug and run this program. On the way we will use the most important commands in IDLWAVE. Note however that there is much more funtionality available in IDLWAVE than we cover here, and it will pay off greatly if eventually you go further and read the whole manual.

I assume that you have access to Emacs or XEmacs with the full IDLWAVE package including online help (*note Installation::). I also assume that you are familiar with Emacs and can read the nomenclature of key presses in Emacs (in particular, `C' stands for <CONTROL> and `M' for <META> (often the <ALT> key carries this functionality)).

Open a new source file by typing

C-x C-f tutorial.pro <RET>

A buffer for this file will pop up, and it should be in IDLWAVE mode. You can see this by looking at the mode line, just below the editing window. Also, the menu bar should contain entries `IDLWAVE' and `Debug'.

Now cut-and-paste the following program, also available as `tutorial.pro' in the IDLWAVE distribution.

```
function daynr,d,m,y
  ;; compute a sequence number for a date
  ;; works 1901-2099.
  if y lt 100 then y = y+1900
  if m le 2 then delta = 1 else delta = 0
  m1 = m + delta*12 + 1
  y1 = y * delta
  return, d + floor(m1*30.6)+floor(y1*365.25)+5
end

function weekday,day,month,year
  ;; compute weekday number for date
  nr = daynr(day,month,year)
  return, nr mod 7
end

pro plot_wday,day,month
  ;; Plot the weekday of a date in the first 10 years of this century.
  years = 2000,+indgen(10)
  wdays = intarr(10)
  for i=0,n_elements(wdays)-1 do begin
     wdays[i] =  weekday(day,month,years[i])
  end
  plot,years,wdays,YS=2,YT="Wday (0=sunday)"
end
```

The indentation probably looks funny, since it's different from the settings you use, so use the <TAB> key in each line to automatically line it up (or more quickly _select_ the entire buffer with `C-x h' followed by `M-C-\'). Notice how different syntactical elements are highlighted in different colors, if you have set up support for font-lock.

Let's check out two particular editing features of IDLWAVE. Place the cursor after the `end' statement of the `for' loop and press <SPC>. IDLWAVE blinks back to the beginning of the block and changes the generic `end' to the specific `endfor' automatically. Now place the cursor in any line you would like to split into two and press `M-<RET>'. The line is split at the cursor position, with the continuation `$' and indentation all taken care of. Use `C-/' to undo the last change.

The procedure `plot_wday' is supposed to plot the weekday of a given date for the first 10 years of the 21st century. I have put in a few bugs which we are going to fix now.

First, let's launch the IDLWAVE shell. You do this with the command `C-c C-s'. The Emacs window will split and display IDL running in a shell interaction buffer. Type a few commands like `print,!PI' to convince yourself that you can work there like in an xterminal, or the IDLDE. Use the arrow keys to cycle through your command history. Are we having fun now?

Now go back to the source window and type `C-c C-d C-c' to compile the program. If you watch the shell buffer, you see that IDLWAVE types `.run tutorial.pro' for you. But the compilation fails because there is a comma in the line `years=...'. The line with the error is highlighted and the cursor positioned at the error, so remove the comma (you should only need to hit Delete!). Compile again, using the same keystrokes as before. Notice that the file is saved for you. This time everything should work fine, and you should see the three routines compile.

Now we want to use the command to plot the weekdays for January 1st. We could type the full command ourselves, but why do that? Go back to the shell window, type `plot_' and hit <TAB>. After a bit of a delay (while IDLWAVE initializes its routine info database), the window will split to show all procedures it knows starting with that string, and `plot_wday' should be one of them. Saving the buffer was enough to tell IDLWAVE about this new routine. Click with the middle mouse button on `plot_wday' and it will be copied to the shell buffer, or if you prefer, add `w' to `plot_' to make it unambiguous, hit <TAB>, and the full routine name will be completed. Now provide the two arguments:

```
plot_wday,1,1
```

and press <RET>. This fails with an error message telling you the `YT' keyword to plot is ambiguous. What are the allowed keywords again? Go back to the source window and put the cursor into the `plot' line, and press `C-c ?'. This pops up the routine info window for the plot routine, which contains a list of keywords, and the argument list. Oh, we wanted `YTITLE'. Fix that up. Recompile with `C-c C-d C-c'. Jump back into the shell with `C-c C-s', press the <UP> arrow to recall the previous command and execute again.

This time we get a plot, but it is pretty ugly - the points are all connected with a line. Hmm, isn't there a way for `plot' to use symbols instead? What was that keyword? Position the cursor on the plot line after a comma (where you'd normally type a keyword), and hit `M-<Tab>'. A long list of plot's keywords appears. Aha, there it is, `PSYM'. Middle click to insert it. An `=' sign is included for you too. Now what were the values of `PSYM' supposed to be? With the cursor on or after the keyword, press `M-?' for online help

(alternatively, you could have right clicked on the colored keyword itself in the completion list). The online help window will pop up showing the documentation for the `PYSM' keyword. Ok, let's use diamonds=4. Fix this, recompile (you know the command by now: `C-c C-d C-c', go back to the shell (if it's vanished, you know the command to recall it by now: `C-c C-s') and execute again. Now things look pretty good.

Lets try a different day - how about April fool's day?

```
plot_wday,1,4
```

Oops, this looks very wrong. All April fool's days cannot be Fridays! We've got a bug in the program, perhaps in the `daynr' function. Lets put a breakpoint on the last line there. Position the cursor on the `return, d+...' line and press `C-c C-d C-b'. IDL sets a breakpoint (as you see in the shell window), and the line is highlighted in some way. Back to the shell buffer, re-execute the previous command. IDL stops at the line with the breakpoint. Now hold down the SHIFT key and click with the middle mouse button on a few variables there: `d', `y', `m', `y1', etc. Maybe `d' isn't the correct type. CONTROL-SHIFT middle-click on it for help. Well, it's an integer, so that's not the problem. Aha, `y1' is zero, but it should be the year, depending on delta. Shift click `delta' to see that it's 0. Below, we see the offending line: `y1=y*delta...' the multiplication should have been a minus sign! So fix the line to

```
y1 = y - delta
```

Now remove all breakpoints: `C-c C-d C-a'. Recompile and rerun the command. Everything should now work fine. How about those leap years? Change the code to plot 100 years and see that every 28 years, the sequence of weekdays repeats.

Lession II: Customization

Emacs is probably the most customizable piece of software available, and it would be a shame if you did not make use of this and adapt IDLWAVE to your own preferences. Customizing Emacs or IDLWAVE means that you have to set Lisp variables in the `.emacs' file in your home directory. This looks scary to many people because of all the parenthesis. However, you can just cut and paste the examples given here and work from there.

Lets first use a boolean variable. These are variables which you turn on or off, much like a checkbox. A value of `t' means on, a value of `nil' means off. Copy the following line into your `.emacs' file,

exit and restart Emacs.

```
(setq idlwave-reserved-word-upcase t)
```

When this option is turned on, each reserved word you type into an IDL source buffer will be converted to upper case when you press <SPC> or <RET> right after the word. Try it out! `if' changes to `IF', `begin' to `BEGIN'. If you don't like this behavior, remove the option again from your `.emacs' file.

Now I bet you have your own indentation preferences for IDL code. For example, I like to indent the main block of an IDL program a bit, different from the conventions used by RSI. Also, I'd like to use only 3 spaces as indentation between `BEGIN' and `END'. Try the following lines in `.emacs'

```
(setq idlwave-main-block-indent 2)
(setq idlwave-block-indent 3)
(setq idlwave-end-offset -3)
```

Restart Emacs, take the program we developed in the first part of this tutorial and re-indent it with `C-c h' and `M-C-\'. You probably want to keep these lines in `.emacs', with values adjusted to your likings. If you want to get more information about any of these variables, type, e.g., `C-h v idlwave-main-block-indent <RET>'. To find which variables can be customized, look for items marked `User Option:' in the manual.

If you cannot wrap your head around this Lisp stuff, there is another, more user-friendly way to customize all the IDLWAVE variables. You can access it through the IDLWAVE menu in one of the `.pro' buffers, option `Customize->Browse IDLWAVE Group'. Here you'll be presented with all the various variables grouped into categories. You can navigate the hierarchy (e.g. Idlwave Code Formatting->Idlwave Main Block Indent), read about the variables, change them, and `Save for Future Sessions'. Few of these variables need customization, but you can exercise considerable control over IDLWAVE's functionality with them.

Many people I talk to find the key bindings used for the debugging commands too long and complicated. Do I always have to type `C-c C-d C-c' to get a single simple command? Due to Emacs rules and conventions I cannot make better bindings by default, but you can. First, there is a way to assign all debugging commands in a single sweep to other combinations. The only problem is that we have to use something which Emacs does not need for other important commands. A good option is to execute debugging commands by holding down <CONTROL> and <SHIFT> while pressing a single character: `C-S-b' for setting a

breakpoint, `C-S-c' for compiling the current source file, `C-S-a' for deleting all breakpoints. You can have this with

```
(setq idlwave-shell-debug-modifiers '(shift control))
```

If you have a special keyboard with for example a <HYPER> key, you could use

```
(setq idlwave-shell-debug-modifiers '(hyper))
```

instead to get compilation on `H-c'.

You can also assign specific commands to function keys. This you must do in the _mode-hook_, a special function which is run when a new buffer gets set up. Keybindings can only be done when the buffer exists. The possibilities for key customization are endless. Here we set function keys f5-f8 to common debugging commands.

```
;; First for the source buffer
(add-hook 'idlwave-mode-hook
  (lambda ()
   (local-set-key [f5] 'idlwave-shell-break-here)
   (local-set-key [f6] 'idlwave-shell-clear-current-bp)
   (local-set-key [f7] 'idlwave-shell-cont)
   (local-set-key [f8] 'idlwave-shell-clear-all-bp)))
;; Then for the shell buffer
(add-hook 'idlwave-shell-mode-hook
  (lambda ()
   (local-set-key [f5] 'idlwave-shell-break-here)
   (local-set-key [f6] 'idlwave-shell-clear-current-bp)
   (local-set-key [f7] 'idlwave-shell-cont)
   (local-set-key [f8] 'idlwave-shell-clear-all-bp)))
```

Lession III: Library Catalog

We have already used the routine info display in the first part of this tutorial. This was the key `C-c ?' which displays information about the IDL routine near the cursor position. Wouldn't it be nice to have the same available for your own library routines and for the huge amount of code in major extension libraries like JHUPL or the IDL-Astro library? To do this, you must give IDLWAVE a chance to study these routines first. We call this _Building the library catalog_.

From the IDLWAVE entry in the menu bar, select `Routine Info/Select Catalog Directories'. If necessary, start the shell first with `C-c C-s' (*note Starting the Shell::). IDLWAVE will find out about the IDL `!PATH' variable and offer a list of directories on the path. Simply

select them all (or whichever you want) and click on the `Scan&Save' button. Then go for a cup of coffee while IDLWAVE collects information for each and every IDL routine on your search path. All this information is written to the file `.idlcat' in your home directory and will from now one be automatically loaded whenever you use IDLWAVE. Try to use routine info (`C-c ?') or completion (`M-<TAB>') while on any routine or partial routine name you know to be located in the library. E.g., if you have scanned the IDL-Astro library:

```
a=readf<M-<TAB>>
```

expands to `readfits('. Then try

```
a=readfits(<C-c ?>
```

and you get:

```
Usage:    Result = READFITS(filename, header, heap)
...
```

I hope you made it until here. Now you are set to work with IDLWAVE. On the way you will want to change other things, and to learn more about the possibilities not discussed in this short tutorial. Read the manual, look at the documentation strings of interesting variables (with `C-h v idlwave<-variable-name> <RET>') and ask the remaining questions on `comp.lang.idl-pvwave'.
