# The default keyboard etc configuration file for Pythonwin.

#

# The format of this file is very similar to a Windows INI file.

# Sections are identified with [Section] lines, but comments

# use the standatd Python # character. Depending on the section,

# lines may not be in the standard "key=value" format.

# NOTE: You should not need to modify this file.

# Simply create a new .CFG file, and add an entry:

# [General]

# BasedOn = Default

#

# and add your customisations. Then select your new configuration

# from the Pythonwin View/Options/Editor dialog.

# This way you get to add your own customisations,

# but still take advantage of changes to the default

# configuration in new releases.

# See IDLE.cfg for an example extension configuration.

#

##########################################################################

[IDLE Extensions]

# The list of IDLE extensions to load. The extensions

# AutoIndent, AutoFormat and possibly others are

# "built-in", so do not need specifying.

FormatParagraph

CallTips

[Keys]

# The list of \_default\_ key definitions.

# See [Keys:Interactive] and [Keys:Editor] below for further defs.

#Events of the format <<event-name>>

# are events defined in IDLE extensions.

Alt+Q = <<format-paragraph>>

Ctrl+W = ViewWhitespace

Ctrl+Shift+8 = ViewWhitespace # The MSVC default key def.

Ctrl+Shift+F = ViewFixedFont

# Auto-complete, call-tips, etc.

Alt+/ = <<expand-word>>

Ctrl+Space = <<expand-word>>

( = <<paren-open>>

) = <<paren-close>>

Up = <<check-calltip-cancel>>

Down = <<check-calltip-cancel>>

Left = <<check-calltip-cancel>>

Right = <<check-calltip-cancel>>

. = KeyDot

# Debugger - These are the MSVC default keys, for want of a better choice.

F9 = DbgBreakpointToggle

F5 = DbgGo

Shift+F5 = DbgClose

F11 = DbgStep

F10 = DbgStepOver

Shift+F11 = DbgStepOut

Ctrl+F3 = AutoFindNext

[Keys:Editor]

# Key bindings specific to the editor

F2 = GotoNextBookmark

Ctrl+F2 = ToggleBookmark

Ctrl+G = GotoLine

Alt+I = ShowInteractiveWindow

Alt-B = AddBanner # A sample Event defined in this file.

# Block operations

Alt+3 = <<comment-region>>

Shift+Alt+3 = <<uncomment-region>>

Alt+4 = <<uncomment-region>> # IDLE default.

Alt+5 = <<tabify-region>>

Alt+6 = <<untabify-region>>

# Tabs and other indent features

Back = <<smart-backspace>>

Ctrl+T = <<toggle-tabs>>

Alt+U = <<change-indentwidth>>

Enter = EnterKey

Tab = TabKey

Shift-Tab = <<dedent-region>>

# Folding

Add = FoldExpand

Alt+Add = FoldExpandAll

Shift+Add = FoldExpandSecondLevel

Subtract = FoldCollapse

Alt+Subtract = FoldCollapseAll

Shift+Subtract = FoldCollapseSecondLevel

Multiply = FoldTopLevel

[Keys:Interactive]

# Key bindings specific to the interactive window.

# History for the interactive window

Ctrl+Up = <<history-previous>>

Ctrl+Down = <<history-next>>

Enter = ProcessEnter

Ctrl+Enter = ProcessEnter

Shift+Enter = ProcessEnter

Esc = ProcessEsc

Alt+I = WindowBack # Toggle back to previous window.

Home = InteractiveHome # A sample Event defined in this file.

Shift+Home = InteractiveHomeExtend # A sample Event defined in this file.

# When docked, the Ctrl+Tab and Shift+Ctrl+Tab keys don't work as expected.

Ctrl+Tab = MDINext

Ctrl+Shift+Tab = MDIPrev

[Extensions]

# Python event handlers specific to this config file.

# All functions not starting with an "\_" are assumed

# to be events, and take 2 params:

# \* editor\_window is the same object passed to IDLE

# extensions. editor\_window.text is a text widget

# that conforms to the Tk text widget interface.

# \* event is the event being fired. Will always be None

# in the current implementation.

# Simply by defining these functions, they are available as

# events.

# Note that we bind keystrokes to these events in the various

# [Keys] sections.

# Add a simple file/class/function simple banner

def AddBanner(editor\_window, event):

text = editor\_window.text

big\_line = "#" \* 70

banner = "%s\n## \n## \n## \n%s\n" % (big\_line, big\_line)

# Insert at the start of the current line.

pos = text.index("insert linestart")

text.undo\_block\_start() # Allow action to be undone as a single unit.

text.insert(pos, banner)

text.undo\_block\_stop()

# Now set the insert point to the middle of the banner.

line, col = [int(s) for s in pos.split(".")]

text.mark\_set("insert", "%d.1 lineend" % (line+2, ) )

# Here is a sample event bound to the "Home" key in the

# interactive window

def InteractiveHome(editor\_window, event):

return \_DoInteractiveHome(editor\_window.text, 0)

def InteractiveHomeExtend(editor\_window, event):

return \_DoInteractiveHome(editor\_window.text, 1)

def \_DoInteractiveHome(text, extend):

import sys

# If Scintilla has an autocomplete window open, then let Scintilla handle it.

if text.edit.SCIAutoCActive():

return 1

of\_interest = "insert linestart + %d c" % len(sys.ps1)

if not text.compare("insert", "==", of\_interest) and \

text.get("insert linestart", of\_interest) in [sys.ps1, sys.ps2]: # Not sys.ps? line

end = of\_interest

else:

end = "insert linestart"

if extend: start = "insert"

else: start = end

text.tag\_add("sel", start, end)

# From Niki Spahie

def AutoFindNext(editor\_window, event):

"find selected text or word under cursor"

from pywin.scintilla import find

from pywin.scintilla import scintillacon

try:

sci = editor\_window.edit

word = sci.GetSelText()

if word:

find.lastSearch.findText = word

find.lastSearch.sel = sci.GetSel()

else:

pos = sci.SendScintilla( scintillacon.SCI\_GETCURRENTPOS )

start = sci.SendScintilla( scintillacon.SCI\_WORDSTARTPOSITION, pos, 1 )

end = sci.SendScintilla( scintillacon.SCI\_WORDENDPOSITION, pos, 1 )

word = sci.GetTextRange( start, end )

if word:

find.lastSearch.findText = word

find.lastSearch.sel = (start,end)

except Exception:

import traceback

traceback.print\_exc()

find.FindNext()

# A couple of generic events.

def Beep(editor\_window, event):

editor\_window.text.beep()

def DoNothing(editor\_window, event):

pass

def ContinueEvent(editor\_window, event):

# Almost an "unbind" - allows Pythonwin/MFC to handle the keystroke

return 1