Help for PowerPro

Jan 2007

Contents

Introduction

Introduction for new users

getting started with PowerPro

It appears this is the first time you have used Powerpro. Thanks for trying the program.

Here are some places to get started in Help:

about the initial configuration

The initial Powerpro button bar should be shown in the upper left of your screen. See the topic Test driving the initial button bar to test drive the initial configuration and read about how to change it to suit your tastes. Powerpro has many features and taking the test drive will help explain some of them to you.

To exit Powerpro, when using the initial configuration, use the *Exit PowerPro* entry under *Menu* -- or Ctrl-right click on a bar and select the *Exit* menu entry. If you are now using your own configuration, you can add this command as an Item on any Bar or Menu: *Shutdown PowerPro

what can Powerpro do?

See List of features for a detailed list of features.

This is a general summary of PowerPro's features:

- Powerpro provides many user actions for **launchin**g programs and commands, such as bar buttons, menu items, hotkeys, mouse actions, scheduled commands, or from a script.
- Whichever user action you choose to launch an application, you can also choose **how** it will be launched. For example PowerPro has a "Switch to if active" checkbox (which prevents accidentally starting a program twice); you can start an application maximised, minimised, hidden, etc; specify whether it should have OnTop status; define its exact window size and position. You can even automate sending commands to an application every time it starts, such as switching Explorer to Details view.
  There are other ways to start programs (such as ordinary toolbar programs, or Desktop shortcuts, or the Start Menu) but PowerPro can start programs with more customisable options.
- PowerPro offers several ways to **control a running program**, such as sending simulated keystrokes to a specified window, or simulating mouse clicks and movements anywhere on the screen.
- Powerpro provides many ways to **switch between tasks**. You can make an "active bar" as an improved substitute for the Windows Taskbar; or if you prefer, create your own Task Switching Menu, shown by a hotkey perhaps.
- Powerpro offers many ways to configure and control the **Windows system** and Windows features such as Explorer, the Clipboard, the Desktop, etc.
- Powerpro **bars** can be positioned, colored and sized in many ways. You can even apply skins to bars.
- The **menus** you create with Powerpro are also very customisable in appearance with submenus and conditional sections.
- The variety of results you can set for clicking a bar button or a menu item is limited only by your imagination of what you would like it to do.
- Powerpro can retrieve a wide range of **information** about the system and running programs, then display that information however you wish. Or use the retrieved information as parameters in your commands.
- It can be set to **monitor** the computer for various conditions then respond with a warning message or an action, such as automatically starting your email program and a timer whenever the modem connects.

- Powerpro can **replace** many small utilities such as a tray utility, a virtual desktops program, a sticky notes program and more. By using PowerPro's tray features, vdesks, notes, etc, instead of dozens of small programs, you can integrate all of these into a single user interface of your own design. Also the parts can be made to interact with each other in more interesting ways.
- Not only is there unlimited variety to what different users create with PowerPro. There is also variety in how they create it. One user will develop a comprehensive setup without writing a single script, only using the GUI dialogs. Another will prefer to use their text editor to create file based scripts and .ini files to define their configuration.

other Help topics for starters

You can choose a starting point in the Help Contents dialog, then use the << and >> buttons to go through all the Help pages in sequence.

The About PowerPro Help topic explains what the red and green text coloring of the syntax definitions and examples signifies, and the substitutions you will need to make.

The Types of help for PowerPro topic lists the documentation and other useful resources for PowerPro users, some installed with the program, some available on the Internet. It also explains how you can download plugins and addons, and how to print a manual.

In the main help file, PowerPro.hlp, the What's new topic lists what's new for each version.

How one user has configured PowerPro. There are more examples of users' configurations in the Yahoo Group's online files storehelp_types.

Suggestions for configuring PowerPro has ideas about how to configure Powerpro.

For more advanced use, the Combining expressions, variables, and PowerPro commands topic describes the alternative syntax formats for running commands, either using literal text for the command's arguments, or using expressions for the arguments.

tip

PowerPro has an exceptionally large set of features. Nobody can learn to use all of it in one day. Therefore it is best to start by using just one or two features which you need most, then gradually develop a more complicated PowerPro configuration.

I hope you will find PowerPro useful in many practical ways, and enjoy discovering what you can do with it.

About PowerPro Help

Types of help for PowerPro

there are several sources of help for PowerPro users

installed help

The installed Help for PowerPro comes in three formats, which are all in your main PowerPro folder. They all contain the same information, so use whichever format you prefer.

PowerPro.hlp	This is in the traditional WinHelp format used since Windows 95. Most of the configuration dialogs have a "Help" button which jumps to the appropriate page in this file. There are a variety of ways to open this Help at the required topic, described in Showing a particular Help topic.
PowerPro.chm	This is in the new html format for help files. It lacks much of the formatting and the various opening methods of the .hlp format.
PowerPro.doc	This MS Word document, provided in case you wish to make a printed manual. Although it is possible to print topics from .hlp or .chm, it is easier to make a neat hard copy from a Word document, setting the paper size, margins, etc, to suit your printer. For example you may need to change the paper size between A4 and US Letter paper size. Also, different printers have a different minimum for the bottom margin. If you change the paper size and/or margins, you will need to update the page numbers shown in the Contents page. Just place the cursor anywhere in the Contents (the text background will go gray). Then press F9. If you do not have MS Word, you could use WordPad (which is provided with the Windows system) to adjust the paper size and margins and then print the file. Or you can find a free Word Viewer at http://www.microsoft.com/word/internet/viewer/default.htm although the MS Word Viewer will not allow editing of the file. Clickable links in the on-screen version of PowerPro Help have been either removed or converted to topic titles like this Topic Title for this document.
PPST.chm	The PowerPro Scripting Tutorial. Note: The complete reference to using Powerpro scripts is in the Help file, including the most recent changes to syntax. The PPST offers a friendly tutorial about writing scripts.

help on the internet

program downloads

The main website for PowerPro is at http://www.ppro.org (or www.powerpro.webeddie.com)

Here you can download new major versions of the program. New beta versions of Powerpro are announced in the main forum.

main forum

The main support forum for PowerPro is at http://groups.yahoo.com/group/power-pro/

There you can ask questions (preferably after checking this Help first) and discuss ideas about using PowerPro. The Yahoo Group also has an extensive Files section containing many useful resources contributed by users, such as extra plugins; PowerPro scripts; screen shots and descriptions of interfaces which have been created using PowerPro; and additional documentation.

This main message list is fairly busy, averaging about 15 messages per day. You can read the messages online using a web browser. Or if you prefer, you can receive them by email. By adjusting your group membership options, you can receive all the messages individually as soon as they are posted, or combined into a digest in one email per day, or receive only the "formal announcements" which are very infrequent.

Particularly valuable downloads available from the Yahoo group's website are the additional plugins, written by some PowerPro users who are capable programmers. The plugins add many additional commands and functions, to extend the power of PowerPro. They are also available, collected into a single zip, from the powerpro.pcrei.com website (see below).

This main forum is also the right place to make bug reports.

beginners forum

http://groups.yahoo.com/group/powerpro-beginners/ This is less busy than the main forum. Some of the experienced users who participate in the main forum also participate here, answering questions from new users. Actually, beginners are very welcome to participate in the main forum (where your message will be read by more people) but if you feel your questions could be rather basic, you might feel more comfortable asking in the beginners forum initially.

animated help and file store

The online server at http://powerpro.pcrei.com holds copies of several resources for PowerPro, which you can download. There are two main reasons for this separate store: the Files section of the Yahoo group has nearly filled the allotted space; and some people have occasionally had difficulties downloading files from the Yahoo site. The powerpro.pcrei.com website is usually faster than the Yahoo site for large files.

This store includes the complete archives of the forum's messages, sorted into threads and saved in KeyNote format (needs the free Keynote program); and a zip containing all of the additional plugins which are not included in the main PowerPro distribution.

About PowerPro Help

Formatting used in PowerPro Help

Some of the formatting styles used in this Help file have a special meaning

fonts

PowerPro Help uses a different font to show examples of code which you can use in the configuration dialogs, in scripts, and in text config files.

Red TimesNR	Text you enter exactly as shown For example the command: *Exec calendar
Green TimesNR	Text you should change when entering it into dialogs and scripts. For example, where Help says: file.delete(filepath) you would actually type: file.delete("c:\docs\testfile.txt") or maybe an expression representing the filepath.

Help's substitute words

These words are only used in Help, not in real usage, so they are shown in green. You substitute them as follows:

filepath	change this to a literal full filepath, such as: "c:\docs\mynotes.txt" (or use a variable or expression or plugin call which will evaluate to a full file path).
filename	a filename without its path or extension, such as: mynotes

filename.ext	a filename and .extension without its path, such as: mynotes.txt
cmdlist	the name of a Command List such as mybar1 or TasksMenu
captionlist	one or more window specifiers, separated by commas, as described in Caption lists. With some commands, all matching windows will be affected by a command using this parameter. With some commands, only the first matching window is affected. "Caption lists" are used in many places, such as specifying the target window to send simulated keystrokes to, the targets of Window commands (close, min, hide, ontop, etc), and the context for conditional items.
expression	means you can use either a built-in Function, or a variable, or a plugin call, or some "literal text in quotes", or a whole number as digits, or a complex string such as: "this text" ++ var1 ++ "more text". Details in Expressions
Function	use one of Powerpro's built in Functions (sometimes called a "keyword" or "a built-in Expression"). These built in Functions allow you to get a wide range of information which you can display or use in the parameters of commands. Some are just a single word, such as shortdate or dialupname. Other functions need one or more parameters in parentheses, such as max(val1,val2) or timerrunning("7"). Details in Functions by category
text	any text in places where literal text is not placed inside double quotes (for example the text of your message in the command *Message text)
"text"	any text in places where literal text must be in quotes, such as inside expressions.
var	You can use any user-defined variable here, instead of "var". Help also uses other example variable names, such as var1, var2, Title, xPos, yPos. You can substitute your own variable names in those places.

About PowerPro Help

Showing a particular Help topic

You can show PowerPro Help in various ways:

- Ctrl-Right click any bar and choose *Help*
- In most of the configuration dialogs, there is a *Help* button which takes you to the appropraite page for that dialog.
- When entering a command in the command editing dialogs, after you select a built in command from the drop down list for the Command box, clicking the nearby *Help* button will take you to the Help topic for that command.
- You can use a command for winhlp32.exe, in the Windows or Winnt folder. Use a parameter from the list below, which shows the Topic IDs for each page in this PowerPro Help.

using WinHelp's command line

```
c:/windows/winhlp32.exe -I topic-id filepath
```

You may need to adjust the path to winhlp32.exe

topic-id should be replaced by one of the Topic IDs shown below.
They are listed in the same order as in Help's Contents dialog.

filepath should be replaced by the full path to powerpro.hlp

Paths should be in quotes if they contain a space

example

`c:/windows/winhlp32.exe -I functionscat C:/Shells/PowerPro/powerpro.hlp`

opens PowerPro Help at the topic which lists all built in functions by category.

Note: See the far end of the list for the other topics about Functions.

list of topic ids

Introduction for new users = firsttime

About PowerPro Help

Types of help for PowerPro = help_types
Formatting used in PowerPro Help = help_formats
Showing a particular Help topic = helpcommand

Installing PowerPro

Installation, upgrading and removal = install
Installed files = installed_files
PowerPro license and lack of warranty = register
What's new = new

Overviews and features

Overview of PowerPro = overview
Standalone utilities which powerpro replaces = newuser_utility
List of features = combine
Suggestions for configuring PowerPro = guidelines
How one user has configured PowerPro = testimonial

Tutorials using the initial configuration

Test driving the initial button bar = test_drive
Changing a label and an icon on the initial bar = tutorial_button
Changing the look of the initial bar = tutorial_look
Changing the size and position of the initial bar = tutorial_position
Changing the tray icon from the initial bar = tutorial_trayicon
Adding a new button to the initial bar = tutorial_add

Demonstrations

Demonstrations and samples = demo_all
Demonstration of menus and context menus = demo_menu
Demonstration of subbars and manually shown bars = demo_subbar
Demonstration of *Window commands = demo_window
Demonstration of sending keys with *Keys = demo_keys
Demonstration of hot keys and mouse actions = demo_hot
Demonstration of *Menu Folder = demo_folder
Demonstration of keyboard macros = demo_macro
Running commands when a window first opens = demo_start
Demonstration of *Format Item and *Info = demo_resources
Bars Showing Other Bars when a Button is Clicked = sliding
Examples of bar positions = demo_bars

Miscellaneous usage topics

Versions of Windows = win95
PowerPro command line = commandline
Running built-in commands from the command line = commandbuilt
Using the middle mouse button = middlemouse

Minimizing a window to the tray = traymin
Changing Explorer list and view settings = explorerforce
Using the keyboard to access a button bar = keybar
Scrolling with mouse movements = scrolldetail
Manual scrolling with the mouse = manualscroll
Automatic scrolling with the mouse = autoscroller
Mouse hover over button = autopress
Adding items to the explorer right click menu = explorercontext
Favorite folders lists for file open/save dialogs = favfolders
Sending folder names to open/save dialogs = favfolders2
Buttons from files in folders = folderbuttons
Sharing PowerPro configurations = sharing
Plugin usage = plugin
Plugin programming = plugin_program
Auto completion = autocompletion
Displaying different subsets of a bar = subbar
Drag and drop onto a bar = dragdropcommand
Positioning the bar = sbs_position
Screen edge positions = screenpos
Positioning a bar in the active window = pos_active

General syntax guidelines

Combining expressions, variables, and PowerPro commands = xout
Expression follows character = exprfollows
Special characters used in PowerPro = special_chars
Caption lists = caption_list
Window handles = windowhandle
Clipboard contents as a command parameter = clipboard_parameter

Configuring with the GUI dialogs

About the dialogs

Configuring PowerPro using dialogs = configure_help
Starting pproconf = start_pproconf
Command entry controls = command_entry_controls

Setup

"Setup" dialog = specialwin
"Advanced setup" dialog = advanced
Automatically moving the mouse cursor to a dialog button = autobutton
Automatically hiding windows = autohide

GUI Control

"GUI Control" dialog = special
Clip filters = clipfilter
Clipboard history tracking = cliphistory
Caps Lock, Scroll Lock, Num Lock = key

Command Lists

"Command Lists" dialog = menu_contents
"Tool tip setup" dialog = tooltip_setup
Bar and Menu "Setup" dialog = menu_setup
Tray icon buttons = trayicon
Command list "Properties" dialog = all
Working with hidden bars = barhidden

Configuring a command list item = button
Running multiple commands = mul_command
Active window switching with buttons = activebar
Omitting windows or words from active window lists = omitlist
Automatically running commands when windows open = autorun

Hotkey/Mouse

"Key/Mouse" dialog = hotkey
Window-specific hot keys = contextkey
Hot key/mouse action explanations = hotkeycombo
Hot key and mouse "Setup" dialog = key_setup
Entering hot key/mouse action information = hotkey_item

Scheduler

Scheduler "Setup" dialog = alarm_setup
"Scheduler" dialog = alarm_details
Alarm log = alarm_log
Suspending alarms = suspendalarm
Running programs after system idle = idle
Entering information for a scheduled command = alarm_item

Timers

"Edit timer" dialog = resetimer
Timer logs = timer_log

Media

"Media" dialog = sound
PowerPro Sounds = sound_list

Configuring with text files

Maintaining configuration using .ini files = import
Layout of text configuration files = importfiles

Configuring with CL functions

Overview of CL functions = cl_overview
CL functions for working with the entire command list = cl_list
CL functions for getting item attributes = cl_getitem
CL functions for setting item attributes = cl_setitem

Built in Commands

PowerPro built in commands = builtin

*Bar

Bars and the *Bar command = bar_help
*Bar format command = bar_format

*Clip

*Clip command = autoclip

*Desktop

*Desktop command = desktop_help
Changing screen display resolution = changedisplay
Saving and restoring desktop icon positions = iconpos

*Exec

*Exec commands = exec_help
Showing folders in Explorer = showinexplorer

Hiding windows with *Exec = hidewindow
Prompting for Yes/No = exec_prompt
Logging keystrokes = keylog
Tiling windows = tile
See mouse cursor position and window information = movesize
Tiny Type and Run box = tinyrun
Date Calculator = calccalendar
Working with CDs = CD_help
Sound volume and mute = soundvolume

*File

*File command = filing
Writing entries to a file = logfile
Working with a randomly selected file = randomfile

*Format

Formatting menus and bars with *Format = format_help
Changing the look of an item with *Format Item = format_item
Window-specific menu and bar contents = contextmenu

*Keys

*Keys command = sendkeys
*Keys target window = sendkeywindow
*Keys codes for keystrokes = sendkeykey
Examples of *Keys commands = sendkeyexample
Sending *Keys to a new window = sendkeystart
Selecting *Keys from a menu = filemenu

*Macro

*Macro command = macro

*Menu

*Menu command = menu_help
Showing menus = menu_show
Working with Explorer windows = explorerwindows
File menus = file_menu
Properties in file menus = properties
*Menu Folder = filedoc
*Menu Folder formatting = folderformat
Entering format information for *Menu Folder command = foldercontentswork
Special folders for *Menu Folder = specialfolders
*Menu Folder with a large folder tree = largefolder

*Message

*Message command = message_help

*Mouse

*Mouse command = mouseseq

*Note

*Note command = note_help
Auto cloning Notes = autoclone

*ScreenSaver

*ScreenSaver command = screensaver_help

*Shutdown

*Shutdown command = shutdown_help

*Skin

Creating skins = skinauthor
Using skins = skins
Sections/subbars for skins = sectionsubbar

*Timer

*Timer command = timer

*TrayIcon

Working with tray icons from other programs = trayiconother
Tray icon support = powerprotray
Installing tray support in Windows 2000/XP = powerprotray2000
Training PowerPro to recognize tray icons = trayiconothertrain

*Vdesk

*Vdesk command = vdesk
Virtual desktop setup = vdesksetup
The Virtual Desktop Menu = vdeskmenu
Initializing virtual desktops = vdesktab
Virtual Desktop Demonstration = demo_desk

*Wait

*Wait command = waitcommand
Wait example = samplescript

*Wallpaper

*Wallpaper command = wallpaper_help

*Window

*Window command = controlwin
Actions for the *Window command = controlwinaction
Specifying the windowID for the*Window command = controlwinwindow
Auto positioning windows = positionwindow

Scripts

PowerPro Scripts = virtual
*Script commands = scriptfile
Running a script when a window or system event occurs = hookwindows
Programming scripts with if, jump, variables, flags = virtualprogramming

*Info labels

Using *Info labels = buttonspecial
Date and Time *Info labels = dtformat
System resource *Info labels = usage
Other *Info labels = buttonother

Expressions and functions

Expressions = expression
Functions by category = functionscat
Functions alphabetic list = functionsalpha
formatdate() and formattime() = format_datetime
Input dialogs = inputs
MessageBox function = messagebox

String functions and operators = stringfunc
MCI function = mci_op

Installing PowerPro

Installation, upgrading and removal

first time installation

Double click on **setup.exe** after unzipping ppro*nn*.zip to a temporary folder.

You can choose where to place your main PowerPro folder. Setup will not create any files or folders anywhere else on your hard disk.

For Windows Vista, PowerPro must be installed with administrative privileges and run in XP-compatibility mode.

upgrading to a newer version

Quit PowerPro first. You can install a newer version on top of an older version, and your configuration will be preserved. There is no need to uninstall the older version first. You may wish to make a backup copy of your whole \powerpro\ folder first, just in case you wish to revert if you have problems with the newer version.

Installation zips of Powerpro do not include a "pproconf.pcf" file, so they will not overwrite your existing configuration.

If you see this message:

"PowerPro cannot install ppro.dllnew because Windows has locked ppro.dll in memory"

Powerpro uses a dll file called ppro.dll which you will find in the same file folder as the PowerPro.exe program. This file may sometimes be locked in memory by the Windows system meaning that the installation program is unable to upgrade the file. If you get this message, you will need to restart your system; this option will be offered to you by the dialog box with the error message. The setup program will remove the existing PowerPro shortcut from your startup group, so that when you restart PowerPro and the ppro.dll will not be loaded. After your system has restarted, set setup.exe again.

removal

Automatic
The recommended method is to use Control Panel, Add/Remove Programs to uninstall PowerPro, or run unwise.exe in the PowerPro folder.

Manual
To manually remove: close Powerpro, delete any PowerPro desktop icons, delete entries from the Start Menu (using Taskbar Properties, Start Menu programs; or Explorer) and finally delete the folder containing PowerPro files.

PowerPro's setup adds very little to the Registry, because program information is stored in the data files in the PowerPro folder. In the Registry, it only makes:

- a file association for *.powerpro files, so you can double click a script file to run it;
- an association for *.ppronote files, so you can double click a Note to view it;
- and a pointer to the main PowerPro folder, stored in HKEY_LOCAL_MACHINE\SOFTWARE\StilettoProducts\Stiletto , "Folder" which is only needed by the install program when doing an upgrade.

If you are confident with using Regedit, you can remove those entries; however it can do no harm to simply leave them there.

Installing PowerPro

Installed files

The following files are installed into your PowerPro folder.
Setup.exe does not install files anywhere else on your hard disk,
except for the shortcuts on the Start Menu and Desktop.

you can run the following executable files

PowerPro.exe	PowerPro's main program
PproConf.exe	PowerPro's configuration program
unwise.exe	To uninstall PowerPro
dundial.exe	Command line Dial-up networking (RAS)
dundial.txt	Explanation of dundial.exe
dunhang.exe	Hang up Dial-up connections
dunhang.txt	Explanation of dunhang.exe

do not run the following executable files

they are only for use by powerpro.exe

stilres.exe	Win16 program used to retrieve GDI/User resources
wrestart.exe	Win16 program used for fast restart and screen saver power
PProtray.exe	Enables certain system tray functions
PProtra2.exe	Enables certain system tray functions
pprotray.exenew	Temporary copy created during installation
pproicl.dll	Read large icons from icl files dll
ppro.dll	PowerPro hook dll
ppro.dllnew	Temporary copy of ppro.dll copied to ppro.dll during successful install
PProctx.Dll	Enables extensions to the Explorer context menu
pproctx.dllnew	Temporary copy created during installation
PProtray.Dll	Enables certain system tray functions
pprotray.dllnew	Temporary copy created during installation
psapi.dll	Used in Windows NT to look up exe file names
stil32jp.dll	jpg to bmp conversion dll

configuration files

pproconf.pcf	The configuration file used by default; if it exists it is loaded when PowerPro starts. If powerpro.pcf does not exist, then default.pcf (the config for beginners) will be loaded instead. Installation zips of PowerPro do not include a pproconf.pcf file. Therefore the installer (setup.exe) will not overwrite your existing configuration when you are upgrading PowerPro into your existing powerpro folder.
Default.pcf	The default configuration for first-time users. It is loaded when Powerpro starts only if there is not a file called pproconf.pcf in your powerpro folder.
BarShowsBars.pcf	A demonstration config
Demo.pcf	A demonstration config
demodesk.pcf	A demonstration config
subbars.pcf	A demonstration config
empty.pcf	For a fresh start with a new empty configuration

PowerPro.icl	Holds icons displayed for PowerPro windows (in powerpro.exe folder)
Default.icl	Copied to PowerPro.icl if this file does not exist (in powerpro.exe folder)

these backups are created automatically

!Auto backup of pproconf.pcf
!Previous auto backup 2 of pproconf.pcf
(and more numbered backups)

other data files (do not edit them)

!pow_temp.bmp	Stores jpeg wallpaper after conversion to .bmp
!PProInfo!.ini	Stores info about tray icons and the last executed program
explorer.windows	Stores Explorer window tracking info
explorer.windows.txt	Stores Explorer window tracking info in text format

Help

PowerPro.hlp	PowerPro Help file.
PowerPro.cnt	Contents file needed by PowerPro.hlp
PowerPro.doc	PowerPro help in Word 6.0 format
PowerPro.chm	PowerPro help in html-help format
ppst.chm	The PowerPro Scripting Tutorial

other documentation

readme.txt	Text overview and installation info
file_id.diz	Blurb for BBS's
pwrpro46.xml	Blurb in xml format
filelist.txt	This file list, as a text file
pprofunctions.txt	List of all functions that can be attached to hotkey for lookup with keys {filemenu c:\program files\powerpro\pprofunctions.txt}

sample skins

SkinSampleKaos.zip
SkinSampleKaos1.zip
SkinSampleNewbie.zip
SkinSampleZlk.zip
SkinTemplate1.txt
SkinTemplate2.txt

sub folders in the PowerPro folder

clip	used by PowerPro to store your saved clips
notes	used by PowerPro to store your Notes
scripts	store your PowerPro scripts here
skins	store your skins files here
plugins	contains plugins included with the distribution.

Note: You can add more plugin dlls into this folder.

Either download individual plugins from:

http://groups.yahoo.com/group/power-pro/files/

or download the complete collection of plugins in one zip file from:

http://www.pcrei.com/ppro/files/index.html#Powerpro_Plugins

Installing PowerPro

PowerPro license and lack of warranty

only the user is responsible for the results of using this program

credits

The Powerpro program, its DLLs, and its various documentation files, are all Copyright 1998-2007 by Bruce Switzer. All Rights Reserved.

The Help files (.hlp, .chm and .doc) were written by Bruce Switzer, with some editing and formatting by Alan Martin and Sheri Pierce. They are Copyright 1998-2007 by Bruce Switzer.

The original PowerPro icon was created by Jonas Hjortland; later versions created by Alan Martin.

The design for "wait for" is based on ideas by Russel Oberio.

The pproicl.dll code to retrieve large icons from icl files was written by Marko Bozikovic.

The regex plugin was written by Julien Pierrehumbert and designed by Julien and Luciano Esperito Santo.

The PowerPro Scripting Tutorial in PPST.chm was developed by Alexander Cicovic and the PPTF team.

The PPMLA (message list archives) and PPSL (script library), both available from powerpro.pcrei.com , are maintained by Nathan Sendhil.

The code to refresh environment variables was written by Vincent Fatica.

The program's main online support forum, http://groups.yahoo.com/group/power-pro/ is managed by Luciano Esperito Santo.

The online server http://www.pcrei.com/ppro/files/index.html which provides downloads of Powerpro related resources, is maintained by David Troesch.

Thanks are also due to the community of PowerPro users, too many to all be named here, including: the writers of the user-contributed plugins, which add significantly to the power of PowerPro; those who contribute their time, ideas and problem solving skills in the forums; beta testers who make bug reports; and those who have uploaded various resources to the Yahoo group's Files store and the powerpro.pcrei.com web site.

legal disclaimer

The 32 bit version of Powerpro is freeware and may be used as desired.

However users must understand and agree that this software is distributed "as is," without warranty as to performance or merchantability or any other warranties whether expressed or implied. Because of the various hardware and software environments into which this program may be put, no warranty of fitness for a particular purpose is offered. Good data processing procedure dictates that any program be thoroughly tested with non-critical data before relying on it. The user must assume the entire risk of using the program.

If the user does not agree with these usage conditions for PowerPro, the user must delete it from the system.

All trademarks used in this Help File are the property of their respective owners and are used for explanatory purposes only.

The jpeg conversion routines in this software are based in part on the work of the Independent JPEG Group.

Overviews and features

Overview of PowerPro

PowerPro is intended to supplement the Windows shell by providing quick, minimal mouse click or minimal key press access to your most used commands, documents, and activities while taking up little desktop space.

Powerpro enables you to create and customize an interface of your own design, for controlling the Windows system and your applications. This can be as simple or as intricate as you wish. It may consist of one or many hotkeys, menus, tray icons, and button bars which can display a wide range of information and execute a wide range of functions.

how it works

The three main files involved are:

- ***.pcf** The interface you create (including your customised bars and menus) is all saved in your Powerpro configuration file, usually called **pproconf.pcf**. You should not edit this file directly with a text editor. Most users only have one config file, called pproconf.pcf which is stored in the main powerpro folder. You can create several alternative .pcf files and switch between them. Only one of your .pcf files is currently in use by powerpro.exe. You should take a backup copy of pproconf.pcf using explorer when you have created a configuration you like.
- **pproconf.exe** You customize your PowerPro configuration file by using the multi-tabbed dialogs provided by pproconf.exe (or if you prefer, by using text based configuration methods). This program only runs while you are changing your setup and exits when you press either OK or Cancel to dismiss its dialogs.
- **powerpro.exe** Your configuration file is executed by this program, which runs continuously to show your bars and menus and execute your commands. Most users set it to start automatically when Windows starts.

independent configuring and execution

A difference between PowerPro and most other programs: usually, when you are using an application's configuration dialog, the application stops working until you close the config dialog.
Differently, powerpro.exe and pproconf.exe are quite independent of each other. While you have pproconf temporarily open to adjust your settings, powerpro.exe continues to operate normally; you still have access to your PowerPro bars and menus. When you click OK to quit from pproconf, powerpro.exe starts using the new info from your changed configuration file.

Also, this separation of the configuration program enables you to run pproconf before starting powerpro. That can be useful if your configuration file contains a serious error which stops powerpro from being able to run it – in such a case, just exit powerpro, run pproconf to remove the error from your configuration, then restart powerpro with the adjusted .pcf file.

installed .pcf files

To get you started, PowerPro includes several alternative configuration files for beginners:

- the initial configuration which has one button bar with some dropdown menus. Stored in default.pcf, this initial bar appears the first time you run PowerPro.
- and some alternative demonstration .pcf files, which contain different kinds of sample bars, to illustrate some of PowerPro's features.

You can either start with one of these beginner configs as the basis of your own PowerPro interface for Windows, then gradually change it to suit your needs; or you can start with an empty configuration file (just copy empty.pcf to pproconf.pcf) and then build your own interface from scratch.

the building blocks

Powerpro provides many functions and commands which can be used as the building blocks in your configuration. Basically you do this by:

- Decide what result you want, such as renaming a file or opening a program.
- Find which PowerPro command you need to achieve that result.
- Choose a user action to attach this command to, such as a hotkey, or right-clicking a new button on one of your bars, or an automatic execution method such as setting a Scheduler item.
- Start in the right dialog for that kind of user action, under one of these tabs of the main dialog: *Hotkey/Mouse*, or *Command Lists* for items on bars and menus, or *Schedule*r, or *Timers*.

  In every dialog where you can define a user action, such as adding to the list of Hotkeys or adding a new item to a command list, there are the same standard controls for entering the command which this user action should run.

There is a Feature List topic showing the many different user-actions which you can set up a response to; and the many different results which you can attach to any of those user actions.

PowerPro can also retrieve many kinds of system information for you, which you can have displayed however you wish, or use as values in a command or in a script.

advanced usage

You can start with some easier items, where each user action has one resulting command, then progress to using PowerPro to automate more complex jobs for you.

Programming: You can attach more complicated results to a user action with scripts. At their simplest, scripts are a series of commands which are run in sequence. They can include any Windows command and any PowerPro command, plus there are special commands for controlling scripts with programming logic such as If ... Else, jumps, For ... loops, etc. There are also user defined variables, and various ways of showing customised gui dialogs for user input at run time.

A script can be saved as a Command List and edited by using the configuration dialogs; or saved in a text file with any text editor such as Notepad.

The Yahoo Group's Files section contains interesting and useful examples of user's scripts. There is a separate scripting tutorial (PPST.chm in your PowerPro folder) and a scripts library (PPSL from the http://powerpro.pcrei.com web site).

Delayed action: Instead of starting a command or script directly from a user action, you can have it executed automatically whenever you wish by a scheduled alarm or by a timer. These are easily setup via configuration dialogs or, if you prefer, by scripted commands or text based command lists.

Scheduled events (also called alarms) can optionally have regular repetition (such as: every Monday and Thursday at 2 am) and conditional execution (such as: only if this application is / is not running).

Timers are another way of setting a result to happen later, once only or at repeated intervals. Each of the 26 independent timers can have commands defined for when it starts, when it stops and when it is reset.

There are also a variety of **Wait** commands, to pause a script for a set amount of time or until a condition becomes true.

Conditional actions: In addition to the If() statement, which can be used in scripts, many of PowerPro's features allow conditional execution without any need for scripting with if().

For example a hotkey Ctrl+F12 could simply be global -- or it could be set to do one thing if Opera is active, another thing if Netscape is active, and something else when any other program has the focus. That is done by setting target windows in that hotkey's editing dialog.

You could create a conditional toolbar (or an extra section on your main toolbar) which is set to only appear while Photoshop is in use, or only while one of your virtual desktops is current. It could have buttons for automating frequently used Photoshop commands.

Similarly, context sections of your menus can be shown only when a certain program is active, or depending on any other set of conditions.

With a script of just a few lines, it is possible to take conditional execution further. Using If(condition) you can test for a huge variety of conditions to do with the operating system, other running programs, what's on the hard disk, modem activity, user activity, the clipboard, even what is currently the default printer.

Overviews and features

Standalone utilities which PowerPro replaces

with PowerPro you can access many features in one integrated program

Powerpro can replace many stand alone utilities which customize or control some part of Windows. For example:

Tray programs: With PowerPro you can create your own tray icons and also control the icons of other programs. For example, a tray icon to access Desktop items as a menu: make a tray icon with a *Menu Folder Desktop command.

Internet session timers: use timers, timer logs, and modem control features.

Hiding/showing desktop icons; also save and restore their positions for different screen resolutions: use the built-in command *Desktop.

Hiding/showing/minimizing all desktop windows together: use built-in command *Desktop.

Hiding/showing/minimizing specified individual windows: use built-in command *Window.

Control of Start Menu placement: use built-in *Menu StartMenu command, or *Menu Folder with startmenu as parameter. Optionally, only show some subfolders of the Start Menu.

Enable/Disable/Start the screen saver by (for example) placing mouse at screen corner: use the screen corner as a hotkey command and the *screensaver command. You can also change the screen saver to a specified or randomly chosen saver.

Sticky Notes. Try PowerPro's Notes with automatic saving, categories and other interesting features.

Clipboard functions and history

Hotkeys and keyboard macros

Scripting can include any PowerPro or Windows commands, with programming logic.

Quick Windows Shutdown/Restart/ etc, with or without confirmation, or access the built-in Windows shutdown dialog.

Virtual desktops, with a built in Virtual Desktops Menu and a set of Vdesk commands.

Job scheduling: use scheduled alarms or timers

Reminder programs: use alarms or timers

Program mouse buttons such as middle button for double click: set middle mouse as a hotkey to run a *Mouse leftdouble command. You can also program the fourth and fifth buttons found on some mice.

Hiding windows, set a window to be always OnTop, size or move a program's window, rollup a window to only its title bar: use Window commands.

Create interactive features: you can show a variety of dialogs with your own text, a choice of clickable buttons, such as OK, Cancel, and a choice of text entry boxes, such as one for choosing folders or files with a browse feature, or one for a color input with a color choosing dialog. You could show your custom dialog from a script which then branches according to the user's input.

integration

By using PowerPro's tray functions, vdesks, notes, saved clips, etc, instead of using dozens of small programs to provide all those features, you can integrate them into a single user interface of your own design. Also the parts can be made to interact with each other in many interesting ways.

Overviews and features

List of features

the power and flexibility of PowerPro

Powerpro gives you the power to control your system and how you access programs by allowing you to choose the combination of how to activate and what to activate.

The next two lists show **How** (the techniques for activation) and **What** (the things you can activate). You can combine any entry from How with any entry from What.

how to activate

buttons on bars

Create any number of small or large bars. Each bar is defined by its own command list.

Features for the bar as a whole can be set in that command list's Properties dialog where there are many options including:

> Bar Positions can be floating; or a choice of many resolution-independent standard positions; or place a bar in the title bar of the currently active window; or integrate a bar into the Windows Taskbar.
>
> Bars can optionally have OnTop status. They can be hidden and shown by commands or automatically in various ways.
>
> A bar can be set to show only if a given program is active. For example a bar which only appears when Photoshop is active could have commands for automating some image editing processes.
>
> Or you can set other kinds of conditions for a bar to appear automatically, such as whenever the modem is connected.
>
> A bar can have a bitmap background, which can be turned on or off for each individual button.
>
> It can 3D or flat, border or not. have several rows, dividers,

Each item in the command list represents one button, defined in the item editing dialog, which includes:

> Buttons can be set to respond to mouse-over, with no click required, by checking Hover clicks.
>
> Each button on a bar can have its own text color and background, can show any icon and/or text.
>
> Buttons can be uniform size or varying heights and widths.
>
> You can set a tooltip for each button. The look of PowerPro's tooltips can be customised.
>
> Left, middle, right click on a button can have different commands. Each command is defined using PowePro's standard command entry dialog.
>
> Drag and drop files from Explorer onto a button to execute a command on them.
>
> Buttons can display dynamically updated information, such as free space on drive D:, date, time, modem activity, the contents of any user variable, the result of any expression.

A bar can have a set of interchangeable subbars. Which subbar is shown can be controlled manually, or automatically depending on which Virtual Desktop is active. Subbars are demonstrateddemo_subbar here.

A bar can be skinned. Skinning offers even more variety of appearance; for example it can be non rectangular with rounded corners. Skinned bars can be combined with subbar features.

There is keyboard access to bar buttons when you prefer not to use the mouse.

You can set defaults for all your bars, then make some bars vary from those defaults in its own Properties.

items on menus

You can define up to 500 user-configurable menus, optionally with cascading submenus. Each menu is defined by its own command list.

Features for the menu as a whole can be set in that command list's Properties dialog.

Pick an item with mouse or keyboard and dismiss the menu.

Optionally, set different commands for left and right click on menu items.

Usually, each of your menus is based on its command list, but they can be created automatically for you from various sources:

> A "file menu" can be defined by the lines in a file.
>
> A "folder menu" can show all files in a folder. Pick one to run it. A file menu can optionally show subfolders as cascading submenus.
>
> A menu of saved clips. Pick one to paste it.
>
> A menu of current tasks, or of hidden windows, etc, by using "menu" as the parameter of any *Window command. Set any action for a click, such as Close or Show.
>
> A menu of favorite and recently used folders (useful in Explorer and/or in Open/Save dialogs).
>
> Create menus from virtual folders such as Control Panel, Printers, Network Places.
>
> These dynamically created menus can be free standing or embedded in another menu, included in the top level or as a submenu.

Show the Start Menu anywhere on screen; or a menu of your desktop icons.

Menus can be shown in many ways – from a hotkey, bump a screen edge, etc

Menus can have submenus, horizontal separators and vertical dividers (to start a new column) using formatting commands in the list.

Menus can have contextual sections, only shown under your defined conditions.

You can also add your own items to Explorer's context menus.

tray icon

Make your own customisable tray icon. It can show any icon and/or any text, such as the date or time in a wide range of formats. It can have a tooltip. It can have any command for left, middle, right clicks. You could use it to show a menu or a popup bar...

hot keys

Define an action for alt, ctrl, win, shift, plus any key

or tap (or double-tap) ctrl, alt, shift, win, caps lock

You can specify a "hotkey follows" key (such as ` or ;)

Hotkeys can be global or depend on the active program

Use your keyboard's extra multimedia and internet keys for any purpose

PowerPro hotkeys can be suspended and reactivated

keyboard macros

A macro consists of a short string of characters and can run any command, such as pasting a frequently used phrase into your word processor. Macros are especially useful if you run out of programmable hotkeys.

mouse actions

Define a mouse action to act like a hot key

Create a different command for each mouse button when you click on:

anywhere on a window

a window's caption (title bar)

or the left and right halves of a caption can be set to different commands

any window's system, min, or close icon

left, middle, right click on the desktop

press and hold a mouse button

a short horizontal or vertical drag

a horizontal or vertical stroke

move mouse pointer to a screen corner

bump edge of screen

chord two mouse buttons

the fourth and fifth mouse buttons can also be given commands

Hotkey/Mouse actions can be global or can depend on the active program.

Hotkeys can be temporarily suspended

command line execution

Use the "Browse and Run" dialog or the "Tiny Run" dialog, with optional auto completion, to run any PowerPro or Windows command.

automatic execution

As well as happening in response to the user actions listed above, a command may be set to run automatically:

When a specified window first appears, based on its caption or exe name.

At a scheduled time: any time/date with a repeat interval; after the system is idle for a specified time; at PowerPro start up. Scheduled alarms can be temporarily suspended.

Based on a timer: for each of the 26 timers, set a command for when it starts, one for when it stops, and one for when it is reset.

Based on a monitored condition, such as when the modem connects /disconnects, when free space on a drive falls below a certain level, etc, using the "monitor" special command list.

what to activate

You can attach any of the following commands to any of the user actions listed above. You can attach a single command, or attach multiple commands such as: start Notepad, then resize it, then paste some text into it.

run any program, shortcut, or document

specify parameters which can be typed literally or as an expression which will be freshly evaluated each time the command is run specify simulated keystokess to send when the program starts

specify window configuration at start: max/min/normal, hidden, centre screen, OnTop...

specify "switch to if active" to avoid running a program twice together

optionally show a dialog to prompt for a user input before starting the program

control any window of any program

switch tasks with your own more customisable Taskbar substitute, called active buttons, or with a tasks menu, by using "menu" as the parameter of a Window Show command

for any Window command, select target window(s) by caption, or under mouse, or all windows, or one from a menu of active windows, or of hidden windows

select the main window, or an MDI child window, or a dialog box, as the target of your command

window commands can include many actions:

activate, close, hide, show, minimize, minimise to the tray, maximize, normal

set topmost, not topmost, minimize to the tray

send to back (underneath all other windows)

change a window's position and size

rollup so only the caption is visible

Automatically minimize some windows to the tray instead of to Taskbar buttons.

Control any dialog by simulating a click on a control button, or by moving to a field and then pasting text into it.

Move cursor to default button of dialog (and optionally autopress the button) for your defined list of dialogs

Maintain a list of favorite and recently used folders for quick recall in Open/Save dialogs.

Make open/save dialogs larger than standard

send keystrokes to a running program

send keystrokes to insert text

to control the program by sending Alt+ or Ctrl+ keys

or send simulated mouse actions to a running program

work with tray icons of other programs

simulate left, middle, or right click

or hide a program's tray icon See details here.

run a script of many commands

a scriptvirtual can contain any PowerPro commands or Windows commands

plus programming logic with if-else, jump, wait, for-loops, and expressions such as user variables and built in functionsfunctionscat

you can do string manipulations and arithmetic

you can read and write to files with the File plugin; or using the *Exec ToFile command which appends a line (useful for logging); or log keystrokes

you can send parameters to scripts when they are run; scripts can set a Return value or set variables to their results; scripts can be nested, one calling another, with arguments and returns.

keep notes

Use PowerPro's Notes feature, like enhanced yellow sticky notes. You can use PowerPro's Notes Menu (maybe attached to a hotkey or bar button) or make your own bar or menu for controlling your Notes using the *Notes commands.

use virtual desktops

You can set a bar button or hotkey, etc, to show PowerPro's Virtual Desktop Menu which has a list of your defined desktops to switch to, and New Desktop, Arrange, Unlock, Lock, Remove From Desktop, Move/Copy from this, Clear this Desktop, Clear All Desktops and Close, Move all Windows to Current Desktop, Clear and relaunch from list, Close and move windows to..., Rename Desktop, See All/Move/Copy to..., See All/Switch To..., Start Desktop From List,

Or create your own desktops control bar using the many *Vdesk commands.

Unlike using a separate virtual desktop program, you can integrate your PowerPro vdesk usage into your other PowerPro controls. For example your commands can use these built in Functions in their parameters: deskname, desknum, vdeskhaswindow(vdesk, windowID), deskempty. A bar or a subbar can be set to only be shown for certain desktops.

control the look of your desktop

Start, enable, disable, change the screen saver with commandsscreensaver_help or set the paper in the Media dialog

Disable screen saver whenever Dial-Up Networking connection is active

Change any system sound or PowerPro sounds

Change the desktop wallpaper with a command or in the Media dialog

All media changing commands can be to a random, or sequential, or specified file

Hide/show desktop icons or the Task Bar

Save and restore desktop icon positions

control the windows system

Shut down or restart Windows, sleep, hibernate...

Control the behaviour of Caps Lock/Shift and Scroll Lock

Track the clipboard automatically and store text clips for reuse

Many ways to use the clipboard contents

Change the display resolution and color depth

Play music CDs, control the volume, or mute; use MCI sound commands.

Clear or shorten the Recent documents list

Use and set environment variables

Track the folders viewed in Explorer, to display on a menu for easy recall

Force any Explorer window to List, Details, Small icon, or Large icon view

Add your own items to Explorer's context menus

gather information

PowerPro's built in Functionsenable you to get information which you can use as parameters in your commands, or show them in a dynamically updated display of your own design.

Functions can tell you about: the Windows system, system resources, running applications, visible and hidden windows, files and folders, modem activity, date and time calculations, current CD track, volume, muted, mouse coordinates, the current default printer, the results of user inputs...

Your command or script can also get information from the user at runtime, with a variety of customisable dialogs prompting for user inputs (such as inputfolder which has a tree browsing feature, or inputcolor which has a color choosing dialog) followed by conditional commands based on the result.

display information

The text labels of bar buttons, menu items, a PowerPro tray icon, (and also the tooltips for each of those) can be set using the configuration dialogs or using CL commands. In all those places you can show static text or dynamic *Info. There are special keywords for *Info displays, for items such as date, time, resources. *Info interprets those keywords (or any expression) and displays the result, updating it regularly.

You can also show message boxes, immediately or as scheduled alarms.

summary

PowerPro makes it possible for you to enjoy new powers and flexibility in how you manage your PC, which otherwise would only be attainable by expert programmers. The number of features may seem overwhelming at first, so it is best to explore them gradually.

Overviews and features

Suggestions for configuring PowerPro

Here are some ideas you may find useful when you are planning how to configure Powerpro to suit the way you want to organize your programs and desktop.

With Powerpro, you decide the way you find most convenient to run programs and Windows configuration commands.

First, you should decide how you want a Powerpro bar to appear. Most people use a bar to launch their most frequently used commands, either directly from buttons or from menus attached to buttons. You can use buttons to show special labels, like time/date, timers, or resource usage. For example, you could have one button that displays a timer associated with ISP usage; clicking on the button could show a menu of Internet-related commands. Some users have several small bars; some have none at all, preferring menus shown from hotkeys.

A menu or a bar is usually defined by a command list. Remember that you can also create menus or bars from any folder or from special folders like your desktop icons. The Start Menu and its sub-menus, desktop icons, and MS IE Explorer Favorites are really just folders of shortcuts which you can also access with the *Menu Folder command; look in your Windows directory (95/98) or in the Documents and Settings folder (2000/XP) for Desktop, Favorites, Recent and Start Menu.

After you have chosen a bar configuration and any associated menus, decide which commands you want to attach to hot keys. If you are mainly a keyboard (as opposed to mouse) user, set up ordinary key strokes (modified by Alt, Ctrl or Win) or tap keys to launch common commands or display menus. If you mainly use the mouse, you may want to attach a menu to "right hold", similar to the Window Menu of the default starting bar, which gives you immediate access to commands or Windows configuration features. Hot keys for clicking on portions of the window caption are also useful for commands like closing or tray minimizing or rolling up windows.

If you have one, the middle mouse button can serve many functions with Powerpro.

If you want to regain the screen space used by the task bar, look into making your own much smaller substitute by using Active buttons on a bar. Or you may prefer to switch tasks using a menu of active tasks.

On the other hand, if you like the task bar, you can position a Powerpro bar there or set up commands to run from your own tray icons.

If you have scheduled tasks or like to use your computer to remind you of things you need to do, use alarms.

Lastly, investigate the GUI configuration tab: it gives you many features for making Windows easier to work with, such as disabling the scroll lock key, automatically moving the mouse cursor to default buttons, and panning partially hidden windows into view when the mouse is over them.

Overviews and features

How one user has configured PowerPro

Each user discovers his or her own way to use Powerpro, to suit their individual needs and preferences. Some examples may be seen in the user group's Files section at: http://groups.yahoo.com/group/power-pro/Files

Here is a description of one user's PowerPro configuration.

"I have the button bar positioned on the right side of my screen with 8 buttons showing. As small as its footprint is, when not in use Powerpro hides itself, conserving virtually all 'screen real estate'. When I place my mouse at the screen edge, Powerpro's bar appears.

Each button may be left-, right-, or middle-mouse button clicked to execute the 'command' for which that button selection is user-programmed.

My top button is set up to show the time on its face. When I left click it, it brings up a custom menu I have created. The menu contains various commands and sub-menus. When I middle click it, it launches a suite of apps I call my 'home page.' When I right click it, it brings up my Start Menu.

When I press another button, it launches a menu that allows me to change screen resolutions on-the-fly using PowerPro's built-in resolution switcher. I have one button set up to dial my ISP when I left click it, and hang up my connection when I right click it. The dial/hangup facilities are contained within Powerpro (see dundial.txt in Powerpro folder).

I have a button that shows me all running apps, another that lists the last several recent commands. I can set up a button or menu to 'launch' a directory or any subdirectory on any drive. I can switch to or kill apps with a simple button click.

Powerpro has a built-in scheduler. I use it to launch Net Attache at one time, Diskkeeper at another, etc. The scheduler is the only one I run continuously, so there is minimal cpu overhead when compared to running individual schedulers for each app. In addition to running specific apps, services, or batch files, the scheduler is easily programmed to play a sound of my choice when scheduled events are triggered (i.e., it may be used as an alarm).

I use PowerPro's built-in Key/Mouse programming capabilities for all my hot keys and special mouse commands, including double left click when I press the middle mouse button. If I effect a control+left mouse button on my screen background, it launches my Start Menu. If I do a control+right click, it launches my custom main menu with its commands and submenus. If I do a control+left click on an app, it rolls the app up to show only the descriptor bar. If I control+right click on the app's descriptor, it unfolds. Although I haven't employed them, I can program my mouse to do whatever using the alt, shift, and 'Win' keys as well.

When I left click one of my buttons, it launches a menu with a list of text phrases/paragraphs I commonly use. Using the mouse to select on of the menu choices, sends the text to the active app. I happen to have two text strings I use a lot. These are sent to my active apps using a middle or right click on the same button.

Tutorials using the initial configuration

Test driving the initial button bar

This topic is for new users who have not yet changed much from the default configuration, as setup by a first-time installation of PowerPro. By now, you may have changed the bar configuration with the initial configuration wizard, but this test drive still covers important information.

The default button bar should be shown in the upper left of your screen.

Move your mouse over the bar without clicking. A small tool tip help window will appear, showing you which commands have been associated with each button. There are three commands per button: left mouse-click, right-click, and middle-click -- if you have a two-button mouse, you can simulate middle click with right+left at once (called a chord) or with shift-left click, or with any hotkey combination of your choice.

Notice that left clicking on the leftmost button will activate the configuration wizard and right clicking shows Powerpro Help (which you are viewing now).

Now try left clicking the button marked Edit. Notepad should start. Right click on the same button and System Editor will start. Close both programs.

Left click on the button marked Menu. A menu of commands will appear. At the top are four submenus for working with your active tasks: select an entry from the appropriate submenu to switch to, put always on-top, take off on-top, or close any program currently running on your system.

Right click on the Menu button to directly access your Start Menu Programs items.

The initial configuration also puts a tray icon with the Powerpro icon on your task bar. Right or left click it to show a message.

Finally, the initial configuration will also let you set and change hot keys/mouse actions. You should try some of these as this is a key PowerPro feature. Left click the first button to run the configuration wizard again to access the initial configuration hot keys.

The main configuration dialog, provided by the separate program pproconf.exe in your powerpro folder, allows for more options than the wizard. See Starting pproconf for various ways to start pproconf.exe.

With the full configuration dialog, you can

- add more bars or menus by creating command lists for them
- change or remove the command list tray used to display tray icons
- add new commands for hot key or mouse actions
- add scheduled events
- enhance the Windows interface using GUI Control or Setup tabs.

If you would like to start using the full configuration dialog, the next 5 chapters have some more help on using it to configure the default bar to suit your tastes:

Changing the tray icon

Changing button label/icon

Changing bar look

Changing bar size and position

Adding a new button

Tutorials using the initial configuration

Changing a label and an icon on the initial bar

Here is how to change the icon on the files button on the default bar. While holding down the Ctrl key, right mouse click anywhere on the Powerpro button bar and select *Configure*. This will activate the configuration tabbed dialog. The Command Lists tab will be activated and the drop down box will be set to the command list for the bar.

Since we want to change the files button, double click on the item labeled "files". The item configuration dialog appears. It has controls for changing the button label and icon along the top, and controls for entering the command to be run for each of left, middle, and right mouse clicks at the bottom.

Suppose we want to change the label to "docs". To do so, overtype "files" in the Label edit box with "docs". Press *Apply* to see the effect.

You may want to eliminate the text label entirely. Just delete all the characters in the edit box.

To change the icon to that used by File Manager, which is the middle command, use the drop down box under "Icon" to select *Middle Icon*. Press *Apply* to see the effect on the button.

Press *OK* if you like the new label and icon. Press *Cancel* to go back to the original ones.

After selecting *OK* or *Cancel* from the button configuration dialog, press *OK* to exit from the tabbed configuration dialog.

Tutorials using the initial configuration

Changing the look of the initial bar

You can move the bar by left clicking and then dragging with the mouse button down.

To change the bar look, hold down the Ctrl key and right mouse click anywhere on the Powerpro button bar and select the *Look* submenu.

Try experimenting with the *Flat* and *Text Under* check boxes. You can also change the look of the bar by adding or removing the border and 3D sizing frame.

Use the Positions submenu to select a position or to lock the bar into its current position and prevent accidental moves or resizing.

Perhaps you would prefer a different color or font for the bar. Ctrl-right click the bar and select Configure to see the main configuration dialog. Make sure the *Command Lists* tab is showing and select the command list called "Bar" from the drop down. Press *Properties*. Check the *Own color* or *Own font* check box, and then use *Set* to change the color or font. Use *Apply* to preview. Try setting the gradient to a number between 10 and 250 to set a gradient look for the bar (set 0 for no gradient). Press *OK* when you find the look you want, or *Cancel* to return to the default.

Tutorials using the initial configuration

Changing the size and position of the initial bar

Ctrl-right click the bar and make sure the Floating position is selected. Then, you can position Powerpro by left clicking and dragging the bar.

Ctrl-right click the bar and note setting of *Look*, *"Bar size from sum of buttons"*. If it is unchecked and *3D Frame* is checked, you can change the size or orientation of the bar by left clicking the bar sizing border and dragging to the new size or shape.

If *Bar size from sum of buttons* is checked, PowerPro sets the size of the bar and arranges for the bar to be just big enough to accommodate all buttons.

You can also select other positions. Ctrl+right click bar to see menu and review *Help on Bars*, *Bar Position*. Then use the *Positions* submenu.

Tutorials using the initial configuration

Changing the tray icon from the initial bar

You can use the initial configuration wizard to change the tray icon set up by the initial bar. Or, if you prefer to start using the full configuration dialog, here is how to do it:

to remove the tray icon

- Ctrl+right button bar to see configuration dialog.
- Show Command Lists tab.
- Select *Tray* from the dropdown list.
- Delete entries in list by selecting and then using Delete button (or keying Del).
- Press OK to save new configuration.

to change the tray icon commands

- Ctrl+right button bar to see configuration dialog.
- Show Command Lists tab.
- Select Tray from dropdown.
- Change the icon using the controls under icon info.
- Change any of the commands using the command entry controls under left, right, middle.
- Press OK to save item configuration dialog.
- Press OK to save new configuration.

Tutorials using the initial configuration

Adding a new button to the initial bar

using the button bar in the default config

While holding down the Ctrl key, right mouse click anywhere on the Powerpro button bar and select *Configure*. This will activate the configuration tabbed dialog.

The command list tab will be activated, with the drop down box already set to the command list for the bar you clicked.

Select the last button in the list and click *Add after*. Enter the new button label and commands on the resulting dialog. Click *OK*, then click *OK* again to save the new configuration.

If you have checked *Bar size from sum of buttons* in the command list's *Properties* dialog, you may need to drag the left side of the bar to resize the bar and see the new button.

Demonstrations

Demonstrations and samples

There is a demonstration configuration of PowerPro which illustrates many features. To start this demo, Ctrl-right click on any bar, select the *"Change configuration"* menu item, and then select *"demo"* from the resulting submenu.

The demo bar will appear in the top left of your screen.

The following 12 chapters in this section describe each demonstration:

Menus and context menus

Subbars and manually shown bars (*Bar Show)

*Window commands

Sending keystrokes

Hotkeys and mouse actions

Menus of file folders

Keyboard macros

Run commands when a window first opens

*Format item and *Info

Bar showing other bars when button clicked

Bar sections and subbars

Virtual desktops

recovery

When you are finished with the demo, ctrl-right click on the bar and select "Change configuration" menu item, and then select "pproconf" from the resulting submenu to restart your configuration. Make sure *>Command Lists >Setup >Special Lists >"Run Monitor List"* is unchecked before switching back.

localising the demos

Since the location of command files will be different for each computer, many of the demonstrations use standard Windows commands like notepad.exe or internal commands like *Message. Or course, you can replace these by any file or command on your system when you use the feature.

Demonstrations

Demonstration of menus and context menus

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select *"Change configuration"* menu item, and then select *"demo"* from the resulting submenu.

Left click the Menu button to see the command list named MenuSample displayed as a menu. This menu illustrates submenus and context menus. Note the Wallpaper and Screensaver commands in the submenu. To see a context menu, start notepad by clicking the Notepad button on the bar. Press the Menu button when notepad is running and is the foreground window (dark blue caption) and note that these entries appear to send keys and to show a message. Try again with notepad running but not active (e.g. click on desktop) and you will see a different menu.

To see how this menu is configured, ctrl-right click bar, select configure, and select command list MenuSample from the dropdown on the *Command Lists* tab of the configuration dialog. You will see on the command list MenuSample that the Paper/Saver submenu is started by *Format StartSubmenu and ended by *Format EndSubmenu in the command list. The context submenu starts with *Format Context *Notepad* and ends with *Format EndContext. Because they are within the context section of the command list, the *Keys and *Message commands between these two commands will only be displayed if Notepad is the active window.

You can also display the MenuSample command list as a menu by pressing key ctrl+alt+m or by shift+ctrl+right clicking the mouse. See the *Key/Mouse* tab of the configuration dialog and note the *Show Menu command associated with both these hot keys. This illustrates that the menu structure (i.e. the command list) is separate from how the menu is shown (by clicking a button or using a hot key, for example)

Right click the menu button to see the SubbarMenu displayed. Selecting an entry displays a different subbar.

Middle click (shift+left click for two button mouse) the Menu button to see the WindowMenu which displays active programs and allows you to switch to, close, OnTop, or NotOnTop the windows. Access the configuration dialog with ctrl-right click on the bar, and note the command list WindowMenu. See how the *Window commands are placed within *Format StartSubmenu and *Format EndSubmenu commands so that the lists of active windows do not follow each other on the main menu.

You can activate the snippets menu by using the hot key alt+ctrl+s.

Right click on the system icon in the upper left caption of any window to see the ControlWin menu.

Demonstrations

Demonstration of subbars and manually shown bars

subbars

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demo" from the resulting submenu.

Ctrl-right click the bar, select configure, and view the command list Bar. It shows three subbars: edit, util, and none. You will see these names on the *Format StartSubbar commands. To see these subbars in action, right click the menu button and select a subbar name from the menu. This will execute the corresponding *Bar SelectSubbar command; these commands are on the command list SubbarSelect.

Tip: do not forget the @ sign on your SelectSubbar commands!

manually shown bars

You can also manually show whole bars. For example, the command list ManualBar can be shown as a bar by right clicking the notepad button. View the configuration for the command list bar and note the *Bar Show ManualBar for the right command for the notepad button. Also note that *"Auto show as bar"* is not checked for this command list in the configuration of command list ManualBar. Finally, you can use the Properties for the ManualBar command list to select own colors and font. When you are finished with the demo, close the ManualBar bar by ctrl-right clicking it and selecting Close Bar from the menu.

Demonstrations

Demonstration of *Window commands

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demo" from the resulting submenu.

*Window commands let you manipulate active windows. Ctrl+right click the bar, select *Configure*, and view the *Keys/Mouse* tab. You will see many *Window commands among the hot keys.

Start notepad using the button on the demo. Right click on the minimize box in the caption and the window is minimized to the tray; click the icon in the tray to restore. The **right-minimize** hot key function in the configuration performs this function. Right click the caption to close; the **right caption anywhere** hot key does this. Restart notepad and right click the maximize button to rollup the window to its caption. Right click maximize again to restore.

You can also execute *Window commands from keystrokes: for example the ctrl+alt+k hotkey has the *Window close *Notepad* command attached, which closes any window with Notepad in its caption. It will produce an error message if notepad is not running because *"error if no such window"* is checked on the configuration for the command. Try ctrl-alt-k both with and without Notepad running.

You can also execute *Window commands from menus. Right click the system menu icon in upper left of caption to see a menu of *Window commands; this is command list control win activated by hot key right sys menu.

window menu

*Window commands can also display menus of active tasks; selecting one item performs the specified action. Middle click (shift+left) the menu button to see the WindowMenu menu which allows you to switch, close, put on top or put not on top any of your running programs. Start notepad using the demo bar to try these on if you have no other programs running.

You can also use the *Window action menu command directly from a button or a hot key. Right clicking a folder executes a *Window show menu (for switching among windows) and tapping ctrl twice quickly shows the *Window close menu (select an entry to close that window).

Demonstrations

Demonstration of sending keys with *Keys

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demo" from the resulting submenu.

You can use the *Keys command to send keystrokes to windows in order to save typing. You can also use this command to automate a program function, often by sending an alt-key to open a menu and then another keystroke to choose a menu option.

The Snippets menu in the Demo config illustrates both capabilities of *Keys. One way to activate it is with ctrl-alt-s. Or, if you prefer a mouse, you can double click the right mouse button.

Start notepad and ensure it is the active window. Then use ctrl-alt-s or right double click to show the menu. You can select a menu item to send keys by clicking on it, by using the underlined menu mnemonic (created with & in snippets command list item), or by using down arrow and enter.

The last two menu items show automation of Notepad functions *print preview* and *select all/copy to clipboard* by sending keystrokes. These have been placed on a context menu which only appears when notepad is the active window. You would often place a series of such context menus for automating different functions in the same command list. For example, if you activate snippets when explorer is active, a different set of commands will be shown.

To view how the snippets menu is configured, ctrl-right click bar, select configure, and select snippets from drop down on command list tab.

Demonstrations

Demonstration of hot keys and mouse actions

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demo" from the resulting submenu.

The sample bar illustrates many different ways to use hot keys/mouse actions. Ctrl-right click the bar, select configure, and select the Key/Mouse tab. Remember as you review these samples that you can use whatever key/mouse combination you find convenient to perform any of the sample actions or in fact to run any Windows file or perform any PowerPro built-in command.

A basic use of hot keys is to start Windows programs. For example, the hot key shift-ctrl-x starts explorer. Use it now to open Explorer.

Now position your mouse over explorer and make sure the window is the active window by clicking on it if needed. Hot key ctrl-d will sort file names by date; it works by sending keys to explorer to select the appropriate menu item. Hot key ctrl-n sorts by name. Both of these hot keys will only operate if explorer is the active window as indicated by *Exploring* which appears in the target window of the hot key configuration and which selects only windows with Exploring appearing somewhere in their caption. You can achieve the same results as the keystrokes with mouse actions: right click and drag horizontally for about 10 pixels for the name sort or vertically for about 10 pixels for the date sort. It may take a bit of practice to get the short drag needed to activate the hot key.

Hot keys also work well with *Window actions.

Hot keys can be used to show menus of choices by running a *Menu Show command.

Demonstrations

Demonstration of *Menu Folder

If you have not already done so, start the demonstrationdemo_all configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demo" from the resulting submenu.

The following examples use names of standard folders on your computer. You may have different names for some of them or your file names may not be in English. If so, you will need to change the folder names to those of your computer before the sample will work for you.

For an example of *Menu Folder, press the button marked *folder* and a list of all folder and files on your c drive will appear. Click any folder or file to activate it. To see the command which generates this menu, ctrl-right click bar, select *Configure*, ensure the "Bar" entry is selected on the *command lists* tab, and double click on the folder item in this command list. Note the format keywords associated with the *Menu Folder command. These are set by pressing the small Find button beside the edit box.

With this first example which shows all of drive c, you have to click on a folder to see the files in that folder. PowerPro can display at most 13000 files and many people have more than this on their c drive. You can also ask PowerPro to display files in a cascading menu. Middle or shift+left click the Folder button to see an example. You will have to wait for a few moments for the menu to appear. It will show all .exe files under c:\program files. If you view the command configuration, you will see how format keywords are used to select just .exe files.

You can also use *Menu Folder with hot keys. For example, ctrl+space shows c:\my documents. And if you bump your mouse against the left screen edge and hold it at the edge for half a second, the shortcuts in c:\windows\start menu\programs will be shown.

Demonstrations

Demonstration of keyboard macros

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demo" from the resulting submenu.

Keyboard macros let you use abbreviations for text you commonly type. To set a group of macros, you create a special command list where each item name is the abbreviation (letters, digits, spaces only) and the left command is a *Keys command, or a *Clip textpaste command, to send the keys. The *Clip textpaste command is faster for longer text but it will overwrite the clipboard.

Ctrl-right click the bar, select configure, and view the command list tab. The command list MyMacros shows sample abbreviations for **me**, **test**, **ad** and **ac**. Ad and ac send the same text but ac uses the clipboard method.

You can use macros to execute any command, not just to send keystrokes. In the sample **xx** starts explorer and **min** minimizes the current window.

To tell PowerPro that the MyMacros command list has this special usage, you must define a macro signal character with a hot key. If you view the key/mouse tab, you will see the = is defined as a hot key to execute the *Macro MyMacro command. This makes = the macro signal character.

To test, use the button to start notepad. Try =me, =test, =ac, =ad and note the results. Try =null and note that the text is unchanged since this is not a defined macro. Then try =xx and =min.

Demonstrations

Running commands when a window first opens

If you have not already done so, start the demonstration configuration: Ctrl-right click on any bar, select *"Change configuration"* menu item, and then select *"demo"* from the resulting submenu.

You can configure PowerPro to check each new window that is opened on your system and run a different PowerPro command for each specified window. Any command can be used, but often this feature is used to press a button on the window, to send text to the window, or to position the window on the screen.

You use a command list to specify the windows to be matched and the corresponding command. Ctrl-right click the sample bar, select *Configure*, and view command list NewWindow.

To activate the feature, you must first press the >*Command lists* >*Setup* button and select the command list "NewWindow" in the drop down labelled *"use this command list to run commands when a new window first opens"*. Press OK to save the configuration when you have done this then re-open the configuration dialog with ctrl-right click. View the list NewWindow.

The command item names are used to match newly opened windows. The item ***exploring*** matches any new explorer window and positions the window in the center of the screen. Start explorer to see the effect. (You can start explorer from the Start Menu or by shift-left clicking the notepad button). The **Untitled - Notepad** item matches every newly opened notepad window and sends text abc to it. Try running notepad from the button. Finally, the command associated with About Notepad presses the OK button as soon as this window opens; try selecting *About Notepad* from the notepad *Help* menu to see the effect. You can press the default button of dialogs by sending the **Enter** key stroke using the command: *Keys {enter}. You can press other buttons on a dialog by sending alt-x (where x is the underlined character on a button name) using: *Keys %x.

You can temporarily disable an item in the command list by checking the *Hidden* check box.

When you are finished with the demo, be sure to go back to *>Command Lists >Setup* and set the drop down box *"use this command list to run commands when a new window first opens"* back to (none) so that this NewWindow sample does not interfere with other samples.

Demonstrations

Demonstration of *Format Item and *Info

Start the demonstration configuration if you have not already done so using the Shutdown and Start Demo buttons.

This demonstration illustrates *Info and other special item labels, *Format Item, and Script programming.

The demonstration shows how to display a bar or menu of resources and other information. The command list called Resources defines the information to be shown on each button. View the list by ctrl-right clicking the bar, selecting configure, and selecting Resources from the dropdown on the *Command Lists* tab.

To display the Resources list as a **menu**, use the hot key Ctrl-Shift-R. The menu could be tidied up by removing icons from the command list properties and by removing the last five command list items. But these are used for the bar and so have been left in the command list.

To display the Resources list as a **bar**, press Ctrl-Alt-R. Press it again to hide the bar.

using the *Info feature

Press Ctrl-alt-r and note how the Resources command list displays as a bar:

top row shows gdi/user (Win 95/98 only), physical memory free, percent physical free.

second row shows Windows uptime in hours:minutes and in days.

third row shows free space on the C drive in megabytes.

fourth row shows swap file size and percent of swap file in use.

fifth row shows total characters received and the rate of characters received on DUN connection. To see the fifth row resources change, dial a DUN connection and download a small file or news item.

Each button shows dynamically updated information. This is set by typing *Info, followed by special Info keywords, as the Name of each item.

Note: Info is not a PowerPro built-in command. It is never used in the Command text box for an item, only in the Name text box for a bar or menu item, or in a tooltip, or as the text of a tray icon.

using *Format Item

The sample Resources bar also illustrates updating an item with the *Format Item command. The updating is performed automatically by a script called Monitor. Note: The Monitor command list is never actually shown as a bar or menu; instead it is run as a script.

Ctrl-right click the bar, select *Configure*, and view the command list called Monitor. We will be arranging for this command list to run repeatedly once per second to update the item 9 (the DUN displays) and items 11 through 15, which will be a small bar graph of download rate, on the Resources bar.

The first two lines in the script use the command *Format item to hide the icon on the ninth item in the Resources command list only if there is no modem connection. (Note: Resources script item 9 assumes there is an icon in c:\windows\notepad.exe).

The next three lines set the color of the ninth item if there has been no download activity on the modem for more than 5 seconds. A variable v is set to the number of seconds of idleness (*dunidle) and then used to control which *Format Item is executed to set or clear background color. Note that the & is set as the variable character on >Setup >Advanced.

The last statements of the Monitor command list create a small bar graph which shows the download rate. This is meant as a fun illustration of what *Format Item and programming can do: other tools such as Microsoft System Monitor are better suited to producing graphics.

To see the icon and bar graph in action, right click the Recd/rate button, which uses the *Exec Monitor Reverse command to stop/start the repeated execution of the monitor command list. Then try downloading on a DUN connection.

When you are done with the menu, ctrl-right click a bar and select quit.

Demonstrations

Bars showing other bars when a button is clicked

You can configure a bar to show other bars at your mouse when you click a button. For an example, try the BarShowsBars demo configuration which you should find installed in your Powerpro folder. You can test it by ctrl-right clicking a bar, selecting Change Configuration from the resulting menu, and then selecting BarShowBars from the resulting submenu.

Click any button. Another bar will appear with commands to be launched.

This effect is configured by creating a separate command list for each of the bars which appear when you click the main bar. Each of these other bars are autohide bars (which is set in each command list's Properties) and *"Auto Show as Bar"* is checked.

Ctrl-right click the bar and note the configuration of items on the "Bar" command list. The Saver button shows the command list ShowOne at the mouse. The other buttons use the *Bar SelectSubbarToButton command to show a subbar of the ShowBar command list aligned with the clicked button. The advantage of the subbar approach is that only one command list (beside the main bar) needs to be maintained.

Note how the ShowAll and Showone bars use the vertical slide setting on the Command List Properties to control slide animation.

To show the vertical bars when the mouse hovers over the main bar without clicking a button, select Properties for the Command List called Bar and check *"hover clicks"*.

To return to your standard configuration, ctrl-right click bar, select Change Configuration menu, submenu item pproconf.

Miscellaneous usage topics

Versions of Windows

A few PowerPro features behave differently in 95/98, vs NT/2000/XP

Powerpro is a 32 bit implementation. There is one version of the program which runs in all 32 bit versions of Microsoft Windows from 95 to XP.

The following restrictions apply to running PowerPro in Windows NT, 2000 and XP (collectively referred to in this Help as WinNT) resulting from limitations as compared to Windows 95, 98 and ME (collectively referred to as Win95/98).

- In WinNT, Windows GDI and User free resources are always reported as 99%.
  Note: In WinNT systems the GDI and User resources are not limited to a fixed size as they are in 95/98; therefore showing the free percentage of these resources is not meaningful.
- Powerpro does not handle special font/color settings for console applications in NT.
- The *Menu UnderMouse command is not supported in WinNT.
- Windows NT is capable of running 16 bit programs in separate Virtual Dos Machines (VDMs). To do this in Powerpro, set up the command and parameters as follows:

  Command: cmd

  Parameter: /c start /separate c:/app_path/16bitapp.exe parameters

  Note: In Win95/98 the DOS window uses command.com, instead of the cmd.exe used in the WinNT systems.

You may discover other variations in how PowerPro functions in different OS versions and on different hardware. For example the Shutdown commands include Logoff, Suspend, Hbernate, etc. These can behave differently depending upon the operating system (98 or NT etc), the capabilities of the hardware, and the Control Panel power options.

Miscellaneous usage topics

PowerPro command line

running powerpro.exe with parameters

starting PowerPro with a different .pcf file

Powerpro normally uses the configuration file **pproconf.pcf**, found in the same folder as the powerpro.exe file.

You can use a different file name or a different folder by putting the path to the configuration file on the Powerpro command line. If the configuration file is in the same folder as powerpro.exe, you can omit the path. For example:

```
"c:\program files\PowerPro\PowerPro.exe" "C:\My Documents\PowerProConfigs\MyOther.pcf"
```

If you use a shortcut to start Powerpro, the command line can be found in the shortcut properties. Note that you must put double quotes around file paths which contain blanks.

a config for each User

You can make the folder depend on the current user by putting a % in the path to the configuration file; the % will be replaced by the current user name. For example, if Ralph was signed on:

```
"C:\My Documents\PowerPro\%\PowerPro.pcf"
```

would be interpreted by PowerPro as:

"C:\My Documents\PowerPro\Ralph\PowerPro.pcf"

note

As well as pcf files, Powerpro puts all files which it can change into this main folder: the timer log, alarm log, clip folder, tray icon info, saved desktop icon positions, saved explorer windows (from explorer tracking option). So if you want to move your current configuration and other data files from the Powerpro folder, you must move all .pcf files, all .iconpos files, all .ini files, all .timerlog files, all .alarmlog files and the explorer.windows file.

relative paths

For .bmp backgrounds, script files, icon files, and shortcuts run as commands:
if a relative path like shared\run.lnk or back.bmp is used for these types of files, PowerPro assumes the file resides in the same folder as the configuration file. This can be used to collect files associated with a configuration in the configuration files folder.

starting with some flags set

You can also use the command line to set flags at start up. Precede the pcf file (if present) by -fn, where n is the flag number to set. The letter f must be in lower case. Repeat -fn to set multiple flags. For example, the following sets flag 6 with the standard pcf file:

"c:\program files\PowerPro\PowerPro.exe" -f6

running a PowerPro command

All of the command line parameters for powerpro.exe which are described above require that powerpro.exe is **not** already running, because they are used to start Powerpro.

By contrast, you can use a different type of parameter – any Powerpro command. This can only be used **after** PowerPro has started. It is a way of using a command line (maybe from a Windows shortcut, or from the Windows' *Run* dialog) to send a command to the currently running PowerPro. Details are in the next topic

Miscellaneous usage topics

Running built in commands from the command line

running powerpro.exe with parameters

You can send PowerPro commands to the running program from a command line. This could be used to execute PowerPro commands from shortcuts or batch command files.

Note: For this to work, PowerPro must already be running. Parameters for the powerpro.exe command line when powerpro is **not** already running are on the previous page.

Type the built-in command, action, and parameters on the powerpro.exe command line.

For example

C:\program files\PowerPro\PowerPro.exe Menu Show MyMenu

C:\program files\PowerPro\PowerPro.exe Menu.Show("MyMenu")

tells the running PowerPro program to show menu "MyMenu".

...\powerpro Exec.ChangeConfiguration("c:\henry\mynewconfig.pcf")

tells the running Powerpro to start using a different configuration file.

You can also use plugin calls, script calls, and assignment statements on the command line.

For example

```
C:\program files\PowerPro\PowerPro.exe .myscript@start(arg1, arg2)
C:\program files\PowerPro\PowerPro.exe myvar1 = "new value"
```

Miscellaneous usage topics

Using the middle mouse button

PowerPro can extend the usefullness of a middle click in various ways

If you have a three button mouse, you may want to use the middle mouse button as follows:

To send double left clicks, set the *middle anyw*here hot key to this command:

Command: *Mouse

Parameter: leftdouble

Or you could attach other commands or menus to the *middle anywhere* or *middle hold* hot key. You can set further hot keys for the middle mouse by using a modifier key like Ctrl or Win.

In addition to the hot key, you may also want to use it either for scrolling or for moving a window by setting the option on the *GUI Control* configuration tab.

See also: Manual scrolling with the mouse and Automatic scrolling with the mouse

See "Key/Mouse" dialog for other mouse actions you can customise, such as mouse gestures and using the extra buttons on five button mice.

Miscellaneous usage topics

Minimizing a window to the tray

which saves space on the Windows Taskbar

If you run many programs at once, you can reduce task bar clutter by minimizing a window to the tray. When you minimize to the tray, Powerpro creates a tray icon for the program and minimizes and hides the window. Clicking on the tray icon restores and activates the program. Right clicking on the icon shows a menu allowing the program to be restored, maximized, or closed.

There are three ways to minimize to the tray:

- use a *Window traymin command attached to a hot key or bar button;
- or place the caption or exe file name in the *Auto tray min* edit box on the *Setup* configuration dialog;
- or set the program to start initially as tray minned in the command entry controls.

example

One way to minimize to the tray is with this command:

Command: *Window

Action: traymin

Parameter: under

You could attach that command to a hot key corresponding to right-clicking the minimize box.

auto try min

You can replace normal minimization to the task bar with minimization to the tray by using the *Auto tray min* edit box on the *Setup* configuration dialog. Separate entries by commas.

If the entry in the edit box **ends** with a *, then windows with captions starting with the characters before the * will be minimized to the tray; if the entry **starts** with a *, then windows ending with the characters following the * will be minimized to the tray. Finally, you can specify the windows to be minimized by using =filename (omit path and the .exe extension) to work with windows of the program filename.exe. For example, if

Exploring*,=notepad

is put in the edit box, then minimizing Explorer windows or Notepad will put them in the tray instead of a normal Taskbar button.

a different icon

Normally, Powerpro uses the icon of the minimized program as the tray icon. But you can change this behavior and select any icon by creating a special command list.

Set up a command list item in this special command list for each new icon you want to use. Set the command list Item's name to match the caption of the window to be tray minimized, and set the command list Item's icon to the desired icon. The command list item commands can be left at (none). Use *xxx as a command list item name to match windows with captions ending in xxx, yyy* to match those starting in yyy, and =ExeName (without the .exe extension) to match all programs called ExeName.exe.

Finally, let PowerPro know that this is a special list by using the *Tray* drop down box on the *Command Lists* > *Setup* > *Special Lists* dialog to select this command list.

vdesks

If you are using virtual desktops (vdesks), showing a window from its traymin icon will also switch to the virtual desktop it was part of when it was tray-minimized.

Miscellaneous usage topics

Changing Explorer list and view settings

choosing how Windows Explorer starts

You can affect the view (large icon, small icon, details, list) and sort by (date, name, type, size) settings for Explorer in two ways: [a] you can force the settings for all cases using drop down boxes on the main *Setup* configuration dialog, and [b] you can change the settings for specific cases by sending keystrokes to Explorer windows.

all cases

To force the same settings for all newly-opened Explorer windows, use the drop-down boxes on the *Setup* configuration dialog Set the first drop down to *No, Single, Double,* or *All* to select which types of Explorer Windows to force. Then select the desired view and arrangement options.

These forced settings will normally override all folders (including the last 50 opened, for which Explorer stores a setting) but if you hold down the shift key while opening the new window, Powerpro will not override the Explorer settings.

specific cases

For a convenient way to change the settings for Explorer windows while you are working with them, use the *keys command to send keystrokes to the active window.

Of course, you could use Explorer's tool bar instead, but the advantage of getting PowerPro to do it for you is that you can attach the following command to a PowerPro hotkey, or even run it automatically from a PowerPro script.

For example: *Command:* *Keys
Parameter: %vid

sends **Alt-V**, then **i**, then **d** to the active window. That would set sorting by Date.

You could attach the above command to a hot key, or to a button on a small bar which is only shown while an Explorer window is active. Read Window-specific menu and bar contents about program-specific bars.

alternative

You can also start Explorer at a specific folder and with specific settings as follows:

Command: c:\windows\explorer.exe
Parameters: /select,D:\Program Files\eudora 3\Attach
More cmds: *keys {to +*attach}%vg

That command launches Explorer and uses the Explorer command parameters to select folder D:\Program Files\eudora 3\Attach. It then sends key strokes **Alt-v g** to select large icon settings.

The + in {to +*attach} tells Powerpro to wait for up to 5 seconds until a window with caption ending with **attach** appears, before sending the keys.

You could create a menu of commands like the above for favorite folders.

If you send keys like that to Explorer when it is launched from Powerpro, those settings will replace any settings for all Explorer windows forced by the *Setup* dialog.

Miscellaneous usage topics

Using the keyboard to access a button bar

Instead of clicking bar buttons with the mouse, you can use the **keyboard** to access the items on a Powerpro bar.

First, you need a way to ready the bar for input from the keyboard.
Set a hot key to the following command:

Command: *Bar
Action: Keys
Parameter: name of command list for bar

When you activate that hot key, the mouse cursor will be moved to the specified bar and the bar will be ready to receive any of the following **keystrokes**:

L	equivalent to left-clicking current button (you can also use Enter)
M	equivalent to middle-clicking current button
R	equivalent to right-clicking current button
left arrow	move to next button
tab	move to next button
right arrow	move to previous button
end	move to last button
home	move to first button

up arrow	move to next row in multi-row bar
down arrow	move to previous row in multi-row bar
Ctrl+Enter	show configuration dialog
Esc	return the mouse cursor to the position preceding the *Bar Keys command.

Miscellaneous usage topics

Scrolling with mouse movements

You can scroll windows vertical or horizontally using mouse movements. This avoids having to move the mouse to the scroll bar to scroll the window. You can scroll either automatically or manually. Automatic scrolling scrolls the window even when the mouse is not moving; manual scrolling requires mouse movement to scroll the window.

Click next 2 pages for details of automatic scrolling and manual scrolling.

Miscellaneous usage topics

Manual scrolling with the mouse

Even if your mouse doesn't have a middle wheel, just a plain middle button, you can start manual scrolling in one of two ways: by attaching a command to a hot key/mouse action or by using the middle mouse button.

with a hotkey

To start scrolling with a hot key, attach this command to the key, using the *Key/Mouse* dialog:

Command	*Exec
Parameter	scrollwithmouse

Scrolling only occurs for the window which the mouse is over when the hot key is activated. Scrolling continues until the left mouse button is clicked.

with the middle mouse button

To set up middle mouse scrolling, use the *GUI Control* configuration dialog. Check the *"Middle mouse scrolls"* checkbox there to scroll only while middle mouse is down; gray-check to scroll with middle mouse up until left button clicked.

To scroll a window, either activate the *Exec scroll hot key or hold down middle mouse and move in the desired direction. For ordinary check, scrolling will continue even if the mouse stops. For gray check or the scroll command, scrolling will pause unless the mouse is near the top or bottom of the window. You can control or disable speed of automatic scrolling with the *"Milliseconds between scroll steps"* box in *>Setup >Advanced >Limits*.

To scroll pages (instead of single lines), click the right mouse button while the window is scrolling.

To quickly move to the start or end of the file, hold the Alt key down and move the mouse in the desired direction.

Mouse scrolling only works with applications that use standard windows scroll bars.

Some applications, such as Microsoft Internet Explorer, already support mouse scrolling. You can disable Powerpro scrolling for these (or any other window) by typing the caption of the window in the edit box beside the middle scrolling check box. Separate captions of different programs by commas. Normally, you will not type the whole caption, but rather only a part. Use *xxx to match all captions ending in xxx. Use xxx* to match all captions starting with xxx.

Use *zzz* for captions containing zzz anywhere. For example, *Internet Explorer will match IE windows.

Miscellaneous usage topics

Automatic scrolling with the mouse

To scroll windows automatically, execute this command from a hot key or mouse action:

Command	*Exec
Parameter	autoscroll

A small gray rectangle with the letter "S" will appear in the current window. Move the mouse above it to scroll up; the further the mouse is from the gray rectangle, the faster the window is scrolled. Move the mouse below the rectangle to scroll down; the further the mouse is from the gray rectangle, the faster the window is scrolled.

Right click to scroll a page. Middle click to scroll 5 lines. Left click to stop scrolling.

The slowest scrolling speed is set by *"Milliseconds between scroll steps"* on the *>Setup >Advanced >Limits* dialog.

Or, you can optionally set the scroll speed by setting the number of milliseconds between auto scroll steps with the *Exec autoscroll command. Use 0 to disable automatic scrolling; in this case scrolling is accomplished solely with middle click (5 lines) and right click (page).

Many newer programs support autoscrolling internally if you middle click on one of their windows. To automatically take advantage of them, define a hot key (say tap shift) which sends a middle mouse click to the programs which support autoscroll and executes the Powerpro command otherwise. To do this, define the hot key twice, and use the target window feature on the first definition.

For example, to use native autoscroll in Internet Explorer:

Hot Key	tap shift
Command	*Mouse
Parameter	middle
Target	*Internet Explorer

Hot Key	tap shift
Command	*Exec
Action	autoscroll
Target	

Miscellaneous usage topics

Mouse hover to click a dialog button

You can ask Powerpro to automatically press buttons
when the mouse is stopped over them for a specified time.

Use the check box *"Left click mouse when..."* on the *GUI Control* tab to do this. You can also set the stop time with the spin box. If you want Powerpro to automatically select standard menu items too, gray check the box.

You can use the *"except"* edit box to specify a caption list of windows where the automatic press is not to occur.

By default, Powerpro will automatically press buttons (including radio buttons and check boxes), combo boxes, combo box list items, standard toolbars, and tabs in standard tabbed dialogs. Powerpro also will automatically press the minimize, maximize, close, help, and system menu buttons in captions and will automatically open standard menus in menu bars.

You can add or remove window types to this list as follows: Assign the command

Command	*Exec
Parameter	autopress

to any hot key. (Avoid using Alt as a modifier key for the hot key as this will close open menus.)

Move the mouse over the window of interest and then activate the hot key. If the window type is not currently one that is automatically pressed, Powerpro will add it to its list. If it is one that is automatically pressed, Powerpro will remove it. In both cases Powerpro notifies you of the results with a message box.

If you use *"Cursor to default button"* from *GUI Control* tab, Powerpro will not press the button moved-to by this features unless you move the mouse from where Powerpro positions it.

See also

You can set any button on a PowerPro bar, or a menu's items, to be "clicked" when you hover over them for a set time. First you need to enable this for the bar or menu as a whole, using the "Hover clicks" checkbox in the command list Properties. You can then enable it for individual buttons using the Item editing dialog.

Miscellaneous usage topics

Adding items to the Explorer right click menu

the context menus, for files and for folders, in file managers

purpose

When you right click a file or a folder in Explorer (and in most other file managers) a context menu is displayed. Actually, Explorer has two different context menus: one for files and one for folders. PowerPro lets you add entries to either or both of those menus.

You can configure to display your context menu for all files and/or folders. You can also configure to selectively display certain menu items only when certain file names or types are selected.

configuration

First, you must install PowerPro **context menu support** by pressing the button on the *Setup* dialog tab of the configuration dialog.

Then you need to create a special command list to hold the items you want to add to the context menus. If you want to add items to the menu for files, you must create a command list called **Context**. If you want to add items to the menu for folders, you must create a command list called **ContextFolder**. The command list names must be Context for files and ContextFolder for folders.

The Context or ContextFolder item names and associated commands will be added to the right click menu for all explorer files or folders.

If you left-click one of the new items on a displayed context menu, the Left Command (as defined in the command list) will be executed with the selected path added at the end of the command line. For example, if you had an item command

```
*Wallpaper ChangeTo
```

and you selected file c:\path\mypaper.jpg, then the command executed would be

```
*Wallpaper ChangeTo c:\path\mypaper.jpg
```

If you select several files, the command is executed separately for each one -- in an arbitrary order.

advanced usage in scripts

If you select several files, and apply a context menu item to them, PowerPro sets two global variables to indicate whether the first or last file is being processed. The variable Context is set to 0 for the first file; you should set it to 1 when processing this file. The variable ContextLast is set 0 at the start of processing and set 1 just before the command is executed on the last file.

parameter placement with _file_ or | or ||

If you want more control over where the selected file appears in the command line, you have two options: use the _file_ variable or use the character |

using _file_

The selected file is assigned to the variable _file_ which you can then use in an expression, such as

```
do("notepad.exe", _file_)
```

which runs notepad on the selected file. To extract the folder or nametype of _file_, use the file plugin's services file.folder(_file_) and file.nametype(_file_). If multiple files are selected, then the command is executed repeatedly with _file_ reset to the next file in each iteration.

using |

If you want the selected file name to be placed in the midst of the command, put a | where you want the file name; for example

```
*File Copy "|" c:\standard\output.txt
```

will copy the selected file to c:\standard\output.txt. Note that you must include the quotation marks if appropriate when using |

If you are using multiple commands, **always** use a | to control placement; otherwise the file name will go at the end of the final command.

using ||

If you want the command to work on the folder path only, instead of the file, use ||. For example,

```
c:\prog\salamand.exe "||"
```

as a command will run salamand.exe with the folder path on the command line.

To use a literal |, precede it with a ', e.g..

```
unzip -lv "|" '| sort
```

You can use *Format to insert separators, columns, and submenus in the menu.

The command list's items will be inserted directly into the top level of the context menu displayed by explorer. Start with a *Format StartSubmenu if you want to insert the items as a submenu.

displaying different items for different files/folders

You can use *Format Context to make the items displayed depend on the file or folder selected. The file name is matched against the text parameter; for example, if the *Format Context text is *.txt, then the items after the *Format Context will only be displayed if the file selected ends in .txt. If you select several files, the first one determines the text matched against *Format Context.

You can use multiple *Format Context commands to match different types of files. You can use any of the *xxx*, *xxx, and xxx* patterns to match file names: *xxx* matches a file name containing xxx, *xxx matches a name ending in xxx, xxx* matches a name starting with xxx.

You cannot nest *Format Context commands; use a series of context commands for each condition instead.

restrictions

You can display at most 128 items. You can display at most 9 submenus. You cannot embed menu commands such as: *Menu Folder, or *Window action menu

context menus for shortcuts (.lnk files)

Be careful when using context menus with .lnk files. In some situations, the PowerPro context menu may not be displayed for .lnk files. In others, it will be displayed but the PowerPro command will act on the target of the short cut instead of the shortcut file itself. Some people have found that this behavior depends on whether the single-pane or dual-pane version of Explorer is used. You can test by putting the command

```
win,debug(_file_)
```

in your context menu and seeing whether the .lnk or target file is displayed.

Some people have found that importing the following into their registry by placing the text in a ,reg file and executing that file will cause the .lnk file itself to always be processed.

```
[HKEY_CLASSES_ROOT\lnkfile\shellex\ContextMenuHandlers\PowerPro]

@="{2EC41A81-AE31-11d2-A6E6-10E356C10000}"
```

example

Suppose the Context command list is set to the following items.

Item Name	*Command, Action, Parameters*
Purge	*Exec Prompt 1 Purge
	[in More cmds box] *Script if (pproflag(1))
	[in More cmds box] *File DeleteNoRecycle
Editors	*Format StartSubmenu
WordPad	C:\windows\wordpad.exe
Notepad	C:\Windows\notepad.exe
MyEditor	C:\Program Files\Myeditor\myeditor.exe
EndSubmenu	*Format EndSubmenu
Context	*Format Context *.bmp, *.jpg
Set as Wallpaper	*Wallpaper ChangeTo
Edit Image	C:\program files\image editor\image.exe
EndContext	*Format EndContext

Right-clicking on a file will display the standard context menu with these additional items: a **Purge** item and a submenu of editor selections. If the **Purge** item is clicked, a confirmation dialog is shown before the file is purged.

If the selected file is a .bmp or .jpg file, then items for wallpaper changes or processing with the image editing program will also be displayed.

Miscellaneous usage topics

Favorite folders lists for file open / save dialogs

PowerPro can help you maintain and use lists of favorite folders for standard open/save dialogs. You can manually maintain a list of favorite folders; you can have PowerPro capture folders as you use them in open/save dialogs; and you can combine these two approaches to have an integrated list of both manually set favorite folders and recently used folders.

You can display favorite folders in a menu or a bar (or both); see bottom of this help topic for more on bars.

MS Office does not use standard open/save dialogs. PowerPro can recognize these dialogs and send favorite folders to them for your favorite list, but it cannot capture the folder used from these dialogs.

If you are not using English Windows, you must set the letter beside "Folder" on advanced setup to the underlined letter in the title beside the file edit box on your open/save dialogs.

display favorite folders in a menu

To have PowerPro track folders as you select them in standard file/open save dialogs, check *"Combined Menu"* or *"Separate Menu"* (or both) on the configuration setup tab. For the combined menu, PowerPro creates a file called "c:\program files\powerpro\favfolder_any.txt" and places an entry in this file for each folder you access. For separate menus, PowerPro creates a separate file in the same folder named after the .exe file of the program with the open/save dialog; for example, for MS WordPad, the file is called "wordpad.txt" since the exe file name is wordpad.exe.

To view the resulting folders in a menu, assign the command *Menu Favfolder to a hot key or bar button and activate the command when the open/save dialog is open. A menu will be displayed of favorite folders; select one to send it to the dialog. If you have checked both combined and separate, the menu will have a column for combined recent folders and a separate column for favorite folders from the active program.

To manually add entries to the menu which will always appear, edit the file and add a line sep (for horizontal separator) or colsep (for new column) to the end of the file. Then list your folders on separate lines after this entry. You can edit either the "_any.txt" file (for all programs) or the "appname.txt" file (for a specific program) or both. You can precede file folder paths by myname= to have "myname" to appear in the menu to represent that folder path.

To use a menu of only manually entered files, make sure the *"Combined Menu"* and *"Separate Menu"* checkboxes are both unchecked on the setup tab. Edit the files to create your manual entries, ommiting the sep or colsep at the start.

for a more configurable menu

The *Menu Favfolder command is equivalent to the command *Keys {to folder}{filemenu favfolder_any.txt;*.txt} You can use variations of these commands for greater control of the menu layout and contents.

display favorite folders on a bar

To track folders as you use them for display on a bar, start by checking *"Text file"* on the *Setup* tab. This causes PowerPro to create a text file in the "c:\program files\powerpro\favfolder_anyshort" folder for each folder as you access it. The text file contains {to folder} followed by the name of the folder.

Then you need to create a bar to display these text files and execute the selected file using a Keys command. To do so:

1: Create a new command list and make sure you check *"Autoshow as bar"*.

2: Set the command list's *Properties* as follows: check *Tool tips*, set *Max text label* to 32, set *Icons* to none, check *Vertical bar* (not vertical text), check *Bar size from sum of buttons*, set *Position* to "Right of active".

3: In the *Properties >Folder Buttons* tab, set the *"Enter folder name..."* text box to "c:\program files\powerpro\favfolder_anyshort" (or change as appropriate if you installed PowerPro in another folder). Check *"Auto-refresh"* and *"Use Last Button as Command'*. Check *"Show filenames as..."* to have the file name shown as text beside the icon. In the *Menu Format* dialog, you can set the maximun number of characters shown. Check the *"Show Folder Name Only..."* checkbox for bars to show the folder name only, not the whole path.

4: Then create a single entry in the command list for the bar

```
*Format
Context
filedialog
```

This will cause a vertical bar, with a list of recent folders, to appear beside open/save dialogs when you use them; press a button to copy the folder to the dialog.

5: Finally, to create the Keys command, create a new item at the end of the list with the command

```
do ("keys", "{from "++_FILE_++"}")
```

which will send keys, from the file corresponding to the pressed button, to the Open/Save dialog.

In addition, you can add manually set folders to the bar, by creating command list entries after the *Format Context with the left command *Keys {to folder}c:\path.

If want a bar with only manually entries, uncheck *"Text file"* on the *Setup* tab or leave the *"Use this folder..."* edit box on *Properties > Folder Buttons* blank.

Of course, you can use other settings on Properties to get a different look for the bar.

Miscellaneous usage topics

Sending folder names to open/save dialogs

You can create menus or bars to send favorite folders to file open/save dialogs. PowerPro has built-in commands to make this easier for you.

If you want more control of the format of these bars or if you want to understand how these built-in commands work, then you will want to review the following information.

Use:
```
*Keys
{to folder}c:\path
```

to send the folder path c:\path to an open/file save dialog.

The {to folder} tells PowerPro to:

- automatically select the file edit box to receive the keys,
- save the contents of that box,
- send the keys to change to c:\path,
- and then restore the previous contents of the file edit box.

If you are not using English Windows, you must set the letter beside "Folder" on *>Setup >Advanced* to the underlined letter in the title beside the file edit box on your open/save dialogs.

To create a bar or menu of favorite folders, put a series of *Keys commands in a command list and show them as a bar or a menu with (e.g.) a hot key that shows a *Bar or *Menu. If you use a bar, you can make the bar appear only if a file open/save dialog is open by putting

```
*Format Context
filedialog
```

as the first command in the list, and positioning the bar in the caption (use a horizontal bar) or beside the active window (use a vertical bar). Don't forget to check *"Auto show as bar"*.

see also

You can ask PowerPro to enlarge all standard open/save dialogs by a setting in the GUI Control dialog.

Miscellaneous usage topics

Buttons from files in folders

creating bar buttons from the files and subfolders of a folder

You can add buttons to the end of a bar based on the files and folders in a subfolder. Left clicking on a button on such a bar runs the associated file or opens the associated subfolder with *Menu Folder. Right clicking on a button shows an explorer view of the folder (if a file is clicked) or the subfolder (if a subfolder is clicked).

To create folder buttons, create a command list and make sure you check "Auto show as bar". You can create ordinary buttons with the command list or you can leave the command list empty to have only buttons from the folder. Then use the *>Command Lists >Properties >Folder Buttons* dialog and set the name of the folder in the *"Enter folder name"* edit box. Use a wildcard file name to select the files to display (e.g. *.* or *.txt) or you can omit the wildcard file name, which is the same as specifying all files with *.*. When the bar is shown, all entries in this folder will be shown as buttons. You can use &() expressions in the folder name; the expression is re-evaluated each time the bar is closed and opened (or refreshed).

Check *"Show filenames as..."* to have the file name shown as text beside the icon. You can set the maximum number of characters in the label with the *Menu Format* dialog.

Check *"Auto refresh"* to have PowerPro automatically update the bar any time a file is added or removed from the folder. Or you can manually refresh the bar by ctrl-right clicking on the bar and selecting *Refresh Folder Buttons*.

Check *"Show hidden"* to show hidden files. Check *"Sort folders to start"* to put buttons representing folders at the start of the set of buttons. Check *"2- pane explorer window"* to show a 2-pane explorer window when a folder button is right clicked. Select sort method from drop down.

Use the *Menu Format* button to set the format of the menu displayed when you left click a button assigned to a subfolder.

Use the *"Show Folder Name Only..."* button for bars which are showing the favorite folder buttons to show the folder name only, not the whole path.

setting an alternative command

Normally, PowerPro executes the selected files as a command. But you can run another command with the selected file or folder as a parameter by checking "Use Last Button for folder button command." (Usually you make this a hidden button). PowerPro will execute the command with the selected file or folder as the command parameter.

Normally, the selected file is placed in double quotes after a space. If you want more control over where the selected file appears in the command line, you have two options: use the _file_ variable or use the character |

For _file_: the selected file is assigned to the variable _file_ which you can then use in a command, such as

```
do("notepad.exe", _file_)
```

which runs notepad on the selected file.

Alternatively, you can control the placement of the file by using a | to indicate the desired position:

```
*script assign x1 length "|" -1
```

would assign x1 the length of the file path less 1. Note that you must include the quotation marks if appropriate when using | (in those places where literal text must be in quotes). Use || to get the folder excluding the file name.

Note that with | and ||, PowerPro does not insert any spaces.

You can set the look of the buttons, whether tool tips are displayed, the size of the icons, and so on using the Properties of the command list.

You cannot use active buttons and folder buttons on the same bar.

Miscellaneous usage topics

Sharing PowerPro Configurations

sharing the entire configuration

You can share your PowerPro configuration with another user by sending the pproconf.pcf file to that user. The receiver can rename the file to, for example, "sharing.pcf" (for safety) and put the file in a new folder and then run the configuration in any one of these three ways: by ctrl-right clicking any bar and selecting New Configuration File; or by using *Exec ChangeConfiguration; or by starting PowerPro by running the command line:

```
"c:\program files\powerPro\powerpro.exe" "c:\path\to\shared\sharing.pcf"
```

assuming the shared .pcf configuration file is in folder c:\path\to\shared.

If the folder references .bmp, icon, or shortcut files, it is possible to include these in your shared information too. The best way is to put these files either in the same folder as your powerpro configuration or a subfolder of your powerpro configuration folder. When you reference the files in your configuration, use only the **relative path**; for example, a .bmp file named "back.bmp" stored in the \theme\ subfolder of your powerpro configuration folder would be referenced as "theme\back.bmp".

To share your pcf and related files, zip them together. The person you are sharing with then unzips the configuration files into a new folder and can then proceed as above to try the configuration.

sharing parts of a configuration

If you want to use command lists or hot keys from a shared configuration in your working configuration, use the *Export as text* feature from the *Setup* tab to export the command lists or hot keys of interest. Then import these into your configuration. You may also need to use some of the "default settings fo bars and menus" from the original configuration's >*Command List* >*Setup* dialog.

Miscellaneous usage topics

Changing Icons

how to change the icons for PowerPro Windows

You can change the icons PowerPro displays for its window, the debug window, notes, scripts (extension .powerpro), the vdesk arrange window, and the configuration program. To do this, you need to collect your set of 32x32 icons into an icon library called PowerPro.icl which you must place in the same folder as PowerPro.exe.

These icons must be in the following order:

0 - PowerPro.exe main icon

1 - Folder (if "Use Internal Folder" checked on Setup>Advanced>Other)

2 - Notes

3 - Debug

4 - ScriptFile

5 - Pproconf.exe

6 - Vdesk

7 - Input dialogs

The PowerPro Yahoo group files contains PowerPro.icl sets you can choose from.

Miscellaneous usage topics

Auto completion

Autocompletion can be used with the tiny run box shown by Exec CommandLine and with the inputdefault dialog used in expressions to provide automatic completion of input keystrokes.

There are three types of supported autocompletion:

- recent URLs, URLs in history, file/shell folder names and any combination of these three
- completion from the dropdown history of items for the run box or input edit
- completion using a set of strings stored in a vector

You use keywords to specify the choice of autocompletion. For ExecCommandLine, the keywords are put after the CommandLine keyword. For inputdefault, the keywords are placed in a text string or variable as an third argument.

keywords

file	autocomplete using file and shell folder names

urlhistory	autocomplete using history URL
urlrecent	autocomplete using URL most recently used
history	autocomplete using the history in the dropdown of the edit box
=varname	autocomplete using the strings in the vector in variable varname

You can use combinations of these keywords, except that you cannot combine urlhistory or urlrecent with dropdown history or =vv.

examples

```
Exec commandline urlhistory urlrecent
mycomplete=inputdefault("default","title", "=mystrings")
```

In the second example, the variable mystrings must be a vector, with each element as one of the strings to scan for auto completion.

You cannot use a local variable with a vector for Exec CommandLine (since this edit box stays open while the script continues to execute and possibly exit).

limitation

Only one dialog using auto complete can be active. For example, if you have a run command box open with auto complete, then auto complete will not function for inputdefault.

Miscellaneous usage topics

Displaying different subsets of a bar

There is a Demonstration of subbars

purpose

The subbars feature lets you display some buttons on a bar and hide others.

Subbars can be shown/hidden in one of three ways:

either [A] specify a virtual desktop name for each subbar

or [B] use the *Bar SelectSubBar command

or [C] use the ative window.

configuration

First, you edit the command list for the bar to put a *Format StartSubBar command at the start of each subbar and a *Format EndSubBar command at the end. These must be Left commands. Use the *Name* of the list item with the *Format StartSubBar command to set the name of the subbar.

to show for a virtual desktop

Give the subbar the same name as a virtual desktop. Check *Show subbar of same name as vdesk* on *Desktops >Setup*. The subbar will appear automatically whenever that desktop is current.

to show with a command

```
*Bar
SelectSubBar
```

BarName @SubBarName

will show the subbar called SubBarName on the bar called BarName. All other subbars of that bar will be hidden. Note that you have to specify the bar name, then an @ sign, then the subbar name. You can omit the bar name (but not the @) if the *Bar SelectSubBar command is on the same bar as the *Format SubBar.

There are many ways to configure the command:

- You could put the *Bar SelectSubBar command on another bar, or a hot key, etc

- Or use an always-shown button on the same bar as the subbar.

- One possible approach for a bar which contains several subbars: make a new button which is permanently shown (i.e. not within a subbar). That button could have the name: Info subbarname, which will indicate which subbar is currently being shown. Then attach a *Menu Show command to that button; the shown menu consists of a series of *Bar SelectSubbar commands.

- **Skins** often use the section/subbar approach to configuration. For a sample of such a bar, ctrl-right click on any bar, select "Change configuration" menu item, and then select "subbars" from the resulting submenu. If you have installed the sample skins, you can see how Skin Sample Kaos and Skin Sample Newbie display this pcf configuration. When you are finished with the demo, ctrl-right click on the bar and select "Change configuration" menu item, and then select "pproconf" from the resulting submenu to restart your configuration.

This configuration consists of a series of *Bar SelectSubbar commands to select subbars; each subbar is meant to consist of a series of command buttons belonging to some category named by the selection button. For more on this type of subbar usage, see Sections/subbars for skins.

- You can quickly create a subbar and a button for selecting that subbar from the command list configuration dialog by clicking Quick Add, or right clicking the list box and selecting Quick Add, and then selecting "Selector and new subbar" from the menu. The selector is added after any currently selected button in the list, and the subbar is added to the end of the list.

- A special version of this command, *Bar SelectSubbarToButton, can be used with bars that show other bars at the mouse when a button is clicked; see here for more details. You can omit @PartBar for SelectSubbarAtButton to show the whole bar.

to show for the active window

Make the subbar name on the Format StartSubbar command a captionlist which matches the window(s) you want the subbar to appear for.. Check *Show subbar matching active window* on *Command Lists >Properties > Bar Tab*. If there are buttons on the command list which are not part of any subbar, they will always be shown. You cannot use Format Context on bars which hav subbars determed by the active window.

notes

Buttons which are not within a subbar are always displayed.

You cannot nest subbars.

You can repeat the same name on different *FormatSubBar commands (i.e. the subbar does not have to be a contiguous set of buttons).

When a bar is first displayed, the first subbar in the command list is shown.

example

Suppose a bar called mybar is configured as follows:

Name	*Command*
Select	*Bar SelectSubBar mybar @edit

	Right: *Bar SelectSubBar @misc
edit	*Format StartSubBar
edit1	c:\windows\notepad.exe
edit2	c:\windows\wordpad.exe
EndSubBar	*Format EndSubBar
all	c:\windows\explorer.exe
misc	*Format StartSubBar
m1	c:\windows\calc.exe
m2	c:\windows\paint.exe
EndSubBar	*Format EndSubBar

Then left clicking the Select button would show the edit1 and edit2 buttons; right clicking Select shows m1 and m2 buttons. The All and Select buttons would always be shown. Note that the bar name is omitted on the right command for Select; this will work since the *Bar SelectSubBar command is on the same bar as the *Format SubBar.

Miscellaneous usage topics

Drag and drop onto a bar

You can left or right drag and drop a set of one or more file names from Windows Explorer or another File Manager onto a Powerpro bar, either to start a command with the file names as the parameters or to configure a new button.

Left drag and drop starts the left command with dropped file(s).

Right drag and drop file(s) onto the bar to activate a menu allowing you to select the command to receive the file or to be configured or to quickly add a new button using the dropped file/shortcut as the left command. For Windows 98 onwards, you can right drag/drop directly from the Start Menu.

You may want to drag and drop filenames into the middle of the command line. To do so, put the character "|" at the point where you want the dropped files to be placed. The "|" will be replaced by the dropped files when the command is run and the text following the "|" will follow the dropped files. Do not forget a space after the "|", if needed.

For non-active buttons, you have a choice as to whether the command is run once with all files as separate parameters, or whether the command is run once per file. Check *>Setup >Advanced >Other >"Drag drop runs command for each file"* to have the command run separately for each dropped file. In this case, the special variables Context and ContextLast are set to 0 at the start and ContextLast is set to 1 for the last file. Variable _file_ is set to the full path to the dropped file for each execution of the command.

If you choose to have all the command run once with all files, variable _file_ is set to the list of all dropped file paths, separated by a carriage return so that the lines function can be used to access them.

If the variable name _file_ appears anywhere in the command text of the accessed button, any "|" is ignored and the file paths are stored only in the variable.

You can drag and drop files to Active Buttons, and they will be passed to the executing program (if the program does not accept dropped files, you will hear an error beep).

Powerpro always attempts to start a new instance of a command when a file is dropped on a button.

Miscellaneous usage topics

Positioning the bar

setting the position

Set the bar position by Ctrl-right clicking the bar and selecting an entry from the Position submenu. Or choose a position from the dropdown in the command list's Properties dialog.

Floating: Bars with the Floating position can be moved by Ctrl-dragging.

Locked: Bars cannot be moved or resized.

Left/Middle/Right caption or Above/Below/Right of/Left of window: Bar is placed in the caption of or next to the active window. Size the bar before selecting this option by unchecking *Bar size from sum of buttons* and checking *3d sizing Frame* and then dragging the frame to size the bar. You can uncheck *3d Frame* after you have sized the bar, if you want.

You can move bars positioned in or beside the foreground window by dragging them. They will be repositioned when a new window becomes active.

Screen Edge: Bar reserves screen space like Windows Taskbar (unless it is autohide). You can select two types of screen edge bars: full screen and current size.

For current size, set the bar orientation and size first. PowerPro will automatically move the bar to the appropriate screen edge if *Move bar to edge for screen positions* is checked on *Command Lists >Setup >All Bars*. You cannot change the bar size once you select this position. Change the position back to floating if you want to change the size.

For full screen, PowerPro will move the bar to the selected edge and set the bar height or width to the full screen. You can change the size of the other dimension of the bar by dragging the bar border.

Task Bar: One PowerPro bar can be placed on the Windows Taskbar.

Fixed: Bar will stay in same position independent of screen resolution. It can be sized with sizing border if *Bar size from sum of buttons* is unchecked.

manual positioning

You can move a floating or active position bar by clicking and dragging any button. If you find that you are moving the bar when you do not want to, you can set use the *Command Lists >Setup >All Bars* dialog to require that Ctrl be down for dragging a bar to move it. You can return to the fixed position quickly by Ctrl-right clicking bar and selecting *Last Fixed*.

Miscellaneous usage topics

Screen edge positions

You can position a PowerPro bar at a screen edge and have it reserve a strip of screen space like the Windows Taskbar by selecting one of *Top, Bottom, Right,* or *Left Edge* (either *current size* or *full screen*). You do this on the bar's *Properties* dialog *Position* dropdown, or on the *Position* submenu of the configuration menu shown when the bar is Ctrl-right clicked.

If the bar is not an autohide bar, it will reserve screen space. In this case, the Windows system will automatically move desktop icons and windows out of the area reserved for the bar.

There are two types of screen edge bars: current size and full screen.

current size

set the bar orientation and size first. PowerPro will automatically move the bar to the appropriate screen edge if *Move bar to edge for screen positions* is checked on *Command Lists*

>*Setup*. You cannot change the bar size once you select this position. Change the position back to floating if you want to change the size.

PowerPro will try to set the reserved desktop space to be just large enough to accommodate the bar. But you can fine tune the size of the reserved space using the vertical (for top/bottom edge) or horizontal (for right/left edge) offsets on the Bar Properties dialog.

full screen

PowerPro will move the bar to the selected edge and set the bar height or width to the full screen. You can change the size of the other dimension of the bar by dragging the bar border.

Miscellaneous usage topics

Positioning a bar in the active window

positioning PowerPro bars in or beside the foreground window

settings

You can position Powerpro bars relative to the currently active window, by selecting one of these options in the *Position* dropdown in a bar's Properties:

in the foreground window's caption:
Left caption, Middle caption, Right caption.

or next to the foreground window:
Left of active, Right of active, Above active, Below active.

You can also fine tune the resulting position by specifying *Horizontal/vertical offsets* in the bar's Properties dialog.

options

• To avoid putting Powerpro bars in captions of or beside dialog boxes, uncheck *Include dialog windows for caption positions* in the *Defaults for All Bars* dialog, from the Setup button of the Command Lists tab.

• When there is no active window to use for the position, the bar will move to a default position on your desktop which you have previously determined by ctrl-left dragging the bar to that position. Or you can specify that the bar be hidden in this circumstance by checking *Hide caption position bars if no active windows* in the Defaults for All Bars.

bars for specific programs

You can have many bars with visibility dependent on the active window. Each can be positioned in the caption of (or beside) the window, but it will only be shown for the specified window(s).

Use *Format Context captionlist as the first Item in the bar's command list.

If you want to make the bar visible only when a certain program is active, your captionlist could use the =exename specifier.

Or, for a toolbar of frequently used folders, visible only for Open/Save dialogs, use *Format Context filedialog.

manual adjustment

You can move bars positioned in or beside the foreground window by dragging them if they are overlapping part of the window. These bars will be not be repositioned by PowerPro until a new window becomes active.

Miscellaneous usage topics

Taskbar Positions

Overview

There are three ways to access PowerPro commands from the Taskbar:

1. Create a folder of shortcuts configured to use PowerPro command lines with PowerPro commands and then create a standard Taskbar toolbar for this folder. However, this method does not provide proper foreground window processing for key sending and menu navigation and restricts you to the standard toolbar layout, and so will not be discussed further.
2. Create a command list and assign it the Taskbar position using the command list bar properties.
3. Install PowerPro TaskbarBar support from the Configuration Setup tab, create a command list called TaskbarBar, and then open the PowerPro bar from the Windows Task bar Toolbars menu accessed through the Taskbar setup menu (right click on task bar). Note that you may have to re-logon or re-boot after installing TaskbarBar support. To remove the bar label in the Taskbar, right click on a gripper in the Taskbar (after unlocking Taskbar, if needed) and uncheck "Show Title".

This section will review the pros and cons of the last two approaches and discuss the constraints on TaskbarBars.

Pros and Cons and Taskbar Position versus TaskbarBar

The Taskbar Position assigned to a command list works by creating a normal PowerPro bar which sits on top of the Windows Taskbar. PowerPro attempts to "force" the Taskbar to make room for the PowerPro bar. You can fine tune the position of the bar with the horizontal and vertical offsets on the bar tab of the properties dialog. You can changes the look of the bar using any approach, including skins. For XP, if you want to make the bar look like other toolbars in the Taskbar, check "Use Theme" on the Command List Properties Dialog, Bars and Menus tab (you may still need to manually set background color and text color).

The main advantage of the Taskbar Position approach is that any PowerPro configuration option can be used to control the look of the bar. The major disadvantage is that a non-standard technique is used to force the bar on top of the Taskbar: this may not be visually appealing and it may lead to instability of Explorer on some systems.

The other approach to putting PowerPro bars in the Taskbar is to create a command list called TaskbarBar, install TaskbarBar support from the Configuration Setup tab, and show the PowerPro bar using the standard Taskbar ToolBars menu. Such a bar uses the standard Windows technique to integrate into the Taskbar, and so will adopt the theme of the Taskbar (unless other colors or images are specified in the command list configuration) and will be completely stable. You use the Taskbar gripper to control the bar size. However, not all of the PowerPro bar setup options can be used, as detailed in the next section.

In summary, the Taskbar position provides the most flexibility but may not be visually appealing or may not be stable. The TaskbarBar approach is less flexible but completely integrated into the

Taskbar using standard Windows techniques.

Detailed Capabilities and Limitations of TaskbarBar Command List

The TaskbarBar command list allows many of the standard bar features. In particular

> You can use left, middle or right click to activate any PowerPro commands. Ctrl-right click activates the configuration dialog.
>
> You can use *info for buttons, and you can use button width, height, color, icons, and background settings.
>
> You can use the cl commands, as detailed below, to manipulate the TaskbarBar.
>
> You can use Subbars.
>
> You can set tooltips.
>
> You can set no icons, 16x16, or 32x32 icons, but not other sizes.
>
> You can use the width, height, hide icon, hide button, multi-line, and disable/no-3d settings for command list items.
>
> You can set maximum text size for the labels in the list. If you set maximum text to zero and check "applies to bars only", then only icons are shown, but any text labels will still be shown on the menu displayed by >> for buttons which are not shown if the bar size is not large enough.

The following features of the Bar Properties **cannot** be used (if a feature is not listed, it can be used):

> Active buttons and folder buttons
>
> Bar position and hiding-related features.
>
> Hover-clicks settings.
>
> *control to put controls in bars
>
> Transparency and gradient.
>
> *info on tooltips. You can change tooltip text with cl commands, if you wish.
>
> Bar size from sum of buttons, mouse through, button text positioning (except center can be used), shadow, no flicker, border
>
> Format item (but you can use cl commands).
>
> Format commands, except for format startsubbar, format endsubbar, and format separator.
>
> For format separator, you should set the item background color to the approximate color the of taskbar to get proper look for the separator.

Using cl services with TaskbarBars

You can use all of the cl services with Taskbar Bars. Normally, changes are applied immediately. If you have many changes and wish to apply them in a batch, use

cl.TaskbarBarUpdate(0)

to suspend automatic TaskbarBar updates. Use cl.refresh("TaskbarBar") to apply them manually. You can restart automatic updates with

cl.TaskbarBarUpdate(1)

You can reverse the current setting by using -1 as the parameter.

General syntax guidelines

Combining expressions, variables, and PowerPro commands

This section discusses how to combine expressions (described in the Expressions topic) with PowerPro commands like *Window. Use this approach for scripts in files or for entering commands into the GUI configuration dialogs.

Powerpro provides two alternative forms of syntax for its built in commands:

literal syntax

This is the original format, which was designed mainly for entering a command's parameters as **literal text**. Plain text does not usually need to be in quotes (except for filenames containing spaces). It is also possible (but sometimes problematic) to use expressions as the parameters, as described below.

Syntax: Command Action literal parameters

Example: Window Close calculator

expression syntax

This newer format is designed to make it easier to use **expressions** for a command's parameters. It is also easy to use plain text as a parameter, by placing it in double quotes. This is the same syntax as used for plugin calls.

Syntax: Command.Action(expr1, expr2, …, expr9)

Example: target = "calculator"

Window.Close(target)

Example: Window.Close("calculator")

note these differences in format

- in the Literal Syntax, the Command and Action are separated by a space; parameters (as text) are usually separated by spaces.
- in the Expression Syntax, the Command and Action are separated by a point; parameters (as expressions) are separated by commas and enclosed together in parentheses.

how to convert

Most built in commands can simply be translated between the two syntax forms as demonstrated above for Window. There are some special cases:

- The expression structure is not available for the Script or the Format commands.
- For Mouse, use win.mouse ("mouse cmds")
- For Keys, use win.sendkeys("keys")
- For Message, use the MessageBox("layout", "text", "title") function.
- For Wait, use wait for (expression)
- For File, Clip and Note, PowerPro automatically uses the plugin functions where available.
- For Window commands, when using the expression syntax, you can put an _ after the action name to avoid an error message if the window does not exist, e.g. window.close_("*notepad*) closes open notepad windows but does nothing otherwise.

using expressions in the Literal Syntax format

This is best avoided simply by using the Expression Syntax format. However if the older Literal Syntax format is needed for some reason, use the following methods:

If you try simply putting a variable in a command, you will find it does not work:

```
myFile = "c/file.txt"

"c:/program files\editor\editor.exe" myfile
```

will try to open the file "myfile", not "c:/file.txt". Similarly,

```
myCaption = "*explorer*"

window close myCaption
```

will try to close a window with caption matching "myCaption", not "*explorer*".

These examples do not work because commands in that form of syntax expect literal text in parameters. Therefore PowerPro does not recognize that you are using a variable or expression in the statement.

using &()

One approach to using expressions where plain text is expected, is to use the &(expr) syntax [if & is set as your "expression follows" character in *Setup >Advanced >Characters*]. However in some situations using &( ) can be problematic. See details of &() in the next topic.

Another approach, which avoids the problems of &( ), is to use a do( ) statement.

using do()

Every field in a do() statement is an expression. Even the command itself, in do()'s first field, is an expression; therefore you can use a variable to represent the command. If entering a literal command, it must be in quotes.

`do(filepath[, params[, workfolder[, howstart]]])`
executes an external command

`do(ppcmd.action(params and keywords))`
executes a built in command using Expression Syntax

`do("ppcmd", "action and params"[,"keywords"])`
executes a built in command using Literal Syntax

Note: Help uses [ ] to surround optional items. Do not include the [ ]s in actual use.

Do() statements can be used not only in script files but anywhere, including the dialog interface for entering commands.

To run files, including exe files for programs, one, two, three, or four arguments can be used. The first is the path to the file to be run, the second any command line parameters, the optional third is the work folder, and the optional fourth is one of "max", "min", "hidden", "traymin".

To run built in commands, you would normally not use do() but would instead use the Expression Syntax form "command.action(args)".

But you can use the Literal Syntax form in a do() statement by setting the first do() argument to the command, and the second to the action followed by any other command parameters.

Note: For Menu commands, the third "work folder" field of do() must be used for the keywords (from the second edit box of the GUI interface used to enter a Menu command).

Another reason for using do(): You can use the Do command to run commands stored as the value of string variables. The command in the variable could be either in the literal syntax or the expression syntax. This could be useful, for example, if you are reading commands from a file into a variable.

Note: You cannot use wait.until as the command in do() statements.

examples of built in commands

The following syntax examples are all acceptable, if not actually recommended:

These are both suitable if the argument is literal text:

```
Bar.Show("MyList")        ;; expression syntax
Bar Show MyList           ;; literal syntax
```

but if the argument is an expression [`var = "MyList"`]

```
Bar.Show(var)             ;; expression syntax
```

is preferable to:

```
Bar Show &(var)           ;; literal syntax
```

and this is unnecessarily complicated

```
do("bar","show" ++ var)   ;; using a do() statement
```

Note that the Expression Syntax is easier to use than a do() statement when you need to use an expression as an argument.

However when the command itself is in an expression, you must use do() :

```
mycommand = "Bar Show MyList"
do (mycommand)
```

Note: command words, action words, keywords, variable names, are not case sensitive.

examples of external commands

```
c:\\windows\\notepad c:\\docs\\file.txt
;; \\ is required if \ is set as the escape character
;; place each filepath in quotes if it contains a space

c:\windows\notepad.exe c:\docs\file.txt
;; if \ is not set as the escape character

c:/windows/notepad.exe c:/docs/file.txt
;; is always OK (unless / is the escape char !)

do("c:/windows/notepad.exe","c:/docs/file.txt")
;; is unnecessary because both items are literal
;; so this will be less complicated:
"c:/windows/notepad.exe" "c:/docs/file.txt"
;; where quotes are only needed around each item if it contains spaces

;; but if one or more items are expressions,
;; or if you wish to specify the starting folder, and how to start, use:
do(program,document,startfolder,how)
;; using quotes around all literal items
```

summary

The choice of alternative syntax forms may seem confusing initially. To make it simpler: in scripts it is best to use the newer Expression Syntax habitually for **all** PowerPro commands.

General syntax guidelines

Expression Follows character

The "expression follows" character is considered an obsolete way to combine Expressions and PowerPro commands. The recommended approach is to use the new Command.Action(args) syntax. See Combining expressions, variables, and PowerPro commands topic for details.

what it does

There is an alternative approach to combine expressions and commands which you will see in many sample scripts since it was the original way implemented to do this in PowerPro. This approach uses the "Expression Follows" character, which may be set in *>Setup >Advanced >Characters*. For example, suppose you set this character to &. Then PowerPro will do special processing each time it runs a command (or processes an expression). This is what it will do:

- Scan the line for &(expr) (note the expr could be a user variable or a built-in function).
- Evaluate the expression or variable in the parentheses
- Take the resulting text, and replace &(expr) by this text
- Repeat the above for any following &(expr) in the line
- Execute the line resulting from all the above substitutions

usage

So if you typed

```
myFile ="c:/a path/file.txt"
"c:/program files/editor/editor.exe" "&(myFile)"
```

PowerPro would replace &(myFile) by c:/file.txt which would create the following line to run:

```
"c:\program files\editor\editor.exe" "c:/a path/file.txt "
```

Since this is a command line, you need to make sure there are quotes around the file name to handle blanks in the file name.

limitations

One problem with &() is that, since it works by substituting the expression or variable into the command line, and command lines are limited to 530 characters, you cannot use long strings with &(). A second problem is that it is confusing to determine when to use &() and when just to use the variable name. You should not use &() in assignments statements, expressions, or plugin calls.

You may see scripts that contain statements like this:

```
var = "&(v2)abc"
```

instead of the recommended correct form:

```
var = v2 ++ "abc"
```

but using &() inside quotes inside of an assignment statement can cause problems.

One problem with "&(var)" is it does not work for long strings. Another problem occurs if the value of v2 contains a quotation mark, e.g. v2 with a value of a"quote. Then the &() assignment results in

```
var = "a"quote"
```

which causes an error message.

To avoid these problem, avoid &().

General syntax guidelines

Special characters used in PowerPro

wildcard characters

The wildcards ? and * are usually allowed in filenames.

In filenames and paths, ? means exactly one character, * means zero or more characters.

word? will match words but not word nor word processor

word* will match words or word or word processor

Also, * (but not ?) is sometimes acceptable in other parameters, such as in window titles, to represent zero or more characters. For example where a window's title (also known as its caption) is required as a command's parameter, Win.Close("*metapad") means the command should operate on any window whose title ends with metapad

bracketing characters

PowerPro mainly uses (round) parentheses in its syntax.

Exceptionally, the *Keys command uses {curly} braces to contain its special keywords.

Square brackets are used for indexing and regular expressions.

the escape character

You can optionally choose either \ (which is set in the recommended Standard Configuration) or choose any other character to be used as the escape character. Or you can choose to have no escape character set.

If you do set an escape character, the following pairs will have a special meaning if they are used within "quoted text".

Assuming you choose \ for the escape character:

\r a carriage return

\n a new line character

\\ is needed to represent a single literal \ in quoted text
(only if \ is set as your escape char)

comments marker

Used in file based scripts, and also in text configuration files, **comments** can be useful for your own notes; they are also useful to temporarily disable a line of code.

; (only allowed at the start of a line) This whole line is ignored when the script is run.

;; (allowed anywhere within a line, or at the start of a line) The rest of this line is ignored.

Also:

;;+ works as a line continuation mark in script files. This must appear as the last thing on a line. A line which terminates with ;;+ will be joined to the following line, then the combination will be executed as if it were a single line in the script.

expression follows character

Usage of the "expression follows" character, usually &(expression), is described in the previous topic.

customizing the special characters

You can change what PowerPro will interpret as these special characters by using the *>Setup >Advanced >Characters* dialog.

First, it will be necessary to uncheck *"Use standard configuration"* in the *>Setup >Advanced >Configuration* dialog.

General syntax guidelines

Caption lists

specifying target windows in a parameter

usage

Caption lists are used to specify one or more windows

- to specify the target window for sending keystrokes with *Keys
- window lists for *Window commands, the Win plugin, etc.
- for the context specifier of conditional menus, bars and subbars
- to specify the context window(s) for program-specific hot keys,
- and in many other places where you specify target or context window(s).

A caption list is a single string or a list of strings separated by commas. Each string in the list is used to try to match a window. (To include a literal comma in a string, put two commas in a row).

The caption list matches a window if any of the comma-separated items match that window, unless you use the ,&, separator as described below.

Note: all spaces are used when matching, so avoid unwanted spaces when using the comma separator.

items in a caption list

Each item in a caption list can use one of these window identifiers: the window caption, the window class, the program file, or the window handle , or you can use one of the special keywords listed below.

• to match a caption (a window's diplayed title)

yyy	to match a whole caption,
yyy*	to match captions starting with yyy,
*yyy	to match captions ending in yyy,
xxx*yyy	to match captions starting with xxx and ending with yyy,
yyy	to match captions containing yyy anywhere.

The caption-matching string cannot be a string of digits, since this would be taken as a window handle. Put asterisks at start and end to avoid this (e.g. *567843*, not 567483)

• to match a class

c=*PartOfClassName*

• to match a program file name

=exename (no path, no .exe) to match any window from program exename

+exename to match any window except dialogs from program exename.

Note: You can avoid matching configuration dialogs by using +exename or by putting ,nodialog at the end of the caption list. [note the comma]

- **to match the window handle**

  use the window handle number. (See next topic.)

- **to match using a special keyword for caption lists**

filedialog	to match file open/save/save as windows,
explorer	to match single or dual pane explorer windows,
explorer1	for single pane explorer windows,
explorer2	for dual-pane explorer windows.
under	to match the window under the mouse,
Taskbar	to match the Taskbar,
autorun	to match the last window selected by autorun,
activebar	to match the last window activated by an active button,
active	to match the foreground window.
rawactive	to match the foreground window, even if it is a PowerPro bar or Taskbar

- **to match window state**

  Put one of <min>, <max>, or <norm> at the start of an item in caption list to restrict match to windows in that state, eg

  <min>*notepad* - to match minimized notepad windows only.

- **to match tray icon window by id**

  Put (number) at end of caption list item to match tray icon window with that id number.

- **to match child window under mouse**

  Put k=string to match if the window class of the child window under the mouse is "string"

- **to match the and of several conditions**

  The comma separator of a caption list normally acts like an "or" so that it is sufficient for any item in the list to match. You can instead achieve an "and" of two items by separating them with ",&,"; for example,

  open,&,=notepad - to match open window from notepad exe

- **to match using a regular expression**

  Put r=regexp to match the regular expression to the caption. regexp is taken to be all the characters following the equal sign, including any commas. The caption and regexp are both translated to lower case before matching. Use x=regex to match the exename to a regular expression.

  r=.*special.* - to match windows with caption containing special

  You can also use re=regexp to match the caption in a case-sensitive match. In this case, the re= must be the only matcher in the caption list (r= can be combined with other match types using comma to separate).

example

notepad,=winword,explorer2,56,nodialog

selects any window containing "notepad" in its caption,

or any window belonging to winword.exe,

or dual pane explorer windows,

or the window with handle number 56.

Those program's config dialogs would not be matched.

mouse position

You can specify that the mouse must be at a screen corner, or screen edge, or quadrant with one of:

@topleft, @topright, @bottomleft, @bottomright, @top, @bottom, @left, @right, @quadtopleft, @quadtopright, @quadbottomleft, @quadbottomright

For example: @bottomright ensures that the mouse is at the bottom right corner.

using a captionlist to exclude window(s)

To select all but the specified windows or mouse positions, start the list with a tilde (~);

For example: ~*text.txt selects any window except those with captions ending in text.txt.

using an expression to match

You can use a normal expression to match by setting the whole match string to

=(expression)

You cannot use a comma separated list: =(expression) must be the whole matcher. You can use the variable x0 in the expression to get the handle of the window to be matched. For example

=(win.handlefrompoint(xmouse.ymouse).class()=="ToolBar32" && win.match (x0, "=notepad,&,*xxx*"))

checks the class of the child window under the mouse and also ensures the main window being matches (with handle in x0) matches the specified string

finding a visible window's identity

To find a window's **caption**, **class**, or **exefile**, for use in a captionlist, you can use PowerPro's WindowInfo feature.

Attach the *Exec WindowInfo command to a hotkey, or a bar button or a menu item. A small information window will be shown. You close it by executing the *Exec WindowInfo command again

As you move the mouse pointer around the screen, WindowInfo shows the following information about whichever window is currently under the mouse, on six lines:

1: mouse pointer coordinates,

both Absolute (point 0,0 is top-left of screen)

and Relative (point 0,0 is top-left of window under mouse)

2: window coordinates: (left,top) - (right, bottom) of window under mouse

3: total window size: width x height

4: client window size and aspect ratio

5: window caption (aka title)

6: window class and exe name

General syntax guidelines

Window handles

the unique window identifier

Each window is assigned a **unique number** by the Windows operating system. This number is called the window handle. PowerPro lets you retrieve window handles and use the *Window and other commands to access windows by their handles.

Window handles are useful when you want to access one out a series of windows which share the same exename or the same caption.

ways to retrieve window handles

window("firstwindow", captionlist)

returns handle of the first window found which matches caption list.

For example: window("firstwindow","active") returns the handle of the active window.

window ("visiblewindow",captionlist)

returns string of window handles of visible windows matching caption list. The window handles are separate by blanks. You can use the word function to extract them, as in

```
list = window (visiblewindow","=exename")
for (j=1; ;j=j+1)
handle = word(list,j)
if (handle == "")
break
; Process the handle
endfor
```

window("anywindow",captionlist)

returns string of window handles of visible and invisible windows matching caption list. The window handles are separate by blanks. You can use the word function to extract them, as above . The window function can return a maximum of about 60 handles; if you think you may need to process more (eg if captionlist is "*" for all windows), use the win plugin call win.handlelist instead.

window("vdesk",vdeskname)

returns string of window handles of windows on the named vdesk; you can use digit 1-9 or the vdesk's name string. Omits locked and system windows (e.g. Taskbar) and PowerPro bars. Use digit 0 as vdeskname to get handles of locked windows; locked windows are excluded from handles listed for a particular vdesk.

LastActiveHandle

function which returns the handle of the last window selected by an active button.

LastAutorunHandle

function which returns the handle of the last window matched by an autorun command list.

using a window's handle

Once you have a window handle, you can use it in any command or context which requires a caption list: in a *window command, in the target list of a hot key, in *format ContextIf, and so on.

For example: window.close(MyHandle)

would close the window whose handle is in the variable MyHandle.

General syntax guidelines

Clipboard contents as a command parameter

There are several different ways to use the contents of
the Windows clipboard in the parameter field of a command.

As an example, suppose the clipboard contains "c:\docs\mynotes.txt" which is the path to a file which we wish to edit in Notepad.

1

The clip function returns **the first line** of the clipboard

Examples: `c:\yourpath\notepad.exe &(clip)`

Or: `do("c:\yourpath\notepad.exe",clip)`

2

The clip.get plugin call returns **all the text** on the clipboard

Examples: `c:\yourpath\notepad.exe &(clip.get)`

Or: `do("c:\yourpath\notepad.exe",clip.get)`

3

You can use the option in >*Setup* >*Advanced* >*Characters* to set the clipboard character to #, where # can be any non-alphanumeric character. Then put the character # in the parameters edit box of the command entry controls.

Example: if the clipboard char is # and the following command is assigned to a button:

Command: `c:\yourpath\notepad.exe`

Parameter: `#`

then pressing the button launches Notepad to edit the file name contained as text on the clipboard.

Example:

Command: `c:\yourpath\netscape.exe`

Parameter: `#`

launches Netscape.exe to view the URL stored as text on the clipboard.

recommended

The clipboard character is considered obsolete; and the &() function can sometimes cause problems (eg with multi-line clipboard contents); so the best of the above options is:

`do("c:\yourpath\notepad.exe",clip.get)`

Configuring with the GUI dialogs : About the dialogs

Configuring PowerPro using dialogs

purpose

You can configure Powerpro command lists, hot keys, and scheduled events in several ways:

For major/detailed changes:

- using the main configuration dialogs of pproconf.exe described below
- importing command lists defined in .ini text files
- using CL functions for whole command lists and for items

For minor/quick changes:

- using CL functions for command list items
- using the *Format Item commands
- using the menu shown when you Ctrl-Right click a bar

using the dialogs

This topic introduces a group of topics about using the main configuration dialogs, which are shown by PowerPro's separate configuration program, pproconf.exe in your powerpro folder.

Pproconf's GUI interface lets you configure Powerpro bar buttons, menu contents, media, hot keys, and alarms with the Configure Powerpro set of tabbed dialogs.

See Starting pproconf for various ways to start pproconf.exe

Pproconf displays a set of tabbed dialogs as follows. These tabs also correspond to the actions you can select with the *Configure command to control which tab is initially displayed. For example *Configure Setup starts pproconf at the main Setup tab.

Setup

The Setup tab accesses various PowerPro functions, plus some of the ways that Powerpro can customise your Windows interface.
• Its *Advanced setup* button shows another multitabbed dialog for more detailed Powerpro settings. Here, the *Configuration* tab has settings about the configuration dialogs themselves. For example you can have pproconf make its tabbed configuration dialog be always-on-top.

GUI Control

This tab sets various functions which customize your Windows interface.

Command Lists

Allows you to change the properties of each command list; and to change a command list's individual Items.
• On this tab, the *Setup* button lets you set the default properties for all bars and menus.

Key/Mouse

Allows you to assign commands to hot keys, mouse actions, or screen corners.
• Here the *Setup* button leads to the default settings for all hotkeys.

Scheduler

Allows you to add or change alarms and to schedule commands either to be run once only or to be run regularly such as every month, every weekday at a certain time, etc.
• This tab's *Setup* button has the defaults for all scheduled items.

Timers

Controls the value and commands of timers.

Media

Controls the sounds, wallpaper, and screen saver; allows you to specify how Powerpro should automatically change them.

Desktop

Specifies the initial name, programs, and wallpaper for virtual desktops.
• This tab's *Setup* button configures virtual desktops generally.

run reconfigure on save

You can run a script called Reconfigure each time the configuration dialog is closed with OK by checking "Run Reconfigure ..." on the Configuration tab of the Advanced Setup dialog.

reminder

In addition to its actions for starting the config dialogs at a particulat tab, the *Configure command also lets you add a new scheduled message by using its NewReminderMessage action *Configure NewReminderMessage.

restart

Use the command Configure Restart to restart PowerPro from the .pcf configuration file. This may help if other programs are interfering with PowerPro operation. Or try Configure Rehook, which just resets PowerPro's keyboard and mouse hooks.

Configuring with the GUI dialogs : About the dialogs

Starting pproconf

There are many ways to start the configuration dialog. Choose one that is convenient to you.

- Start the program pproconf.exe from a shortcut, or using Explorer, or as a PowerPro command.
- Ctrl-right click bar and select menu entry *Configure*.
- Ctrl-right click bar and select *add button* or *delete button*.
- Right drag-drop a file, a desktop shortcut, or a Start Menu item (Win98+) to a bar and select an option from the resulting menu.
- Left or Right or Middle Click on a bar button and hold down the mouse until the configuration dialog appears. The choice of mouse button you hold down selects the appropriate configuration dialog.
- Drag and drop a file, desktop shortcut, or Start Menu entry (win98+) into the Command Lists configuration dialog.
- To configure the command list shown by *Menu Show on a button, alt-click the button

- Create a bar button, or a menu item, named "Configure", with the *Configure command. You can specify which tab of the configuration dialog will be shown initially.

note

The configuration program, pproconf.exe, runs independently of powerpro.exe which executes your configuration.

Configuring with the GUI dialogs : About the dialogs

Command entry controls

purpose

Powerpro uses a standard set of configuration controls to enter commands to be run by a button, menu item, hot key, timer, scheduler, and so on.

A command is a file or program you want to launch; or it is a built-in command used to manipulate windows or running programs or to change your Windows configuration. Built-in commands can start with an asterisk (*), although you can omit the asterisk if you want. Also, some plugin calls act as commands.

configuration

There are several ways to enter a command into the **Command** edit box:

- type or Paste the command into the edit box,
- select a built in command from the drop down,
- press the program files button to select a command from the start menu,
- press the file browse button to browse for a file,
- use the wizard button to select a command using wizard menus,
- or use the paste button to paste a previously copied command from Powerpro's own "command clipboard" (does not use the standard Windows clipboard).

If you enter a file or program: you can use the **Parameters** edit box to enter command parameters. If you select a built-in * command, PowerPro will change the remaining dialog controls to suit the chosen command. The label of the Help button will also change, indicating which topic in PowerPro Help will be shown if *Help* is pressed.

A command of (none) is ignored. A command starting with a semi colon (;) is ignored. A command starting with two semi-colons is ignored; in addition, any remaining more commands are ignored.

Use the **Copy** and **Paste** buttons to copy and paste commands and their parameters between different sets of command entry controls. This uses pproconf's own "command clipboard" and will not change the contents of the Windows clipboard

Use the **Apply** button, if present, to save the current configuration for testing. Applied configurations will be removed if you use the **Cancel** button. (When you press Apply, the configuration is also saved in the file pproconf.apply and you can return to that configuration with the *"Restore previous backups..."* button on the *Setup* tab.). Use the **Test** button to have PowerPro run the command after applying the current configuration.

For file commands, you can set the starting **window position** (normal, minimized, maximized, tray minimized, hidden), the **topmost** status, and whether or not PowerPro **switches to** an existing window of the same program if it is already running. You can also enter the initial **working directory** for the program.

You can use variables and expressions in built-in commands and file executions.

You can use the More Commands edit control to enter multiple commands.

To play a sound each time a command is run, enter the .wav file name for the sound in the More Commands edit box. You'll also need to check *"Play .wav files"* on the Advanced Setup dialog to have PowerPro play the sound (rather than the associated program for .wav).

Configuring with the GUI dialogs : Setup

"Setup" dialog

purpose

The Setup dialog allows you to set some Windows interface features: move the cursor to the default button of new dialogs and optionally press the button; automatically hide new windows when they are opened; automatically tray-minimize certain applications when they are minimized; track folders used in Open/Save dialogs; track explorer windows as you open them; and to force explorer windows to a given view.

The lower three rows of buttons are for some special PowerPro configuration features, such as importing command lists from text files, activating Explorer context menu support, or activating support for some advanced TrayIcon features. The "Show All Bars" button shows hidden bars too, which can be useful while configuring them.

From this main Setup tab, you can also access the Advanced setup dialog, which even includes a page for configuring how the configuring dialogs operate.

dialog details

Cursor to button...
Check to have Powerpro automatically move the mouse cursor to the default button of any dialog.

...except
This optional list of exceptions uses the captionlist method of specifying windows.

Press default button
For the windows listed here, the default button will be pressed automatically. *Cursor to button* must be checked.

Automatic Hide
Enter a caption list of windows to be hidden when they are created.

Automatic Tray Min
Enter a caption list of windows of to be minimized to the tray instead of the task bar. Also check *"Traymin if program starts minimized"* to have PowerPro tray min windows which match the caption list and which start minimized or which are minimized when PowerPro starts.

Save up to n folders...
Using PowerPro's features for open/save dialogs.

Explorer Windows
You can specify that Powerpro track Explorer windows for use with the *Menu Explorer command. Use the Advanced dialog to set size of history.

Force...
You can specify that Powerpro should force settings for Explorer view and arrangement.

Install Tray Support
Required for working with tray icons from other programs, in bars or menus.

Install TaskbarBar
Installs support for TaskbarBar command list to be shown as Taskbar toolbar.

Install Context Menu
Enables adding items to Explorer's right click menus

Find/Replace text

Shows a dialog allowing you to search for text throughout the configuration file and replace it by other text. For example, if the folder path to many of your commands changes because you moved some files, you can use this dialog to search and replace the path throughout commands.

For more complex editing of your configuration, use import/export buttons to send configuration to a text file, then edit this text file, then re-import the text.

Import or Export as text

Plain text configuation files are described in Configuring with text files. Note: to export text files of command lists with More Commands, you must use the carriage return as command separator.

Restore Backup/Restore Previous

Unless you indicate otherwise on the advanced setup dialog, PowerPro keeps five generations of backup for your configuration file. A copy of the current configuration is kept in "!auto backup of ...", and a copies of the 5 previous configurations are kept is "!Previous auto backup of ...". You can restore either of these backups using buttons on the Setup dialog. This button can also be used to restore the result of the most recent press of the Apply button.

Configuring with the GUI dialogs : Setup

"Advanced setup" dialog

On the main Setup tab, you can press the *Advanced setup* button to access a dialog which lets you set many less-used Powerpro options. There are four tabs: *Configuration* lets you change options controlling the configuration dialogs, *Other* controls miscellaneous features, *Characters* let you define or change special characters used in commands, and *Limits* set timer and count limits.

"Configuration" tab

Use standard configuration

This first checkbox lets you specify that you follow the standard configuration. That makes it easier for you to share scripts which are written using this standard.

The standard configuration option, when checked, sets the following other options and disables changing them:

On the Configuration tab: *Use multi-line more commands edit* is forced to yes.

On the Characters tab:

Expression in () follows forced to &

Quote must precede expression follows character forced off

Use \ for escape forced on

Process the "expression follows character" in assign forced on

Start and end characters for Keys forced to { }

Command separator, *prompt character*, and *clipboard character* are each forced to null

Include desktop icons

If checked, desktop icons will be included on the Start Menu shown by pressing the Capture button on command configuration controls.

No Auto backup

Check to prevent PowerPro making automatic backups of configuration files.

Run reconfigure

If checked, any command list called Reconfigure will be run after the configuration dialog is closed with OK. The could be used to restart *waits or to reset command list items changed with cl.xxx functions.

Config on top
Check to display pproconf's configuration dialogs as topmost

Remember column widths
Check to have PowerPro store column widths set in configuration dialog lists.

Icons on configuration
Check to display item icons on command lists in the configuration dialog. (This will slow display of these lists.). Gray check to display icons only for command lists marked "Show as Bar".

Location of mouse click selects
Use to control which tab is displayed initially when you double click command list item. Check to display tab of clicked item (left or middle or right). Gray check to only display left/middle/right for auto show as bar command list. Uncheck to always display left.

No asterisk in drop down
Removes asterisk from drop down of built-in commands on command entry controls.

Use multi-line command edit
Allows multiple lines to be entered in the *More Commands* field of command entry controls.

Put debug information at end of list
Check to have PowerPro enter information in sequence at end of debug window list; uncheck for reverse sequence at top..

"Other" tab

Show all windows
If checked, all windows for a task are shown whenever any window for that task is activated.

Show tray minned
Shows windows which PowerPro has tray minimized if they are activated, e.g. by launching a document which the window program is associated with.

Play .wav files
Check to have PowerPro play .wav files used as commands; uncheck to use standard associated program for .wav (e.g. Media Player).

Stop Alt-F4
Prevent alt-F4 from closing bars.

Use timer for active
Check to have active bars refreshed every 2 seconds. Gray check for refresh every 1 second. Only needed if active bars are non-responsive due to interference of another program.

Fast Send Key
Uses a faster engine to send keys but does not work for some very rare ctrl-key combinations (e.g. ctrl-tab). Note: This default setting can be over-ruled in any individual *Keys command by using the {fast} or {slow} keywords.

Allow only alphas
Allows only alphanumerics in file names formed from captured clipboard items

Use internal folder icon

Check to use the folder icon in PowerPro.exe file; uncheck uses the system folder icon (which may fail in NT).

Position for input dialogs

Select starting position for dialogs displayed by input and inputdialog functions.

Restore desktop

PowerPro restores saved desktop icon positions when the screen resolution is changed; however, this option may cause Explorer aborts on some systems.

No prompt for restart

If checked, PowerPro will never prompt for a restart after execution of *Exec Resolution.

Drag drop many files

If checked, when files are dropped from explorer onto a button, the button's command will be run separately for each file dropped, with variable Context set to 0 at the start and ContextLast set to 1 and variable _file_ set to the file for each execution of the command.

Shortcut evaluation of &&, ||

If checked, then && and || will only be evaluated as far as needed to determine result (for example, in 0 && .myfunc, myfunc will not be called).

Variables must be declared

If checked, PowerPro will produce an error message if a variable is used before being declared in a Global, Local, or Static statement; this helps avoid misspelled variable names.

Replace %env%

If checked, PowerPro will replace any environment variables found in command line or directory work area (must be %environment var%).

Delete unused environment vars

If checked, PowerPro deletes what it believes are unused environment vars during an environment var refresh; some users have reported problems with this option, however.

"Characters" tab

Start and end char

Specify characters to replace { } for specifying special keys when sending keystrokes.

Expression character

Use this character to insert expressions or variables such as &(name) *or* &(x0+var1) To remove special meaning in a command line, precede by a single quote (') or the escape character, if specified.

Quote must precede

Affects how the escape character is used with expression char and prompt char. If checked, then it must precede either of these for the script insertion or prompting to occur. If unchecked, then a preceding escape character stops the script insertion and prompting.

Process in assign, if, for

Check to have expression character recognized and processed in assign, if, and for. This is advanced usage; normally, you do not use the expression character in these statements. If unchecked, then any usage of the character is replaced by blank.

Escape character

Use this character for escape in quoted strings

Reverse results

Use at the start of a captionlist to make it an exclusioin list. For example in hot key targets to select all windows which do not match the following strings. ~ is the default for this.

File name character

Used with open/save file tracking, this is the underlined character in the title beside the file name edit box. You may need to change this for non-English languages.

Command separator

Obsolete; the recommendation is to this field leave blank and instead use multi-line More Commands, as selected by that option on the Configuration tab. You can use this character to separate multiple commands; leave blank for none. You can include the command separator character in a command (without it acting as a separator) by preceding it with a single quotation mark (').

Prompt character

Obsolete. Leave blank and instead use do ("xx", input ("title")) to prompt for input.

Clipboard character

Obsolete. Leave blank and instead use the clip function to insert the first line of the clipboard; or the clip.get plugin to insert the entire clipboard.

"Limits" tab

Max entries per column

Sets maximum number of rows in *Menu Folder and *Menu Explorer

Milliseconds button held down

Button configuration dialog will be shown after the button is held down for this number of milliseconds; set to a large number to disable.

Explorer Windows

Sets maximum number of entries in *Menu Explorer; you may need to restart PowerPro if you change this value.

Scroll interval

Sets time in milliseconds between scroll steps when automatic mouse scroll is activated.

***Menu Folder interval**

Sets time in milliseconds before tool tip appears for *Menu Folder and *Clip menu. Set 0 to disable tool tip.

Send keys delay

Set delay in milliseconds for first key and subsequent keys

Marker Window

Sets the number of pixels used in the marker for hidden bars.

Time mouse hovers

Sets the hover time in milliseconds for cases when bars are set to activate a left click after mouse hovers over button.

Hook disable

Set non-zero to disable internal hooks. Only needed if debugging configurations.

Max wait before aborting

Set to number of milliseconds PowerPro waits on active windows to return information regarding their window text or icon. Smaller values (less than 500) mean PowerPro is less likely to be frozen by non-responsive windows, although for very small values it may abort too soon and temporarily miss the information. However, if the window fails to respond in time, PowerPro will continue to monitor the non-responsive window until the window is closed or becomes responsive.

"Notes" tab

For information on the Notes tab, see *Note command.

Configuring with the GUI dialogs : Setup

Automatically moving the mouse cursor to a dialog button

Check the "Cursor to default button" checkbox on the Setup dialog to have Powerpro automatically move the mouse cursor to default button on dialogs.

You can omit certain dialogs by including their captions in a caption list in the edit box beside the checkbox. You need not enter the whole caption: enter xxx* for captions starting with xxx , enter *yyy for captions ending in yyy and enter *zzz* for captions containing zzz anywhere.

You can have Powerpro automatically push the default button by including a caption list of windows in the "Press default button" edit box at the bottom of the dialog. Powerpro will wait for 1 second before pressing the button by default; you can change this wait time with the internal PressDelay option.

If you gray check the checkbox, Powerpro only moves mouse cursor and presses the default button for captions specified in the "Press default button" edit box

Configuring with the GUI dialogs : Setup

Automatically hiding windows

You can specify that Powerpro should automatically hide any windows, should they become visible.

Put the comma-separated captions of the windows you want to autohide in the Auto Hide edit box on the main Setup dialog.

Configuring with the GUI dialogs : GUI Control

"GUI Control" dialog

The *GUI Control* dialog configures many options related to the way you interact with Windows.

configuration

There are check boxes for controlling Caps Lock, Num Lock, and Scroll Lock keys.

You can indicate that windows should be centered when switched-to from the active window list or the active task buttons.

You can indicate that Powerpro should show more of windows which it activates and which are mainly off the screen.

You can specify that Powerpro should disable the screen saver while a RAS connection is active.

You can specify that if a scheduled *ScreenSaver or media tab command changes the saver while it is running, then the running saver should be changed to the new one.

You can indicate that Powerpro should show window size and position whenever any window is moved or sized.

You can have PowerPro force newly created windows to be completely on screen.

You can specify that Powerpro should enlarge the file list windows used in file open and save dialogs (only works for programs that use standard Windows dialogs).

You can set the maximum width of Taskbar buttons; for example, setting this to 22 produces a button consisting solely of the icon. This does not work on Windows XP or later.

You can use the middle mouse button and mouse movement to scroll windows.

You can indicate that Powerpro should pan (move) windows into view when the mouse is held over them at the screen edge; you can set the speed of panning by setting the step size in pixels.

You can ask Powerpro to press buttons, select combo box items, etc., if the mouse is stopped over the button for a specified time.

You can indicate that Powerpro should activate windows when the mouse passes over them and set a delay in milliseconds for how long the mouse has to be over the window for it to be activated. You can further specify that the active window should only be changed if the mouse is over a caption.

You can specify that Powerpro automatically track text pasted to the clipboard.

You can specify that holding the left, right or middle mouse button down and turning the scroll wheel activates the alt-tab dialog to switch tasks, except for a caption list of windows.

You can specify that dragging the right mouse over a window will move or size or do both to that window. If the mouse is in the center, the window is moved. If it is in a corner, both with width and height of the window are resized. If it is near an edge but not a corner, only the width or height are changed. While the window is being moved or resized, a tool tip is optionally shown giving the mouse position (Absolute screen and Relative to window), the window size, the window client area size, the aspect ratio of client to full size, and the window caption and exename, Note that the minimum drag amount needed to activate the size or move is given by Configure > Keys/Mouse > Setup > "Drag Length".

Configuring with the GUI dialogs : GUI Control

Clip filters

You can have Powerpro place captured clipboard items in subfolders of the clip folder by entering filter strings in the filter edit box on the *GUI Control* dialog.

Filter strings take this form

String=subfolder

where String is xxx*, *xxx, or *xxx* to match xxx at start, end, or middle of clipped item; subfolder is the name of the subfolder of the clip folder in your Powerpro directory where you want to put any item matching the String.

For example

```
*.zip="zip files"
```

puts any captured item ending in .zip into the subfolder "zip files". Note that you must put the subfolder name in double quotes if it contains blanks.

You can separate multiple matching strings by commas:

```
*.gif,*.jpg,*.jpeg=Pictures
```

puts strings ending in .jpg, .jpeg, or .gif into the pictures folder. Avoid blanks in the matching String.

The strings in the clip filter edit box on the GUI Control dialog are processed in sequence:

try*=tryfiles *.zip="zip files"

would put any strings starting with try in tryfiles and then any other strings ending in .zip in "zip files". Note that you use a space to separate multiple filters.

If the captured clipboard item is longer than 500 characters, only the first and last 250 characters are used when checking filter strings.

You can control whether or not items which match a filter are also placed in the active list with a checkbox on the GUI Control dialog.

Configuring with the GUI dialogs : GUI Control

Clipboard history tracking

To enable automatic tracking of information as you paste it to the clipboard, use the drop down box beside "Track" on the GUI Control dialog to select the type of clipboard information you want tracked. Select whether you would like plain text, text plus rtf, or all types of information tracked. When this is done, Powerpro will automatically track the most recent items pasted to the clipboard.

Use the *Clip MenuPaste command to display a menu of recent clips that have been tracked; selecting one puts the clip on the clipboard and pastes it to the current program. If you set a non-zero value for *milliseconds mouse hovers over *Menu Folder* on the Advanced dialog, PowerPro will show the first few lines of the stored clip as a tool tip for the *Clip Menu display. For non-text items, PowerPro will show a picture of the item in a tool tip as you hover the menu item.

If you are tracking all types of information, you can choose to paste only the text information in a file by holding down the Ctrl key when selecting the file from a menu or when using the PowerPro Clip File command.

You can also ask Powerpro to automatically copy selected items to subfolders of your clip folder with filter strings; see below for details.

You can prefix the clip name with a time stamp hhmmss by checking this option on the GUI control tab; this will ensure that clips starting with the same text have a different clip name and do not overwrite each other.

You can play a sound file each time an item is captured by setting the *PowerPro Clip* sound on the Media tab.

You can run a command list called "ClipCaptured" by checking the *Run ClipCaptured* checkbox.

You can set maximum clip size for captured clips; larger clips are ignored.

Some programs, like some versions of PhotoShop, interfere with PowerPro clipboard tracking. You can use the *Reattach* check box to have PowerPro check every five seconds and if necessary re-establish itself at the front of the list of programs tracking the clipboard. Or you can manually reattach PowerPro to the clipboard with the command *Clip Reattach.

Clipboard tracking may produce occasional aborts with some programs. You may be able to correct this by setting the delay value on the GUI Control dialog. If the delay is greater than 0, PowerPro will wait for (delay value)* 333 milliseconds before capture the clip. This may prevent aborts. Note that if you capture more than one item during the delay, only the last item is processed by PowerPro, so use a small a delay as possible.

further information

Clipboard plain text is actually stored in a .txt file in the clip subfolder of your main Powerpro folder. You can edit it with your standard editor. You can access that editor from the *Clip menu by right-clicking the menu item. Clipped rich format text is stored in .clprtf files which only Powerpro can read. If you track all types of information, the clipboard is stored in a

.PowerProClip file. In this case, the file name will start with (Image) for bitmaps, (EMF) for Windows Enhanced Metafiles, and (Files) for lists of files copy from Explorer. (EMF files also generate an extra .PowerProClipEMF which PowerPro manages). Image and EMF files have do not contain text for PowerPro to use in the file name, so PowerPro always appends the date and time.

You can double click on .PowerProClip files and PowerPro will load the file contents to the clipboard.

Using Explorer, you can create subfolders of the clip folder and use these subfolders to permanently store text snippets you want to access. Create the snippets by copying them from the main clip folder, by using clip filters, or by entering them directly by saving files from Notepad or any other editor which can save plain .txt files. You can then access these snippets from the *Clip menu.

You can access only the clips in one folder xxx with

Command *Clip

Parameter menu xxx

You can show only the automatically tracked text items with:

Command *Clip

Parameter menu active

clip menu layout

The *Clip menu command is actually implemented by a *Menu Folder similar to the following:

Command *Menu Folder

Parameter c:\program files\PowerPro\clip

Format noext noicons sorttime folderstart folderdot cmd "*Clip filepaste"

If you would like a different display of the clip menu, create your own *Menu Folder command using the above as a model. Note the cmd field which runs a *Clip filepaste on the selected item.

Configuring with the GUI dialogs : GUI Control

Caps Lock, Scroll Lock, Num Lock

The "GUI Control" dialog contains check boxes to permit you to control the behavior of the Num Lock, Scroll Lock, and Caps Lock keys.

You can specify that pressing Shift always clears Caps Lock, to avoid reversed mixed case like pOWERpRO.

By setting the *Shift clears Cap Lock* check box to the gray-checked state, you specify that Shift should clear Caps Lock only when a letter is pressed with Shift.

Or you can also completely disable the Caps Lock key.

You can disable the Scroll Lock key. This key is rarely used, and when activated unknowingly, causes irritating behavior from the Arrow and other keys.

You can disable the Num Lock key. Check the box to set the key permanently off. Gray check to set the key permanently on.

Note: control of the Num Lock key does not work on some systems.

Configuring with the GUI dialogs : Command Lists

"Command Lists" dialog

purpose 1 - bars and menus

The command lists dialog is mainly used to create lists of Items for:

display as a menu. See *Menu command

display as a bar. See Bars and the *Bar command

(special purpose command lists, for advanced use, are described below)

There are two steps involved with making menus and bars: first use the *Command Lists* dialog to create the list of commands; then use a command like *Menu show or *Bar show to display the command list as a menu or a bar. Thus one particular command list could be used for either a bar or a menu depending on the command which shows it.

configuration

Select the command list you want to work with from the drop down at the top of the dialog, or to create a new one press New list.

Press Properties to control how the command list will be displayed as a menu or bar. Press Setup to set the defaults for all bars and menus.

To configure one item in the command list:

Select an item in the list and use the controls at the right of the dialog to change it. You can also right click on an item to access a popup menu of configuration options, or you can double click on the item to change it. When you edit an item, Powerpro takes note of which column (left, right, middle) you clicked on, and starts the command editing dialog with this column selected.

You can edit item labels in place by clicking on them twice slowly enough to avoid double clicking. You can see the complete contents of a field which does not fit in its column by holding the mouse over the field which causes a tool tip to appear; you may have to click on the field to update the tool tip. You can also resize the configuration dialog by dragging any border to see more of the fields in the entries in the command list.

There are several ways to add new items:

- By using Add Before and Add After to add an item and configure all commands and features. See Configuring a command list item for details on entering command items.
- By dragging a file or shortcut from Explorer or a desktop icon and dropping it onto the command list dialog. to set the left command, icon, and label from the dropped item. Or, for Win98 onwards, you can also drag directly from the Start Menu. The item is added after the current selection.
- By using Quick Add to add a new item after a selected item and set the left command to a file or a Start Menu entry. Only the left command and a subset of command features can be set with Quick Add.

You can select multiple items for drag/drop move/copy and for copy/move to other lists, but other configuration functions require a single selection only.

Press Apply to immediately test changes using Powerpro. If you cancel the configuration dialog, applied changes will be removed. When you press Apply, the configuration is also saved in the file "pproconf.apply" and you can return to this configuration with the *"Restore previous backups..."* button on the Setup tab.

You can also add items to a bar's command list by right drag/dropping a file, desktop icon, or start menu entry (Win98 and later) to a bar and selecting from the menu which results.

If you want Powerpro to automatically display the command list as a bar, check the *Auto show as Bar* box at the top of the command list. You can also use the *Bar command to display bars.

advanced configuration methods

As an alternative to GUI configuration described above, you can use the cl functions to configure command lists from scripts; or you can edit command lists stored in a special kind of text file which can be imported and exported using buttons on the main Setup dialog.

purpose 2 - special command lists

the items in a command list can be run consecutively as a script

a list can define a set of keyboard macros

(the next five are set in Command Lists >Setup >Special Lists)

Tray: displaying tray icons

Open: listing the commands to run when specified windows first open

Active: to set non standard icons or colors for active buttons

Icons: to set custom icons for tray minimised windows

the "Monitor" list is run once per second (or two seconds) -- if checked (or gray checked)

the "Reconfigure" list is run each time the configuration dialog is closed with "OK" -- if "Run reconfigure" is checked in Advanced Setup >Configuration

the "PProShutdown" list is run when PowerPro is closed – if a list with that name exists.

the "Context" and "ContextFolder" lists are used for items which PowerPro will add to Explorer's right click context menus

the "ClipCaptured" list is run as a script whenever a clip is captured – if "Run ClipCaptured" is checked in the GUI Control dialog

if they exist, lists called "HookSystemEvents" or "HookWindowEvents" will be run as a script whenever a system or window event occurs

if it exists, "HookMenuFolder" is run as a script for each file selected from a Menu Folder

if it exists, "HookCommandLine" is run as a script just before running the command line in the Tiny Type and Run box

if it exists, "OnScriptDebug" is run as a script just before displaying debug dialog

if it exists and Exec OnErrors("hook") is run, "HookErrors" is run as a script whenever an error occurs

if a command list's name is the name of a virtual desktop – if "Rerun command list with desktop name each time desktop is activated" is checked in Desktops >Setup

Configuring with the GUI dialogs : Command Lists

"Tool tip setup" dialog

set the tool tips' font, colors, timing and sliding animation

To adjust the tooltip settings for all PowerPro bars and menus:

Either: click *Tool tip setup* on *Command Lists >Setup >All Bars and Menus.*

Or: select any command list from the dropdown list on the Command Lists tab, then click *Properties >Bars and Menus >Tool tip setup*. Although this second method accesses the dialog from a command list's Properties dialog, it accesses the same dialog and sets **all** of PowerPro's tooltips.

With this dialog, you can set:

- Background and text colors and the font for tool tips.
- Whether slide animation is used to show the tool tip (Win98/2000 only).
- Whether PowerPro will draw tool tips for its tray icons; this allows multi-line tool tips.
- Whether or not PowerPro will stop showing tool tips after you click a button (and not star showing them again until the mouse is moved off the bar and back).
- The delay until tool tips appear for bars.
- A character used to create multi-line tool tips. Whenever this character is found in tool tip text, a new line is started. For example, you could create one line per command for buttons or menu items which have left, middle, and right commands.
- The delay until tool tips appear for *Menu Folder and *Clip menus (zero means no tool tip).

Configuring with the GUI dialogs : Command Lists

Bar and Menu "Setup" dialog

setting defaults for all bars and menus

purpose

This dialog controls the appearance of command lists displayed as bars or menus. The settings here are defaults which will be applied to all command lists – unless this default is contradicted by a setting in an individual command list's Properties. It is also used to specify the command lists used for tray icon, auto run, active button icon, and tray minimized icons.

configuration

The Defaults dialog is selected by clicking on the *Setup* button from the *Command Lists* dialog. A tabbed dialog will appear allowing you to select options for All Menus, All Bars, both All Bars and All Menus, and for command lists being used for special purposes such as tray icons.

"All Menus" tab

Cache Menu Icons:
check to store icons for menus in separate memory managed by PowerPro. This will allow menus to be displayed more quickly but will require more memory. Uncheck to use the Windows operating system icon cache. Gray for caching large icons.

Force cursor over newly opened menus:
if checked, Powerpro will force the mouse cursor over newly opened menus; this is especially useful if you autohide menus after the cursor is moved off them.

Right selects command in menus:
if checked, selecting a menu item with the right button will execute the corresponding Right command from the command list. (You may also use middle click to select the middle command, but this feature does not work on all systems and currently is not guaranteed to be reliable). You must make sure *"Display Menu on mouse down"* is not gray-checked to use this feature.

Track new windows for *Menu RecentCommands:
you must check this to have PowerPro maintain the recent commands menu.

Default no icons

Check to specify no icons on newly configured command list properties.

Hide *Menu Folder

Will automatically hide *Menu Folder and *Clip menus when mouse moved off of the menu.

Icons on *Window and *Menu folder menus:

check to include icons on these menus.

"All Bars" tab

Less 3D effect:

Less pronounced outlines are used when drawing 3D borders on buttons. Gray check to make button height two pixels smaller by eliminating the extra space used to draw this border.

Allow off screen bars:

If checked, PowerPro will not try to make bars partly visible if off screen when they are first shown.

Move Bars to edge:

If checked, PowerPro automatically moves screen edge bar positions to the appropriate screen edge.

Stop Alt-F4:

If checked, press Alt-F4 when a Bar is the foreground window will not close the bar.

Use timer:

If checked, a timer will be used to ensure refresh of active bars and caption-positioned bars. Normally, should be left unchecked.

Show Bar selectsubbar as pressed:

If checked, the button with a Bar SelectSubbar corresponding to visible subbar is shown as pressed.

Hide caption bars if no active windows:

check to hide bars positioned in window captions when there is no active window.

Show tray iconized windows on active buttons:

check to show windows which PowerPro has tray minimized, on active buttons. Gray check to show all hidden windows.

Windows to show on active buttons:

Determines whether or not to show an active button for a top-level window. Select all standard (all unowned), .or one per exe (which limits active buttons to one to represent any windows from a single program). Note that 1 per exe does not apply to tray icons shown on active buttons.

Sort active bar buttons

Select standard sorting (like Taskbar) or sorting in order of window creation (for this windows that are hidden and then shown will re-appear at original position).

Dynamically update text/icon/flash on active buttons:

check to have text andi cons dynamically updated on active buttons. You must restart PowerPro after changing this option. This option may cause the Taskbar to display blank buttons on some older Windows systems.

Flash active buttons:

Check this option (and the above) to have PowerPro flash active buttons with the Taskbar button flashes.

Cache active icons:

Check this option (and the above) to have PowerPro keep a private copy of window-specific icons; this may improve responsiveness.

Use class icons for active buttons:

Check to have Powerpro use the window class icon, even if a window-specific icon is available. Uncheck to have PowerPro ask the window for its icon; in this case, set the time limit to wait on Setup|Advanced|limits. Checking this option to use the class icon leads to stabler behavior but may mean that PowerPro uses an icon different from the Taskbar for some programs.

Display menu on mouse down on bar button:

if checked, *Menu Show commands are executed as soon as you click down on a mouse button with this command. Gray check to have the mouse up select the menu item. Gray checking is not compatible with checking "Right selects commands in menus option" option (see above).

Include dialog windows for caption position:

check to include dialog windows when positioning bars in captions.

Drag to move:

If checked, left click and drag any button to move a bar; if unchecked, you must ctrl-left click and drag to move a bar.

Use slide animation:

Bars are shown and hidden using a slide animation (Win98 or later).

Marker window:

A marker window can optionally be displayed for a bar when it is hidden. You can set the marker size in pixels.

Time mouse hovers

You can set the time the mouse hovers over a button for the button to be clicked, assuming *"Hover Clicks"* is checked on the command list *Properties* of the bar.

"All Bars and Menus" tab

You can set the default **background color**, **text color**, and **font** for bars and menus.

Omit these strings:

Insert a list of strings separated by commas; these will be removed from the window caption displayed in *Window menus and active buttons.

List of captions of hidden windows

Insert a caption list for hidden windows to appear on active buttons.

Default background

If set, this background image will be used for all bars and menus. You can override for an individual bar or menu by setting the background for that bar or menu to none.

"Special Lists" tab

Run Monitor:

If checked, PowerPro will run the command list called Monitor once per second (gray check for once every two seconds). This can be used to perform background processing, e.g. updating the format of a bar item. Make sure to debug your command list by manually *Script Running it before activating the repeated running. You can also used the command *Exec Monitor reverse to set or clear the repeated running of the Monitor command list.

Use PowerPro tool tips:

If checked, PowerPro will draw tool tips for its tray icons; this allows multi-line tool tips.

Using the drop down boxes to choose command lists' names, you can specify that the commands on those special lists should be used for:

Tray: to display tray icons

Open: to provide a command list to control commands to be run when a specified window first opens

Active: to provide a command list to control the icons assigned to tray minimized windows

Icons: to provide a command list to control the icons assigned to active buttons

Configuration Bar Positions

Locking Bar Positions to Other Bar Positions

purpose

You can cause a bar position to depend on the location of another bar using the Lock To button from the Bars tab of Command List properties (or by using the cl.LockTo function). Moving the target bar will automatically move the locked bar to the same relative position. If the target bar becomes invisible, the locked bar is hidden; the locked bar is reshown when the target bar is shown.

configuring relative position

Each bar can be locked to up to one other bar. Bars to be locked to other bars must have the floating or locked setting specified in the Position drop down box of their configuration.

To lock the bar position, you specify three things using the dialog displayed by the Lock To button:

1. The name of the target bar to be locked to (when configuring, you can specify the name of a bar which does not have a command list in the pcf file, and the locking will still happen when that bar is created using the pcf file later or using cl functions)
2. The point relative to the target bar that the locked bar is to be targeted at.
3. The point relative to the locked bar to be set at the target bar position.

Note that two points are specified: one relative to the target bar and one relative to the locked bar. PowerPro moves the locked bar so that these two points coincide.

The fastest way to set the relative positions is to use the "Select from Preset..." button which displays a menu of common alignments and sets the locked points based on the menu selection.

For finer control, set the locked points manually.

Points are specified using four numbers: Horizontal Percentage (of current bar width), Horizontal Offset, Vertical Percentage (of current bar height), Vertical Offset. To understand how these work, first consider just the percentage settings, and assume offsets are zero.

Suppose you want the locked bar to be located just to the right of the target bar. This means that the locked bar's left should be aligned with the target bar's right, and the locked bar's top should be aligned with the target bar's top. So the settings would be

Target Horizontal Percentage – 100

Target Vertical Percentage – 0

Locked Horizontal Percentage – 0

Locked Vertical Percentage – 0.

As another example, if in the above you wanted the bottoms to align, then the two vertical percentages would be 100. If you wanted the bars to be vertically centred (but still end-to-end horizontally), then the two vertical percentages would be 50. Finally, if you wanted the locked bar to be horizontally centred under the target bar, then the settings would be

Target Horizontal Percentage – 50

Target Vertical Percentage – 100

Locked Horizontal Percentage – 50

Locked Vertical Percentage – 0

Note that at least one of the percentages will normally be zero to avoid overlapping the bars. (But you can overlap bars if you want.)

The offsets are used to make fine adjustments to the above. Usually you need set the offset only on either the target or locked bar, whichever is convenient. For example, if in the first example above you wanted the locked bar to be 1 pixel to the right of the target bar, you would set the target bar horizontal offset to 1.

configuring visibility

PowerPro automatically arranges for the locked bar to have the same visibility as the target bar: the locked bar will appear and disappear with the target bar. You can use the ""Slide Direction for Show" setting for the locked bar to specify how it will appear and disappear. However, you should leave the "Hide After" setting for the locked bar at "No Hide" and the "Show If Bump" at "none". This ensures that the locked bar's visibility depends solely on the target bar.

If a bar is locked to a target bar, and the target has a "Hide After" setting, PowerPro will ensure that the target bar is not hidden as long as the mouse cursor is over the target bar or any bar locked to that target bar.

locking multiple bars

You can lock multiple bars to the same target bar or you can create chains of locked bars. For example, if bars B and C are both locked to A, then both B and C will maintain their relative position to A and will be visible if and only if A is visible. You can also create a chain like C locked to B locked to A. In this case, C moves whenever B moves, and B moves whenver A moves (and this will move C as well).

You can create circular chains. For example, if A is locked to B and B is locked to A, then moving or hiding one will do the same to the other. In this case, you have to give some thought to setting the relative positions so that the position of A relative to B is the same as B relative to A.

Configuring with the GUI dialogs : Command Lists

Tray icon buttons

adding your own icons to the tray

purpose

You can display the items in a special Command List dialog as icons in Tray on the task bar. Left, middle, or right clicking on a PowerPro tray icon activates the corresponding command.

You can also use this feature to replace the text of the clock in the system tray by any dynamically varying text using an *Info label.

configuration

First, configure a command list to contain the items you want as tray icons. If you want to replace the system clock by text, configure one of the icons with a text *Info label. Then, select that command list on the Command List Setup dialog beside the tray drop down.

You can include tool tips for any tray icon. These tool tips can contain dynamically varying text using a *Info labels; note however that PowerPro limits tray icon tool tips to at most 63 characters.

The icon displayed in the tray is the one chosen for the item. If no icon is available, the Powerpro icon is used.

The width of the text item replacing the clock is set from the width field. If the width is zero, then the length of the initial value of the *Info field is used. If the width is negative then the width is given by the initial width of the *Info field plus the absolute value of the width (i.e. the width provides a buffer beyond the initial field size).

Tray icon text replacing the clock is normally forced to one line. But you can have multiple lines by setting positive values for both the height and width of the item associated with the *info field which replaces the clock. In this case PowerPro will word wrap the text in a rectangle of the specified size.

note

These PowerPro tray icons do not require PowerPro's Tray Support to be installed.

See also PowerPro's features for controlling the tray icons of other programs (which do require Tray Support).

Configuring with the GUI dialogs : Command Lists

Command list "Properties" dialog

using the dialog

purpose

The Command List Properties dialog sets the properties of one command list.

Some of these properties will only be relevant when the list is shown as a bar. Some apply when it is shown as a bar or as a menu. If it is run as a script, all of the appearance settings will be ignored.

Some of the properties set here, such as text color, may overrule settings in the defaults for all bars and menus and may in turn be overruled by a setting for an individual button.

configuration

To access the dialog, select the command list to be configured in the drop down at the top of the Command List tab and press the Properties button. You can use the tabbed dialog to access command properties which apply whether the command list is a displayed as a bar or a menu, properties for bars only, and properties for active buttons and folder buttons on a bar. Press the Apply button at any time to preview the effect of any formatting command on currently visible bars.

"Bars and Menus" tab

You can change the command list **Name**, whether or not **tool tips** appear, and what size **icons**, if any, appear on the bar or menu when displayed .

You can set a **background color**, a **text color**, and a **font** for this command list. You can set separate background and text color for buttons, buttons with mouse hovering over them, and pressed buttons. Check gradient to have vertical gradient on buttons (gray check to reverse direction).

For XP and later, you can also specify that these settings be taken from the current theme (also called visual style) which has been assigned to Windows standard toolbars. In this case, you must also ensure that the file powerpro.exe.manifest exists in the same folder as the PowerPro exe. Remove the ".unused" from the file which is provided with the PowerPro install. (Do the same thing with the PProcconf version to have visual styles apply to PowerPro Configuration program.) You may still need to manually set background or text colors.

You can set the maximum number of characters of text to be displayed as the **label** (set to 500 to display all text specified in the item label). You can also access **tool tip** setup which applies to both bars and menus.

For situations where you are displaying a command list as a bar and a menu, you can use a checkbox to force 16x16 icons for the menu display, you can use a checkbox to indicate that menu text should be taken from the tool tip (and so be different from the bar text), and you can use a checkbox to have the maximum text limit apply to bars only.

For both menus and bars, you can set an image file to be tiled for a **background image**. Use * to have PowerPro set the background bitmap to the wallpaper under the bar or menu. For menus with background images, you can set the **horizontal offset** in the Bar section to indent the text and icons on the main menu to show a bit map pattern in the left (you may have to set the position to e.g. caption to enable the offset field for entry). (Note; if you experience aborts with some .bmp files: this is a Windows bug; as a workaround, put an = in front of your .bmp file which should fix the problem although multi-column menus may not show the background as well).

The **hide after** feature shown under Bar properties applies to both bars and menus and can be used to hide a bar or menu after the mouse cursor is off of it for the specified time.

"Bars" tab

Except for hide after, these apply only when the command list is displayed as a bar.

Position sets the bar position.

Lock To displays a dialog allowing bar to be locked to another (Position must be float or locked).

Height sets the button height in pixels. A zero height uses the default, which is based on the icon height and text size, but you may have to manually set height for vertical or rotated text. This height setting is ignored for menus. It will apply to bar buttons whose height is set to 0.

Hide after sets whether PowerPro should hide the bar or menu -- see visibility. Hide after applies to both bars and menus, but the Bump and Marker settings only apply to a hiding bar.

Bump You can specify a screen edge to be bumped to show the bar and the amount of time the mouse must be held at the edge to show the bar; this time is the same for all bars. You can also specify that the bump must be within the boundaries of the bar at the screen edge.

If you prefer a different approach to showing bars, you can also show bars by defining a hotkey or bar or menu command to execute a *Bar show command. Note that "showing" bars applies to two cases: hidden bars and non-topmost bars (which are obscured by other windows).

Marker creates a small marker strip when bars are hidden.

Slide direction You can choose how to use slide animation when hiding and showing the bar. You must also check "Use slide animation" on Command lists >Setup.

Offsets specify offsets for positioning in or near the active window, Taskbar, and screen edge positions.

border check to draw a thin black border around the bar; not applicable to translucent, pixel-blended bars.

3d/Sizing check to show white border at top of bar and dark border at bottom to give 3d look to bar; use to size and shape bar if "bar size to sum of buttons" unchecked and position is floating, fixed, or full screen edge. For translucent, pixel-blending bars, no frame is drawn, but the borders of bars can still be used to size. Gray check to include sizing frame but with same color as bar background.

no flicker Eliminates bar flicker (but uses more memory to draw bar).

vertical bar buttons are aligned beneath one another. Only a single column is supported: for multiple columns, use one of the following: vertical bars together; or use Format NewBarRows at end of rows; or use 3D frame to drag border to desired bar shape.

hover clicks if mouse hovers over button, that button is left clicked; set the hover time on Command lists >Setup >All Bars

round bar select degree of roundness for corners of bar and which corners to round

round button select degree of roundness for corners of buttons

flat check for flat bar; gray check to avoid button border when mouse passes over. If grey checked and a press color is set on "Bars and Menus" tab, no 3d effect is shown on press down as well.

same size check to force all buttons to width of first button

all desks sets bar to be displayed on all virtual desktops; unchecked displays on the desktop when bar is first shown

topmost check to display bar always on top

bar size if checked, bar is automatically resized to accommodate all buttons; if unchecked, use 3D sizing frame to manually size and shape bar (floating or fixed position only).

subbar from active if checked, only subbars with name matching the active window caption will be shown

right icon icons are shown on the right of text

shadow show a shadow under bar (XP with shadows enabled only)

pixel blend: Requires transparency greater than 0. Used to determine how to blend images into underlying windows. If unchecked, images are blended solely using transparency setting. If checked, images are pixel-blended. If gray-checked, button images are pixel-blended into bar images and result blended to underlying windows.

Mouse click through: If checked and transparency is greater than zero, then mouse clicks pass through bar.

gradient bar color is varied the specified number of steps with specified background color in middle of color range

vertical Check to have vertical gradient; uncheck for horizontal.

transparent (Win2000 and later only) set 0 for standard opaque bar. Set 1-254 for growing level of translucence. Set to special value 255 to have background color of bar only made transparent.

vertical text text is shown one letter per line running down button; set button height >0 too

rotate text text is rotated and runs up the button; set button height >0 too. Not all fonts can be rotated, you may have to also set font to (e.g.) Arial using "Own Font"

text under check to display text under the icon on bar buttons

center text centers text label

notes

You can also ctrl+right click the bar to get the configuration menu and set some of those format options from the Look submenu.

You can force new rows on non-vertical bars with the *Format NewBarRow command. You can start a new row and show a horizontal separator line with *Format NewBarRowLine. Finally, you can insert a vertical separator line with *Format BarVerticalLine.

If you gray-check *"Hover clicks"*, only *Menu commands are activated by hovering over a button. After displaying a menu, if you move the mouse to a different button where left clicking shows a menu too, then the first menu is closed and the second menu is opened.

To change a bar size manually, make sure the *Position* is Floating or Fixed, *Sizing Border* is checked (look configuration menu accessed by ctrl-right click), and *"Bar size to sum of buttons"* is unchecked: then left-drag the bar border.

"Active Buttons" tab

Here you set the number of active buttons and tray icon buttons, and whether they should display icons only. You can also specify that the last button in the command list should be used to set the middle and right commands and the colors for active buttons and you can specify whether the foreground window should be shown pressed.

You can specify a list of captions to control which windows/tray icons appear as active buttons. If you want this list to apply to tray icons only, put a # at the start. You must install tray icon support before tray icons buttons will work.

"Folder Buttons" tab

Specify a folder whose entries will appear as buttons at end of bar. You cannot show folder buttons on a bar which shows Active buttons.

Configuring with the GUI dialogs : Command Lists

Working with hidden bars

controlling bar visibility

The Command List Properties dialog contains options for hiding a Powerpro bar. (Note: these options **always** hide the bar; if you want to show the bar only if a certain program is active or a certain window is visible, use the *Format Context command).

Set the time *Hide After* to a value greater than zero to enable hiding. When the mouse cursor is moved off the bar for this number of milliseconds, the bar will be automatically hidden.

You can also set the amount of time the mouse must be held at the edge to show the bar (this time is the same for all bars).

To show the bar, choose a screen edge from the drop down on the Properties dialog. Bumping (moving the mouse to this screen edge and holding the mouse at the edge for the time specified) will show hidden bars. If you want the screen bump to be limited to only showing the bar if the edge bumped is within the bar boundaries, check *Bump must be within bar size*.

For Win98 onwards, you can choose how to use slide animation when hiding and showing the bar. You must also check *Use slide animation* on *Command Lists >Setup*.

If you want a small marker window to show at the screen edge of a hidden bar, check *Show Marker Window*.

You can also show a bar by assigning the *Bar Show, *Bar ToMouse, or *Bar SelectSubBar commands to a hotkey, another bar button, or a menu item and then executing this command.

To avoid showing the bar if a full screen program or a DirectX program is running, check *Disable bump if full screen program running*.

You can also show all hidden bars from the Setup tab of the configuration dialog.

If you cannot make your bar visible, run the PowerPro Configure program using one of these methods and reset the bar properties to avoid hiding.

Configuring with the GUI dialogs : Command Lists

Configuring a command list item

To open the "Edit list Item" dialog: select an item in the command list then click the Edit button; or double click an item in the list; or click the Add before or Add after button.

Name: To set the item label, type a label of up to 127 characters into the label combo box or select a dynamic display (date/time/resource) label from the ***Info...*** button. Leave the label edit control blank to omit a label (e.g. if you just want to show an icon). For command lists displayed as menus, you can optionally precede any letter in the menu item Name by an & to use that letter as a menu mnemonic to select that item using the keyboard. (Use && for a plain &). You can display controls in buttons using *control.

Id: Use to enter id for item to be used in the cl functions for single items and for skins.

Tool tip: Enter tool tip text. You can avoid a tool tip display for this item by setting the value to none. You can create multi-line tool tips with the separating character specified on the tool tip setup (default is slash (/)). You can display dynamic information on tool tips using *Info. The multi-line character still applies within the information displayed by *Info.

Image: Choose the source for the icon from the drop down if you want the icon to be based on the Left, Middle, or Right command. For a menu where other entries have icons, you can set this to (none) to avoid an icon for this item. Or browse for an icon file with the ... button. Use the spin box to choose a specific icon from a file. You can choose a .bmp file for icon in which case the whole button face will be tiled with the bitmap in the file.

Background, text color, and font: Check the box and use the *Set* button. You can also change these items with a cl function command. For item colors and font to work on menus, the menu must be showing icons or have its own background color or its own default font.

Hide: check to hide this item when the bar or menu or tray icon is displayed (does not apply to *Format commands). Also de-activates items in a Macro list or in the special command list of commands to be run when a specific window first opens.

Hide icon: Check to hide icon. Can be set programmatically with cl function.

Disable/No 3D: If command list is shown as a menu, disables this item. If shown as a bar, eliminates 3d effect when mouse cursor is over button.

Multiline: A bar or button height and a button width must be specified. Text will be wrapped onto multiple lines within this button rectangle. Carriage returns will be processed.

Hover clicks: If *"Hover Clicks"* is checked on command list *Properties*, this drop down is used to select which bar button command is activated by hovering; you can also select *none* to disable hover this for button.

Width: Leave 0 for default width, which is just wide enough to accommodate the text and icon. Set to a positive number to specify a fixed width in pixels. Set to a negative number to specify that the width should be the default width plus the specified width (negative sign removed). Width is only used for buttons, not menu items.

Height: Leave 0 for the default height or for the height set on command list *Properties*; else set button height in pixels.

Commands: Items can have three associated commands: use the left/middle/right buttons to access each set of command entry controls. Use the start menu or find button or the command dropdown list to set a program or enter a built-in command. You can also drag/drop files or desktop icons to the dialog to set the command.

Configuring with the GUI dialogs : Command Lists

Running multiple commands

There are two ways to run multiple commands:

- Put all the commands on a command list (or in a script file) and use one of the script running commands. Using a script is the most powerful way to specify multiple commands and works for any number of commands, which you can store in either a command list or a file.
- Or configure the commands to run from a single Item in a command list by using the More Commands edit box. This is convenient for a small number of commands but there is limited space.

You can specify three or four commands in the More Commands edit box as the second and subsequent commands. Start each command on a new line. Generally, lines in the More Commands box are written as they are in script files. You can either enter the command directly or use the binoculars button for a dialog, which will write the line correctly for you. However, you cannot use `for` in More Commands, and you can only use at most one `if` *or* `if-else` command.

If you specify a file name with blanks as a command, you must put it in double quotations.

You can set the work directory, or the formatting codes for *Menu, by preceding with the character pair !`.

```
*Menu Folder c:/path/test !`cmd "*Script Run ="
```

You can try to determine how the a program is shown by ending the line with

`*hide`	to hide the window
`*min`	to minimize the window
`*max`	to maximize the window
`*traymin`	to tray minimize the window

(these options may not work with all programs)

examples

```
c:/windows/notepad.exe edme.txt !`d:/mydir *min
```

starts notepad minimized, editing file edme.txt, with starting folder d:\mydir.

Command: `c:\windows\Notepad`
More Commands: `*wait 1`

*keys hello

starts notepad, waits for 1 second, then sends the keys "hello".

Command: c:\windows\write

More Commands c:\sounds\hello.wav

starts Textpad and plays the specified .wav file; check *"Play .wav files"* on Advanced dialog to avoid any media player windows.

tip

The dialog which fills in one line (called one "statement") in the More Commands box for you can be helpful in learning how to enter statements in script files. When writing a script file, if you need to enter a command with complicated parameters: you can do it in this dialog by editing any command list item, then paste the resulting line into the script file, then cancel editing that command list item.

Configuring with the GUI dialogs : Command Lists

Active window switching with buttons

a more customisable alternative to the Taskbar

purpose

If you want to configure your own version of the task bar, you can create Powerpro buttons which automatically track each top-level window on your system so you can quickly switch to, close, or minimize any active window.

You can also put the icons from the Taskbar tray onto an active bar by installing tray support first from Setup > Configure.

You can combine active window and tray icons on one bar or have separate bars for each. You can use filter to select which active windows to track and have different active bars with different filters.

You can specify the colors and the commands for middle and right clicking an active button, by configuring a special, hidden button at the end of the bar's command list and specifying, in the bar's Properties, that active buttons should use the last button for setup.

When you drag a file over an active button and wait momentarily, the corresponding window is shown. If you drop the file on the button, PowerPro will attempt to send the file to the corresponding window, but this only works for some programs (those which accept files dropped on their window captions).

configuration

You set up active task buttons with the *Active Buttons* tab Command List's *Properties* dialog. They are also affected by some settings in the "Defaults for All Bars and Menus" which is shown from the Setup button in the Command Lists dialog.

in the "Active Buttons" tab:

To display active buttons, set the maximum number of active buttons to a number greater than zero. Check "Show buttons for active window" or "Show buttons for tray icons" or both. Powerpro will display a button for each active task or tray icon.

You can have the active buttons shown after some normal buttons, or you can have a bar of active buttons only (in which case the command list for the bar is empty).

To display text on active buttons, you must set a button width to accommodate text by configuring the final (or only) command list item to have the desired button width in pixels and by checking *Last item for Setup* on the active button tab. With enough width set, both button and text will be displayed on an active button. To display the icon only, check *Active buttons show icons only*.

You can control the look of active buttons and the commands for all active buttons by checking *Last item is used for setup*. In this case, include an extra item in the command list at the end. This last item in the command list is not displayed as a button. Instead its text and background colors and its commands are assigned to each active button. If you assign a *Windows command to this last button, use the activebar target window to have the *Window action apply to the window selected by the active button.

Normally, the bar size is fixed and the buttons grow and shrink. Instead, you can cause the bar itself to set its size according to the number of active buttons by checking *"Bar size set to sum of buttons"* on Bar Properties or the Look submenu shown by ctrl-right clicking the bar. Set the button width on the *"Last item for setup"* button; otherwise the button width will be set to the width of an icon.

To indicate which window is in the foreground (active), check *Show button corresponding to active window as pressed*.

To display only minimized windows as active buttons, check *Show only minimised...*

To select which windows appear as active buttons, enter captions in the last edit box. If you leave this blank, active buttons will be shown for al windows.

in the "Defaults for All Bars" tab:

To ensure that active button text is dynamically updated, check *"Dynamically update active button text"*.

You can control whether hidden and tray minimized windows are displayed on active buttons with a check box on the Menu and Bar Setup dialog. You can also use this dialog to select specific hidden windows to be displayed by including them in a caption list.

You can use the omit list to cause any active window to be excluded from the active task buttons or to edit the name of text for the active task buttons. Or, you can use the *Exec Window built-in command to hide the window.

You can determine how many active buttons to show for a program: one button for each top level window, one button for each unowned top-level window (standard), or one window per program. In this last case, a menu of windows is shown if you click on a button representing a program with more than one top level window.

When displaying the icon and colors for a window on a button, Powerpro normally uses the window class icon. You can specify specific icons and colors for programs by creating a special command list and selecting this command list with the icon menu drop down in the *Special Lists* tab of the Command List Setup dialog. Create one entry in the list for each program with an icon or color that you want to specify. Set the list item name to a caption list like =exename. If you want to change the icon, set the item icon to the icon you wish to use for all windows matching the caption list. If you want to change the text or button color, check the corresponding "Own" checkbox and set the color. You can restrict the icon or color to windows in a specific state by using <min>, <max>, or <norm> at the start of the caption list item. When scanning the command list, PowerPro will use the first item which matches the active window, so you can have multiple entries for a window in different states. If there is a match and colors are specified, they override colors specified by *Last item is used for setup.*

You can cause activated windows to be centered using a switch on the GUI Control dialog. Gray check to center the mouse cursor as well.

Some programs interfere with the hooks PowerPro uses to track active windows for the active bar. If you find PowerPro active bars are not responsive, try checking *"Use timer for active buttons"* on the Advanced dialog.

Configuring with the GUI dialogs : Command Lists

Omitting windows or words from active window lists

You can use the omit list edit box on the command lists *Setup >All Bars and Menus* dialog, either to omit part of a window's title or to omit windows completely from the list of active windowsc or from the active task buttons.

To omit a word, type the word followed by a comma. For example, you could use this technique to delete vendor names.

To omit an entire window, type the window name as it appears in the caption title of the window followed by a comma, e.g. Program Manager in the omit list will mean that no entry for Program Manager will appear.

If you include a string followed by an asterisk (*) and comma in the omit list, then any active window with caption text starting with that string will be deleted. For example, 1MBFort* will delete any program name starting with 1MBFort.

You can also delete any window associated with the program filename.exe by including =filename in the omit list (no .exe, no path).

For example, specify: Microsoft,*Notepad

to remove Microsoft from all window titles and to remove any text entry ending in "notepad".

Configuring with the GUI dialogs : Command Lists

Automatically running commands when windows open

purpose

You can automatically run commands each time that a window with a specified caption is first created. A command could send keys to the window, or press a button, or set the window position, or move the window to an existing virtual desktop, or execute any other Powerpro command.

(For advanced user, you may also want to look at Hooking Windows Events.)

configuration

Use a special command list to do this.

Specify the command list name in the *"Open"* drop down box on the *Command Lists >Setup >Special Lists* dialog. Once this is done, each time a new window is opened and the caption matches a command list item name on that command list, Powerpro will execute the corresponding command from the command list.

Each item on the command list corresponds to a command you want to run when a specific window opens.

The item's *Name* specifies a caption list of the window(s) to be affected. Use xxx* as a command list item name to match any captions starting with xxx, *yyy to match any captions ending in yyy, and *zzz* for captions containing zzz anywhere. You can also specify a entry of =exename to match any window created by the program with .exe file name exename (no path, no .exe). Finally, you can specify filedialog to match file open/save/save as windows, explorer to match single or dual pane explorer windows, explorer1 for single pane explorer windows, or explorer2 for dual-pane explorer windows. You can use &() expressions in the item name; the expression is re-evaluated each time a new window is opened and checked.

the command for each item

To press specific buttons on the specified windows, use *Keys to send alt-x, when x is the button mnemonic letter, with {to autorun} at start of the *Keys keys sequence. Thus:

*Keys {to autorun} %ef

To position a newly opened window on the screen, use the command:

*Window Position x y w h autorun

To move the window to a specified desktop, use the command *Vdesk MoveAutoRun.

To show a menu, you can use *Menu Show. However, you may have to put *wait 1 in the main *Command* and *Menu Show in *More Commands* if you find the menu disappearing as soon as it is shown due to other activity on your system when the window first opens.

special characters in the item's Name field

If you only want to run commands if the new window is a dialog, precede the caption/path with a #.

If you only want to run the command if the new window is **not** a dialog, precede the caption/path with a $.

If you want the command to apply to single pane explorer windows (folder windows) only, precede it by an !.

If you want the command to apply to 2-pane explorer windows only, precede it by an @.

limiting the matches

Powerpro normally executes all commands in the command list which match the caption. However, if a matching caption has the command:

Command	*Script
Parameter	quit

then no further command list entries are checked.

examples

Suppose you create a command list with these entries and specify this list's name on the Command List Setup dialog

Name	$*notepad
Command	*Window Position 30 50 100 200 autorun
Name	*bothersome dialog*
Command	*Keys {to autorun}{en}
Name	*explor*
Command	*Vdesk MoveAutorun explorer

Then whenever a non-dialog window with caption ending in notepad was opened, it would be positioned to 30 50 and sized at 100 200. Also, whenever a window with "bothersome dialog" in its caption was opened, the Enter key would be sent to it. If a window containing Explor in its caption appeared, it would be moved the desktop named explorer (this desktop must already exist).

Configuring with the GUI dialogs : Key/Mouse

"Key/Mouse" dialog

use hot keys to expand the way you interact with Windows

purpose

You can use hot keys and mouse actions to perform any Powerpro command, such as starting a program, changing the look of a window, changing your windows configuration, sending keys, showing a menu, or running a script. (If you need to set a very large number of hotkeys, consider using Keyboard Macros instead.)

configuration

Your hotkey definitions are listed in the *Key/Mouse* tab of the configuration dialog. Press *New* or *Clone* to add a new hot key, *Delete* to remove one, *Edit* (or right or left double-click) to change a hot key, and *Disable* to temporarily disable a hotkey. Or you can right click on an item to access these functions from a menu.

You can click the column headings to change the sort order.

You can see the complete contents of a field which does not fit in its column by holding the mouse over the field which causes a tool tip to appear; you may have to click on the field to update the tool tip. You can also resize the configuration dialog by dragging any border to see more of the fields in the entries in the command list.

further information

Hotkeys normally do not function if a Dos or console windows is active. But you can change this and many other aspects of hot key performance with the *Key/Mouse Setup* dialog.

You can create global macro keys to paste text phrases or paragraphs by assigning the *Keys command or the *Clip File or *Clip TextPaste command to a hot key. Or create a menu of *Keys commands and use a hot key/mouse action with *Menu Show

You can assign double click to middle clicking the mouse by associating the **middle anywhere** hot key with the command *Mouse leftdouble.

By using mouse stroke hot keys which execute *Menu Show commands, and which depend on which program is active, you can define menus which depend on the active program and which appear after a mouse stroke.

The Win modifier key is also used internally by Windows; you cannot redefine hot keys that Windows has already defined.

Note on chording: some mouse drivers "miss" the second mouse up when two mouse keys are released at once leading to strange mouse behavior; to clear, you may have to press and release each mouse key separately.

Configuring with the GUI dialogs : Key/Mouse

Window-specific hot keys

purpose

You can define hot keys which function depending on whether or not windows you specify are active, or on whether or not the mouse is at a screen corner or edge or quadrant. This allows you to define hotkeys to have different actions depending on the active window.

configuration

To define a hot key which only functions for specified programs, define a hot key as usual, but use the Target Window edit box on the hot key configuration to enter the caption list of windows for the hot key.

To define a hot key which functions for all except a specified list of programs, put a ~ at the start of the Target Window edit box and then list the windows for which the hot key is to be ignored.

examples

The following command definition sends the key sequence Alt-F S Alt-F4 to Notepad and Explorer only (this sequence saves the active file and then exits):

Command:	*Keys
Parameter:	%fs%{f4}
Target Window:	=Notepad,Exploring*

The following will only send keys if the mouse is at the left edge of the screen:

Command:	*Keys
Parameter:	%fs%{f4}
Target Window:	@left

further information

You can define a hot key to have specific meaning for certain programs and other meanings for other programs by defining the hot key multiple times with different Commands and Target Window entries.

When you press a key which is a hot key, Powerpro uses the following searches to select from the possibilities:

- First, search to see if there are any hot keys defined solely for the currently active window. If so use them.
- If there are no hot keys specifically for this window, but there are hot keys for all windows or all but certain windows (and the active window is not excluded), execute them.
- If the only hot keys which are defined are specific to other programs, then send the raw input key to the currently active program.

Configuring with the GUI dialogs : Key/Mouse

Hot key/mouse actions

You can use these actions to activate commands with hotkeys.

(Mouse 4 and 5 actions are only available in Win2000 or later versions of Windows.)

prefix key then char	press and release the prefix key then press any key
screen top left	move mouse to top left screen corner
screen top right	move mouse to top right screen corner
screen bottom left	move mouse to bottom left screen corner
screen bottom right	move mouse to bottom right screen corner
bump screen left/right/top/b	moving mouse to screen edge
left anywhere	left mouse click anywhere

middle anywhere	middle mouse click anywhere
right anywhere	right mouse click anywhere
mouse 4 anywhere	click mouse 4 button anywhere
mouse 5 anywhere	click mouse 5 button anywhere
left desk	left mouse click on desk top
middle desk	middle mouse click on desk top
right desk	right mouse click on desk top
left caption	left mouse click anywhere on caption; if no modifier keys, you must wait momentarily
middle caption	middle mouse click anywhere on caption
right caption	right mouse click anywhere on caption
mouse 4 caption	click mouse 4 button in caption
mouse 5 caption	click mouse 5 button in caption
right caption double	right mouse double click anywhere on caption
middle caption (left half)	middle click on left half of caption
middle caption (right half)	middle click on right half of caption
right caption (left half)	right click on middle half of caption
right caption (right half)	right click on right half of caption
middle sys menu	middle click on system menu icon in caption
right sys menu	right click on system menu icon in caption
middle minimize	middle click on minimize icon in caption
right minimize	right click on minimize icon in caption
left close box	left click on close box icon in caption
middle close box	middle click on close box icon in caption
right close box	right click on close box icon in caption
middle maximize	middle click on maximize/size icon in caption
right maximize	right click on maximize icon/size in caption
middle border	middle click on window border
right border	right click on window border
left double anywhere	left double click
middle double anywhere	middle double click (you must also define middle single as hot key)
right double anywhere	right double click
left hold	press and hold down left mouse button
middle hold	press and hold down middle mouse button
right hold	press and hold down right mouse button
mouse 4 hold	press and hold down mouse 4 button
mouse 5 hold	press and hold down mouse 5 button
left drags left/right/up/down	press left mouse, drag less than 25 pixels horizontally, release. Separate left, right, up, and down short drag hot keys are supported. You can control the maximum number of pixels that

	PowerPro will interpret as the hot key using *>Key/Mouse >Setup*.
wheel forward/back	move mouse wheel one position forward then quickly back; must be at least one second after any other mouse wheel movement to avoid inadvertent activation (not available in win95) .
wheel forward only/back only	move mouse wheel one position forward or back. Use *>Key/Mouse >Setup* to control whether the mouse wheel is also sent to other applications.
chord l+m	chord (simultaneously press) left and middle button
chord l+r	chord (simultaneously press) left and right button
chord m+r	chord (simultaneously press) middle and right button
horizontal move	move mouse back and forth horizontally
vertical move	move mouse up and down vertically
tap shift	press and quickly release shift key
tap ctrl	press and quickly release ctrl key
tap alt	press and quickly release alt key
tap caps lock	press and quickly release caps lock key
tap apps	press and quickly release apps key (beside right ctrl)

Configuring with the GUI dialogs : Key/Mouse

Hot key and mouse "Setup" dialog

settings for all hotkeys

purpose

Use the Setup button on the Key/Mouse tab to fine tune the Hot Key and Mouse features.

configuration

You can specify that double tapping is needed for the tap ctrl/alt/shift hot keys or for function key hot keys (this only applies to function keys used without Alt, Ctrl, Shift, Win).

To avoid activating the hot key action if a full screen program or a DirectX program is running, check "Disable bump or screen corner if full screen program running".

You can use the checkbox to disable screen edge bump and screen corner hot keys while a menu is showing to prevent accidentally closing the menu when selecting an item near the screen edge or corner.

You can use the check box to specify that Powerpro will wait for up to 1.5 seconds for all modifier keys (alt, ctrl, shift, win) to be up before executing any hot key command. If unchecked, Powerpro only waits for commands which send keys.

To make it easy to navigate menus shown by hot keys involving Ctrl or Shift, you can use a check box to specify that Ctrl is Enter and Shift is down arrow while a menu shown by a hot key is open. Note that you can assign a *Show Menu hot key to Ctrl+down (arrow) or Ctrl+up arrow as well, and then use the arrow and Ctrl keys to navigate the menu. Use Alt or Esc to dismiss a menu.

You can specify the PowerPro should ensure the list of hot keys on the configuration dialog reflects the local keyboard.

You can specify that PowerPro should recognize keyboard hotkeys entered when a Dos or console window is active, except for tap keys and *Macro keys. Gray check if you find slow performance with *Keys commands; however, if you gray check you may have problems if you send hot key characters with *Keys.

You can use the checkbox to specify that PowerPro should prevent the mouse wheel actions from being sent to other programs when a mouse forward only or back only hot key is defined. Gray check to block the sending only if the target field is matched. This may not work in all cases.

You can specify the character to be used for "char then key" hot keys. The character cannot be a letter or a digit, and you cannot use the shift key with the character.

You can specify a delay in milliseconds for the screen corner and screen bump commands; the command will only be executed if you leave the mouse cursor in the corner or at the edge for at least the specified delay.

You can specify the minimum distance in pixels from the screen corner needed to activate screen corner hot keys.

You can specify a delay in milliseconds for the tap key commands; the command will only be executed if the tap key is held down for **less than** the specified delay time.

You can specify a minimum hold time for mouse press and hold hot keys,

You can fine tune the mouse stroke hot keys by adjusting the minimum length of the stroke in pixels, the maximum deviation from horizontal/vertical, and maximum time allowed to complete the stroke. You can also specify a stop time; if the stop time is greater then 0 then the mouse must stop after that number of milliseconds after the completion of the stroke for the hot key to be activated.

Configuring with the GUI dialogs : Key/Mouse

Entering hot key/mouse action information

using the edit dialog for a hot key/mouse action

Key/Mouse: At the top of the hot key edit dialog is a set of check boxes and a drop down used to select hot keys/mouse actions and modifier keys for the hot key/mouse action. You can also use the "Type Key" edit box to type a key directly (eg Alt+Shift+T); use tab, esc, or the mouse to exit this edit box.

Disable: Check to disable a key without removing it from the list. Disabled keys are prefixed by X- in the list of hot keys.

Target: You can assign hot keys/mouse actions which run only when a specified windows are active or when the mouse is at a specified position by using the Target Window edit box. Leave this edit box blank to have the hot key apply to any window. Enter a captionlist to have the hot key apply only to the windows or mouse positions matched by that list.

To help you remember the purpose of the hot key, you can record a comment in the Target Window edit box by putting a semi-colon (;) ahead of the comment.

virtual key code and scan code

The virtual key code is shown beside the selected hot key. You can experiment with entering your own key code >0 and <256 by selecting *"Enter virtual key"* from the drop down. If a key code corresponding to a defined hot key is entered, that hot key name will be shown in the hot keys list.

Some keyboard drivers do not send unique virtual key codes for some specialized keys. If you want to try to use such a key as a hot key, you can experiment with sending the scan code as well as the non-unique virtual key code. The easiest way to enter the scan code information is to click the "Enter Key" edit box, press the key to be used, and then press Escape. Make sure

"Use Scan" is checked. If this does not work or if the resulting hot key does not function, try using the Exec TraceKey command to capture the key information, and then enter it by checking "Use Scan", checking "Ext" if the extended key bit is set, and entering the scan code as a decimal number. (Note: PowerPro may be unable to prevent the normal action of such a key even if it can define it as a hot key; you can use the Tweakui program under explorer>commands or work with software provided by the driver manufacturer to deactivate key command in this case)..

scan code for left and right tap keys

You can use the scan code to have different commands for the left and right shift, ctrl, and alt tap keys. Select the tap key, check "Use Scan", and then use

Left Shift: scan=42 ext = 0 (vkey=16)

Right Shift: scan=54 ext=0 (vkey=16)

Left Ctrl: scan=29 ext=0 (vkey=17)

Right Ctrl: scan=29 ext=1 (vkey=17)

Left Alt: scan=56 ext=0 (vkey=18)

Right Alt: scan=56 ext=1 (vkey=18)

keytrap plugin

The keytrap plugin provides more facilities for hot keys than PowerPro that you may want to use. It has the following capabilities :

1. There are some keys that PowerPro cannot access (such as some of the special keys on a Logitech keyboard) that Keytrap can successfully reprogram.
3. Ability to use any key as modifier
4. Ability to control whether hot key is signalled on key up or key down
5. Ability to control whether or not key is eaten.

command section

The command entry controls at the bottom of the edit hot key dialog are used to change the command or built-in (*) command run when the hot key is activated. Use the start menu or find buttons or select a command from the drop down. You can also drag/drop files or desktop icons to the dialog to set the command.

Configuring with the GUI dialogs : Scheduler

Scheduler "Setup" dialog

general settings for alarms

purpose

Powerpro has alarms to let you start commands at defined times. The Scheduler "Setup" dialog provides control of these features.

configuration

The Scheduler Setup dialog is displayed when the Setup button on the Scheduler dialog is clicked. Set check boxes to:

Have Powerpro ring alarms which occur when Powerpro is not active. Otherwise, missed alarms are not rung but are recycled or discarded according to the alarm's Interval setting. (However, alarms less than four minutes old are always rung).

Play the alarm sound when an alarm displays a message box (gray check to loop sounds).

Play the alarm sound when a command is run by an alarm.

Keep an alarm log.

Specify that a ringing alarm should stop any running screen saver.

Specify whether captions for alarm message boxes should be set to the message.

Specify whether changes should be allowed to scheduled messages when they are first shown.

Specify whether or not alarm messages should be shown in front of the active window when the alarm rings. Gray-check to specify messages should be shown "always on top".

Specify whether alarm message windows show take the focus.

Specify whether the Escape key should close alarm message boxes.

Specify whether the date picker or separate year/month/day edit boxes should be used to set the date for alarms in the Scheduled item dialog.

Specify whether alarm message dialogs show use offset (stacked) positions when more than one alarm message is open.

The dialog also contains several drop down lists which you use to:

> Set the format for dates in the alarm list.
>
> Set the screen position for alarm message windows.
>
> Set a chime at a regular time during the hour (e.g. every 15 minutes).
>
> Set a resource warning level percentage to have Powerpro display a message box whenever GDI or USER resources fall below this level. You can also monitor resource usage with a button label set by the command list item dialog. For Win95/98 only.

Use the Media dialog to set the sound associated with alarms and chiming.

Configuring with the GUI dialogs : Scheduler

"Scheduler" dialog

setting scheduled commands or alarms

purpose

Use the Scheduler dialog to set alarms to run commands or display messages at predefined times. You can set alarms to repeat on a regular basis. You can also set alarms to be run after your computer has been idle for a specified time, after an idle alarm, and for when Powerpro is initially started.

configuration

The list box of the alarm contents dialog shows the list of alarms, sorted with the earliest at the top.

Use the New or Clone button to add a new alarm, the Delete button to remove an alarm, the Edit button (or double click on an alarm) to change it, or Disable to temporarily disable an alarm without removing it from the list. You can also right click on the list of alarms to access a popup menu.

You can see the complete contents of a field which does not fit in its column by holding the mouse over the field which causes a tool tip to appear; you may have to click on the field to update the tool tip. You can also resize the configuration dialog by dragging any border to see more of the fields in the entries in the command list.

Adding or changing an alarm activates the Edit Scheduled Command dialog.

further information

Alarms are usually used to start commands, but you can also use alarms to close running programs by running a *Window close command.

To quickly add a new message box (reminder) alarm, run the *Configure AddReminderMessage command.

To run a series of commands when an alarm is rung, use More Commands or use the the alarm to execute a script. To run a series of commands at startup, associate a startup alarm with a script.

You can specify the year, month, day ordering for dates, and other general aspects of alarms, using the Scheduler *Setup* dialog.

Powerpro only checks to see if a scheduled event should occur once per minute. If you set an event for now, it will not occur until the next minute.

PowerPro will not run scheduled events while the Configuration dialog is open.

scheduled messages

If you use *Message for the command to show a reminder at the scheduled time, when the message box alarm rings, you can change the message text and re-schedule it. Use an option on the scheduler *Setup* dialog to control whether changes are initially allowed on this message dialog (disallowing changes prevents messages being changed accidentally).

You can also use the message dialog to select the time until the next alarm from a drop down box or by entering at as months:days:hours:minutes.

You can request that the message alarm be copied and shown again as well as being saved to be shown again after the interval time.

Configuring with the GUI dialogs : Scheduler

Alarm log

logging scheduler alarm events

You can ask Powerpro to log alarm events by using the *Keep Alarm Log* check box on the Scheduler *Setup* dialog.

The log file will have the same name as the current configuration file used for Powerpro, except that the file extension will be .alarmlog. For example, the log file for the default configuration is PowerPro.pcf.alarmlog. The log is always placed in the same directory as the Powerpro.pcf file.

A log file entry will be written whenever an alarm rings. It will consist of the following fields, separated by blanks:

Current Year

Current Month

Current Hour

Current Minute

Alarm Year

Alarm Month

Alarm Hour

Alarm Minute

Alarm command and parameters.

Alarm work directory/message.

Configuring with the GUI dialogs : Scheduler

Suspending alarms

You can suspend ringing of alarms by executing the following command (e.g. though a button or menu item):

Command	*Exec
Action	Alarms
Parameter:	off

To resume alarm ringing, use

Command	*Exec
Action	Alarms
Parameter:	on

To reverse the status, i.e. suspend alarm ringing if it is active, or resume alarm ringing if it is suspended, use

Command	*Exec
Action	Alarms
Parameter:	toggle

When alarm ringing is resumed, alarms which would have rung when alarm ringing was suspended are rung or discarded according to the setting of *"Ring Missed Alarms"* on the Scheduler Setup configuration dialog.

Configuring with the GUI dialogs : Scheduler

Running programs after system idle

running programs after the system is idle for a specified time

You can run a program when the system has been idle for a specified time (or after this idle period ends) by using an alarm. For Powerpro, idle means that no keyboard or mouse input has been received. Other programs may be running but as long as no keyboard or mouse actions occur then the system is considered to be idle.

In the *"Edit scheduled item"* dialog, use the *Idle* radio button. Then set the time to the amount of idle time to elapse. For example, set the time to 00:30 to indicate that the program should be run if the system is idle for 30 minutes.

For post-idle alarms, select the *Post-idle* radio button. The post idle alarm command will be run after you move the mouse or press a key after an idle alarm has occurred. Note that you must use an idle alarm in order for the post-idle alarm to function; you can use an idle alarm with command set to (none) if necessary. When a post-idle alarm occurs, the variable **lastidletime** is set to the total number of seconds the computer was idle (including idle time before the idle alarm occurred).

For example, if you want to mute sound while you are away from the computer, use an idle alarm of *Exec VolumeAll 0 and a post-idle alarm of *Exec VolumeAll 255.

The minimum idle time is one minute.

You can have many idle alarms each with different idle times.

The Interval setting for idle alarms is forced to be *"Save for re-use"*. If you want to remove an idle alarm, you must delete it with the configuration dialog.

If the program to be run by the idle alarm is already running, Powerpro will not restart it.

Powerpro only detects mouse or keyboard events which are directed at GUI programs. It does not detect input to Dos or Console programs. If you use such programs extensively, you may find Powerpro activates idle alarms in error.

Configuring with the GUI dialogs : Scheduler

Entering information for a scheduled command

using the "Edit scheduled command" dialog

Type: Select from an idle, post-idle, start-up, or normal alarm. Idle alarms are rung after the specified number of hours and minutes of no keyboard or mouse action, start-up alarms are rung when Powerpro starts, and normal alarms are rung at a specified time.

Date and Time: Set the scheduled time for normal alarms using the date, hour, and minute controls. This is the time the alarm command or message will be run next. Use the interval drop down to control repeated display or the message or running of the command.

Interval: Scheduled commands can be configured to repeat: use the interval drop down to control whether and when the alarm is repeated. The alarm is first executed at the set date and time. Then the recycle interval is added to that date and time to obtain the next date and time that the alarm will be executed. You can select a standard recycle interval from the drop down box or you can enter a specific interval as up to four numbers separated by colons: months:days:hours:minutes. You can also enter an interval of months, hours, days, minutes using numbers and words, like *3 months 2 days* or *1 day 5 hours 4 minutes*. Only the first two characters of the word need be specified.

S M T W T F S: For alarms that repeat (where the interval is not *"once only, then erase"*), you can select the days on which the alarm will execute by checking under one or more of these letters (for Sunday, Monday, etc).

Command: Enter the command to be run in the command entry controls at the bottom using the start menu or find buttons or by selecting a command from the drop down. Use the built-in (*) command *Message to enter a reminder message. You can also drag/drop files or desktop icons to the dialog to set the command. See here for details of command entry controls

Run if missed: Check to have command run if it occurs when PowerPro is not running: then missed events will be run once when PowerPro next starts. This setting gives individual control of whether missed events are run. You must uncheck the setting on *>Scheduler >Setup* which applies to all alarms for if it is checked all missed events are always run.

No sounds: Check to disable any sound for this event regardless of sounds set on scheduler setup.

Log: Check to have alarm actual time and command written to the log file pproconf.alarmlog each time the alarm is activated. You can also check *"Keep Log File"* on *>Scheduler >Setup* dialog to have all alarms logged, regardless of whether the Log item is checked for any individual alarm.

examples

for an alarm to occur each hour on the half hour, starting at 9:30: set the day and time to 9:30 and set the interval to one hour for an alarm to occur once a day: set the time and first day, and select interval "1 day"

for an alarm to occur on 15th of each month: set initial month and date of 15th and select interval of one month

for an alarm to occur Monday, Wednesday, Friday: set the first date, set interval 1 day, and check M, W, F check boxes

Note: if you want the ability to quickly configure a new reminder message, then assign the command *Configure NewReminderMessage to any button or menu or hot key.

Configuring with the GUI dialogs : Timers

"Edit timer" dialog

setting timers and their associated commands

purpose

PowerPro has a set of 26 timers. You can configure them using the Timer dialog from the Timer tab on the configuration tabbed dialog. Double click on a timer to configure it.

You can specify that the timer should start automatically when Powerpro starts. You can specify that the timer values should be saved and restored when Powerpro starts and stops. You can indicate that the timer should count down.

configuration

Select the timers tab and double click on a timer in the list to be changed. Use the check boxes at the top as follows:

Running	Start (checked) or stop (unchecked) the timer.
Down	The timer counts down and stops at zero (the reset command is executed at 0).
AutoStart	The timer is started as soon as PowerPro starts running.
AutoSave	The timer value is automatically saved periodically.
Log	The timer is logged.

Next you can specify that a timer should run only when a RAS connection is active or when a specified program is active (the foreground window):

Run when dialup... To associate a timer with a RAS (dial-up) connection, check the "Run Timer when Dialup Active" check box and set the timer name to the dial up name. Powerpro will automatically start and stop the timer according to the status of the RAS connection. You can associate more than one timer with the same connection: e.g. have a daily timer and a monthly timer. (To create a daily/monthly timer, add an alarm which clears the timer daily/monthly). To have a timer which runs when any dial-up is active, set the timer label to "*any".

Run when program... To associate a timer with a program, check the "Run Timer Program Active" check box and set the timer name to the exe file name of the program to be timed; omit the path and the .exe from the name (e.g. netscape for Netscape Communicator). Powerpro will arrange for the timer to be running only when the specified program is the foreground (active) program.

associated commands

You can also associate a command with starting, stopping, and resetting the timer using the command entry controls.

You can use the Find or StartMenu button to set the command, or select a built in command from the drop down. You can also drag/drop files or desktop icons to the dialog to set the command.

The **Reset** command is used in conjunction with the Reset Hour, Minute, and Second values.

For timers which count down:
whenever the timer reaches zero, any associated Reset command is executed. If any of the Reset Hour, Minute, or Second is greater than zero, the timer is reset to that value. Otherwise, the timer is stopped.

For timers which count up:
if any of the Reset Hour, Minute, or Second is greater than zero, the associated command is executed whenever the timer reaches a multiple of the number of seconds represented by the Reset values.

example

To execute a command every 5 seconds, set the reset second to 5 and the reset hour and minute to 0. Or to run a script every 1 minute and 30 seconds, set the reset minute to 1, the reset second to 30, and the reset command to a *Script run command.

You can also use the *Timer Set command to set a timer value and state.

Configuring with the GUI dialogs : Timers

Timer logs

You can ask Powerpro to log timer events by using the *Log* check box on the Timer editing dialog for each timer.

The log file will have the same name as the configuration file used for Powerpro, except that the file extension will be .timerlog. For example, the log file for the default configuration is PowerPro.timerlog. The log is always placed in the same directory as the Powerpro .pcf file.

A log file entry will be written whenever a timer starts, stops, is cleared, or is re-set. As well, when Powerpro shuts down, a stop timer entry will be written for any running timers. When Powerpro starts up, a start timer entry will be written for any automatic start timers.

The logs have fixed-format records, structured as follows

Column	Contents .
1	Always blank.
2-8	Button of last timer command.
9	Always blank.
10	Timer id (single character).
11	Always blank.
12	Action: "+" if timer started, "-" if timer stopped, "0" if cleared, "R" if reset
13	Always blank.
14-17	Year when event recorded.
18	Always blank.
19-20	Month.
21	Always blank.
22-23	Day.
24	Always blank.
25-26	Hour (military clock, i.e. 24 hour time)

27	Always blank.
28-29	Minute
30	Always blank.
31-32	Second
33	Always blank.
34-41	Total timer value in seconds.
42	Always blank.
43-47	Whole hours in the timer.
48	Always blank.
49-50	Whole minutes in the timer.
51	Always blank.
52-53	Seconds in the timer.

To be clear: the timer value is shown in two different formats: columns 29-36 show the timer value in seconds. Columns 38-48 show the timer value as hours, minutes, seconds.

You can set timers by using the "Timers" dialog or by using *Timer commands.

Configuring with the GUI dialogs : Media

"Media" dialog

The Media dialog is used to set sounds, screen saver, and wallpaper, and to control automatic changes of these files by Powerpro.

configuration

To change sound, wallpaper, or screen saver information, double click on an entry from the list in the Media tab or use the Edit button. You will be able to change the associated file and select whether and how often PowerPro will automatically change the file.

If you let PowerPro automatically change wallpaper, you can specify that when PowerPro picks a new wallpaper, it should first pick a random folder from the parent of the current wallpaper's folder, then pick a new wallpaper from within that folder. Check this option to have a random folder chosen only at PowerPro startup or gray check to have a random folder chosen for every automatic change. You can also specify that wallpaper changes should not occur if a full screen program, like a game or screen saver, is running.

If you use PowerPro to change screen savers, you can select to have PowerPro change the saver each time it is started by PowerPro or the system; or change it at every PowerPro startup; or at a timed interval.

When Powerpro changes the screen saver, you can set whether or not Powerpro will stop any running saver and restart with the new saver using the Setup dialog.

Powerpro allows you to use jpeg files as well as bmp files as wallpaper. If you set a jpg file as the wallpaper, Powerpro will convert it to an 8 bit (256 color palette) .bmp for display. If you prefer to use a 24 bit (true color) image, convert the .jpg to .bmp using an image editor or image converter, then specify the .bmp file in the Media dialog.

Configuring with the GUI dialogs : Media

PowerPro sounds

Powerpro sounds are set from the Media dialog.

You must have a sound card and the appropriate drivers (or the PC speaker driver) to hear sounds in Windows.

Powerpro supports access to many standard Windows sounds in the Registry plus these sounds:

Powerpro Chime	Plays whenever Powerpro chimes (see Scheduler Setup dialog)
Powerpro Alarm	Plays whenever Powerpro runs an alarm (see Scheduler Setup dialog). Use a single asterisk to have the PC Speaker beep for alarms.
Powerpro Clip	Plays whenever Powerpro captures a clipboard item.

Configuring with text files

Maintaining configuration using .ini files

Note: The format of text files for **import** and **export** from pproconf has changed in PowerPro version 4, now using a standard .ini file format. These files can be edited in any plain text editor; or by using standard inifile features as provided by the ini.dll plugin.

PowerPro lets you store the configuration of command lists, scheduled events (also known as Alarms), and hot keys in text files as a supplement or replacement for this configuration information in the pcf file. A text file can be used to store one or more command lists. Scheduled events and hot keys must be stored in a separate file each.

To get examples of text configuration files, you can export your existing configuration using the *Export as Text* button on the *Setup* tab of the configuration program's main dialog.

importing text configurations with commands

You can import the files directly into running PowerPro using the Configure ImportCL, Configure ImportSched, and Configure ImportHot commands. If you do not specify the full path to the ini file with these commands, then the file is assumed to be in the config subfolder of the folder containing the .pcf file.

PowerPro follows these rules when importing text configuration through these commands:

- Importing scheduled events or hot keys **replaces** any existing scheduled events or hot keys which came from a Configure Import and **supplements** any scheduled events or hot keys stored in the .pcf file.
- Scheduled events from text files which are at least four minutes out of date are not run, unless the RunIfMissed keyword is included in the configuration.
- Text imports into PowerPro must be redone each time you start PowerPro. You can use a Startup scheduled event to run the Configure Import commands.
- Importing a command list from text with the same name as an existing command list from text replaces that command list. If there is an open bar based on that command list, the bar is closed. You can use the autoshowbar keyword on import to show imported command lists as bars, or you can use *Bar Show.
- Importing a command list with the same name as a command list from the pcf file is not permitted.
- Text imports created from Configure Import commands are never saved in the pcf file.
- You can only use Configure ImportSched from a startup scheduled event; you cannot use this command from any other type of scheduled event.
- If you use the GUI interface to reconfigure while PowerPro is running, all text imports made directly into PowerPro using the Configure Import commands are lost. You can use the Reconfigure command list to automatically re-import text configurations after a GUI configuration if you like.

To maintain your configuration in text and read it at startup, create two startup events in the pcf file: one should be a Configure ImportSched to read the scheduled events from text; the other should be a *Script RunFile MyText to run a script MyText which has Configure ImportCL commands to read your command lists (and possibly also Configure ImportHot to read any hot keys/mouse actions).

importing text configurations with the dialogs

You can also import text files into the pcf configuration using the "*Import from text*" button on the *Setup* tab. In this case the configuration information is stored in the pcf file and becomes part of the permanent configuration read by the PowerPro program. You are also allowed to import command lists with the same name as any command list in the pcf file, to replace that list.

You can use Configure WriteAllToPCF to write all of your configuration, including any imported text files, to a pcf file, where the configuration can be viewed using the pproconf program. This command is intended to help debug text imports by showing what has been imported to memory.

Configuring with text files

Layout of text configuration files

Text files follow the ini format standards. This means that the input consists of section headers and value assignment lines.

section headers

these consist of a word enclosed in square brackets [].

PowerPro text configuration has three types of section headers

[name:Properties] is used to give Properties information for command list configurations and is only allowed in command list files.

[n] is used to signal the start of a new command list item, a new scheduled event, or a new hot key. The word n can be any number; the actual value of n is ignored.

[x...] is used to head a section to be ignored by PowerPro: any section which has a name starting with x is ignored. For example, sections names [X], [xFiles], [XMarksthespot] would all be ignored. This means that everything up to the next section header is skipped.

value assignment lines

These take the form: value = text including keywords

Values: The value is a predefined word as described in detail below. Case is ignored. All blanks or tabs at the start of a line are ignored. The value is followed by one or more blanks then an equal sign then one or more blanks and finally the configuration text. All blanks at the start or end of the text portion are discarded.

Keywords: For many values, the text consists of a series of keywords separated by blanks. Keywords can appear in any order and case is ignored. Some keywords end in a colon which means that they are followed by further information, for example text color is indicated with: text: rrr ggg bbb. At least one blank must precede this extra information.

Comment lines: All blank lines in a configuration file are ignored.
Any line whose first non-blank character is a semi colon (;) is ignored,
except for these "Special Comment Lines": ;include ;if ;global ;local ;static

special comment lines

- Configuration files can include other files with a statement of the form:

```
;include c:\path\to\file.ini
```

The full path to the file should be given. The text in this file is processed as if it occurred where the ;include line occurs. The included file may itself contain ;include statements, up to a maximum depth of 5.

- You can use an expression for inclusions:

```
;include (expression)
```

You must use parentheses around the expression

- For importing directly into PowerPro, you can set variables to expressions with ;global var = expr, ;local var=expr, or ;static var=expr. This is useful with conditional configuration import.
- You can conditionally import sections of the configuration file with

```
;if (expression)
statements-1
;else
statements-2
;endif
```

The expression is evaluated and if true, statements-1 are processed; otherwise, statements-2 are processed. The else part is optional. You can nest if-else-endif.

continuation of lines in ini files

You can continue any line to the next by ending the line with a backslash (\). The backslash must be the final character on the line; it cannot be followed by a space. PowerPro acts as if the following line is joined at the backslash. No intermediate space is inserted.

text configuration of commands

The command list, scheduled events, and hot keys configuration files all required that commands be specified. The same set of four values is used in all cases:

Cmd = text is used to specify the command; quotes are never used even for file names with blanks

Param = text is used to specify the command parameters, including the action for PowerPro commands (e.g. Close action for Window command would be Param=Close captionlist)

Work = text indicates the work folder for .exe files run as commands, or formatting information for Menu and some other PowerPro commands

How = text indicates how to start the program. text can be any of min, max, normal, traymin, hidden, hide (these last two are the same). It can also include switchto or topmost to set the "switch to if active" and "Top most" checkboxes.

To specify **multiple commands**, follow the above keywords by an ascending number for each set of commands and then use the sets of four keywords in order by the number. For example, a hotkey running multiple commands:

```
Cmd1 = c:\program files\myprog\myprog.exe
Param1 = -switch
Work1=c:\work\folder
how1=max
cmd2=wait
param2= for activewindow("=myprog")
cmd3=keys
param3=the quick brown fox
```

This specifies three commands: myprog.exe is started maximized, then PowerPro waits until it is active, and then the keys are sent.

As noted in the section on command lists below, for configuration of commands in command lists, the values Cmd, Param, Work, and How are preceded by one of the letters L, M, or R for specifying Left, Middle, or Right commands. For example

```
LCmd1 = c:\program files\myprog\myprog.exe
```

```
LParam1 = -switch
LWork1=c:\work\folder
```

file format for text configuration of scheduled events

Each item in the text configuration of scheduled events (also known as Alarms) must have a separate section with section header [n], where n is any number. The actual value of the number does not matter. Within the section, use the Cmd, Param, Work, and How values to specify the command as described above in the section on Text Configuration of Commands. Order does not matter, except that for multiple commands the sequence numbers must be ascending.

To specify the event time and any switches, use the event assignment value.
There are four possible formats, depending on the type of event:

```
Event = yyyymmdd hhmm recur: recurrence other keywords
Event = startup other keywords
Event = idle hhmm other keywords
Event = postidle other keywords
```

The first form is used for events set to happen at a specific date and time. The date must be specified as an 8 digit integer and the time as a 4 digit integer; use military time for events after noon (e.g. 1915 for 7:15 PM).

If the event is to recur, the recur: keywords must be specified (note the colon) and it must be followed by one of the recurrence strings from the scheduled event dialog, with all spaces removed (e.g. 1Hour, 1Day, 2Weeks, EndNextMonth). Alternatively, the recurrence can be specified as four numbers which are interpreted as Month Day Hour Minute to the next event occurrence.

If the specified time and date is earlier than when the setup is loaded, the recurrence is applied until a date and time after the load time of the script is reached; if there is no recurrence, then the event is ignored.

The other three forms of the event assignment are used to specify events to occur at startup, after hhmm hours and minutes of idleness, or after an idle event ends.

The other keywords which can be specified correspond to the check boxes on the scheduled item dialog; they are

Log	Log this alarm regardless of master setting on alarm setup.
RingIfMissed	Ring this alarm if missed by more than four minutes regardless of master setting on alarm setup.
NoSound	Play no sound for this event regardless of master setting on alarm setup.
NoSun	Do not execute event on Sunday; can also use NoSat, NoMon, NoTue, NoWed, NoThu, NoFri.
Disable	Disable this event.

example

The following configures two events: one disabled startup event to set a variable and one recurring daily event on weekdays only, which is logged, to run a program backup.exe.

```
[1]
event=startup disable
cmd=var="this is a test"
[2]
event=20040404 recur: 1day NoSun NoSat Log
cmd=c:\program files\back\back.exe
param=-switches
```

file format for text configuration of hot keys

Each hot key in the text configuration must have a separate section with section header [n], where n is any number. The actual value of the number does not matter. Within the section, use the Cmd, Param, Work, and How values to specify the command as described in the above section on Text Configuration of Commands. Order does not matter, except that for multiple commands the sequence numbers must be ascending.

For specifying a Target for the hot key, use

```
Target = text
```

where text is the same text as would be entered for the target in the configuration dialog.

To specify the hot key, use

```
key = keyname
```

where keyname can be any of the keynames found in the dropdown of the configuration dialog, with any spaces removed (e.g. a, 0, F2, MiddleDragDown, LeftArrow, Pad2). You can also specify VKnnn to for virtual key nnn, where nnn is a number between 1 and 255. You can optionally follow VKnnn by spaces then a scan code 1 to 255 and optionally spaces then 1 to indicated the Extended key bit.

To specify modifier strings, the keyname can optionally be preceded by one or more of Alt+, Ctrl+, Shift+, Win+, Semi+.

For upper case keys, like A or +, PowerPro will automatically provide the Shift+. Hence note that a is a different hot key from A (the latter is Shift+a).

The keyword Disable can optionally follow the key name to disable the hotkey.

For example, the following shows four hot keys. The first is disabled and has a target.

```
[1]
key = middlecaption disable
cmd = window
param = min under
target = *notepad*
[2]
key = alt+ctrl+a
cmd = message
param = you pressed ctrl and alt and a
[3]
key = middlehold
cmd = menu
param = folder c:\program files\powerpro\clips
work = cmd "clip menupaste"
[4]
key = VK255 114
cmd = message
param = virtual key 255 scan 114
```

file format for text configuration of command lists

Command list configuration files can specify both the properties for the command list and also the command list items.

The properties section provides the ability to input information contained on the dialog with four tabs which is shown by pressing the Properties button in the Command Lists tab of the configuration program.

The command list items are used to specify information shown on the item entry dialog: left, right, middle commands, icons, tool tips, and item names to be entered as well as item specific format information.

properties for command lists

The Properties section must come first in the command list and there can be only one Properties section per command list. The Properties section title must take the form

[name:Properties]

where name is replaced by the command list name to be imported, for example:

[mymenu:Properties]

You can put many command lists in one file by starting each new list with its own [name:Properties].

The **[Properties]** section has the following assignment line values:

copy =	Followed by command list name; all the properties for that command list are copied. Some can be overwritten by subsequent values in the current command list.
Backbmp =	Followed by path to bmp file for background, or * for desktop, or (none) to force none.
Skin =	Followed by path to skin .txt file.
format =	Followed by keywords giving general format for displaying command list [see details below]. It is possible to use multiple format values format1=, format2= etc.
active=	Followed by keywords about the display of command lists with active buttons [see details below].
activeinclude =	Followed by caption list used to limit active bar buttons.
autotraymin=	followed by a caption list of windows to be tray minimised whenever you minimise them. This will over-rule the "Auto tray min" setting in the Setup dialog, but won't be written to the pcf.
fontname =	Followed by name of font to use in text when display the command list.
fontinfo =	Followed by font keywords [see details below].
lockto =	Followed by information for locking bar to another [see details below].
folder =	Followed by path to folder for folder buttons.
foldermenu =	Followed by keywords used to specify menu folder options for folder button bars.
folderformat =	Followed by keywords specifying options for Folder button bars [see details below].

details of Format=

This specifies layout information found on the first two tabs of the Properties dialog. The following list details the keywords that can appear on format= lines (remember that you can use as many format= lines as desired; these can be numbered format1, format2, etc.)

back:	Followed by three numbers giving red, green, blue components of background color.
hover	Followed by three numbers giving red, green, blue components of hover color.
press	Followed by three numbers giving red, green, blue components of press color.

text:	Followed by three numbers giving red, green, blue components of text color.
hovertext:	Followed by three numbers giving red, green, blue components of hover text color.
presstext:	Followed by three numbers giving red, green, blue components of press text color.
UseTheme	Use theme for button colors and look.
roundbar:	Followed by bar roundness setting (0-7).
roundwhich:	Followed by integer 0-4 indicated which corners to round (0 all, 1 top, 2 bottom, 3 left, 4 right).
roundbtn:	Followed by button roundness setting (0-7).
hovergradient:	Followed by hover gradient setting 0-7 (representing dropdown index).
hovergradientreverse	Include keyword to reverse coloring.
pressgradient	Followed by press gradient setting 0-7 (representing dropdown index).
pressgradientreverse	Include keyword to reverse coloring.
iconsize:	Followed by one of 8, 12, 16, 24, 32, 64. 96, 128, 256.
maxtext:	Followed by maximum number of text characters to display.
height:	Followed by one number giving height of button in pixels (omit this keyword to use default).
position:	Followed by single word giving position of bar; use the positions from the drop down of the configuration dialog but remove all spaces and parentheses. The valid values are Floating, Locked, LeftCaption, RightCaption, LeftEdgeCurrentSize, RightEdgeCurrentSize, TopEdgeCurrentSize, BottomEdgeCurrentSize, Taskbar, TaskbarNoStart, FixedTopRight, FixedTopRightOffset, FixedTopCenter, FixedTopLeft, FixedBottomLeft, FixedBottomCenter, FixedBottomRight, LeftOfActive, RightOfActive,BottomOfActive, TopOfActive, FixedRigthCenter, FixedCenterLeft, LeftEdgeFullScreen, RightEdgeFullScreen, TopEdgeFullScreen, BottomEdgeFullScreen.
HOffset:	Followed by positive or negative integer giving horizontal offset (e.g. for caption position).
VOffset:	Followed by positive or negative integer giving vertical offset (e.g. for caption position).
HideAfter:	Followed by number giving milliseconds until bar is hidden; omit this keyword for non-hidden bars.
ShowIfBump:	Screen edge to be bumped to show hidden bars; followed by one of None, NearestEdge;TopEdge; BottomEdge; RightEdge; LeftEdge
SlideFrom:	Direction to slide hidden bar from when showing; follow keyword by one of FromLeft; FromRight; Horizontal; Vertical; FromBottom; FromTop; None.
Transparent:	Transparency value; keyword is followed by number 0 to 255.
Gradient:	Gradient for background color; keyword is followed by number 0 to 255.
VerticalGradient	Gradient for background color; keyword is followed by number 0 to 255.
MouseThrough	Mouse clicks pass through bar.
PixelBlendOver	Button pixels replace background pixels before pixel blending into underlying windows.
PixelBlendInto	Button pixels blend into background pixels before pixel blending into underlying windows.
Shadow	Show a shadow under the bar.

Left:	Followed by pixel position for left of bar with Position: Floating or Locked.
Right:	Followed by pixel position for right of bar with Position: Floating or Locked.
Top:	Followed by pixel position for top of bar with Position: Floating or Locked.
Bottom:	Followed by pixel position for bottom of bar with Position: Floating or Locked.
RLeft:	Followed by relative pixel position on scale -1000000 through 1000000 for left of bar with Position: Floating or Locked. Translated to actual position based on current screen resolution.
RRight:	Followed by relative pixel position on scale -1000000 through 1000000 for right of bar with Position: Floating or Locked. Translated to actual position based on current screen resolution.
RTop:	Followed by relative pixel position on scale -1000000 through 1000000 for top of bar with Position: Floating or Locked. Translated to actual position based on current screen resolution.
RBottom:	Followed by relative pixel position on scale -1000000 through 1000000 for bottom of bar with Position: Floating or Locked. Translated to actual position based on current screen resolution.
AutoShowBar	The command list will be automatically shows as a bar.
Flat	Bar buttons are drawn flat except when mouse is over them.
FlatFlat	Bar buttons are always drawn flat.
Tool tips	Include this keyword to have tool tips displayed.
BarIcons	Include this keyword to display icons of specified size on bar only
MenuIconSmall	Icons on menus are always 16x16 regardless of icon size.
ToolTipsMenuText	When command list displayed as menu, tool tip text is used as menu label.
MenuNoMaxText	MaxText: is ignored when command list displayed as menu.
TopMost	Bar will be displayed as always-on-top.
SameSize	Forces bar buttons to be same size as first bar button.
Vertical	Bar is oriented vertically.
Border	Black border is drawn around bar.
3DFrame	3dFrame is drawn around bar.
No3DColor	3dFrame is same color as bar.
TextUnder	Text will be placed under the icon for a bar.
TextCenter	Label text is centered.
TextRotate	Text on buttons is rotated.
TextVertical	Text on button runs vertically.
RightIcon	Icons are shown to the right of text.
BarSize	Size of bar is set of total size of visible buttons
BumpWithin	Screen bumping to show bar only works if bump is within borders of bar
Marker	A small colored marker window beside the edge where a bar is hidden.
NoFlicker	Avoids flicker for bars with *info buttons; may not be needed so try without before using since extra memory is required to avoid flicker.
HoverClick	Hovering over a button clicks it.
HoverClickMenu	Hovering over a button with a Menu command clicks it.
Activesubbar	Subbar shown depends on active windowt.

details of Active=

This value gives keywords used in display of command lists with active buttons; the following list shows the available keywords for Active=

Count:	Followed by maximum number of active buttons to be displayed (up to 25)
UseLast	Last item in command list is used to configure color, size, middle, right commands for active buttons.
TrayUseLast	Last item in command list is used to configure commands for tray icon active buttons
PressFore	Button corresponding to foreground window is shown as pressed.
MinOnly	Only minimized windows are shown as active buttons.
TrayIcon	Tray icons are included in active bar (must also configure as described in help).
PowerProTray	Tray icons created by PowerPro from minimization to tray are included as active buttons.
NoActive	Only tray icon programs, not active windows, are shown.

details of FolderFormat=

This value gives keywords used in display of command lists with folder buttons; the following list shows the available keywords for FolderFormat=

AutoRefresh	Folder bar automatically refreshed when corresponding folder contents changed.
ShowText	Text labels in addition to icon shown on bar.
SortDate	Folder buttons sorted by date.
SortDateDown	Folder buttons sorted descending by date.
SortExt	Folder buttons sorted by file extension.
NoSortFolder	Folder names are not sorted to top of bar.
IncludeHidden	Hidden files are included.
Explorer2	Use 2 pane explorer window when button right clicked.
NoLnk	Remove extensions .lnk, .pif only.
NoExt	Do now show file extensions in text labels.
UseLast	Use last button in command list for color, size, middle, right commands for active buttons.
FolderOnlyOnFavBar	Show folder name only for entries from favorite folders bar.

details of FontInfo=

This value gives information describing the font. The Fontname value must also be specified separately. The following list shows the FontInfo= keywords:

Size:	Followed by font size.
Weight:	Gives weight of font (amount of boldness).
CharSet:	Number giving character set of font; see Windows documentation for further details if omitting this keyword does not work for the font you want to use.
Italic	Use italic font.
Bold	Use bold font.

details of LockTo=

Specify as

LockTo = "clname" n1 n2 n3 n4 n5 n6 n7 n8

Here clname gives command list for target. It must be in double quotes if it contains blanks. n1 through n4 specify horizontal percentage, horizontal offset, vertical percentage, vertical offset for target (clname); n5 through n8 similar values for locked bar (the one specified in the Properties).

example of a properties section

Following is an example of a properties section which sets a command list called "mylist" to have up to 10 active buttons, text color 255 128 0, myback.bmp as the background, tool tips, and the specified button height and width, caption position, and icons to the right of text.

```
[mylist:Properties]
format = text: 255 128 0 tool tips
format = height: 22 width: 18 righticon position: RightCaption
BackBMP=c:\path\to\back\myback.bmp
Active = count: 10
```

configuring command list items

Each new command list item must appear in a new section with a section header of the form [n], where n is any string of digits. The value of n does not matter. You can optionally precede the n by the command list name followed by a colon, e.g. [mylist:10], in order to create unique section names in an ini file for strict ini compliance (but this is not required by PowerPro.)

The lines in the section consists of assignment values indicating the label, icon, tool tip, format options, and left, right, and middle commands. You only need specify the assignment values for information you want to configure.

Label =	Followed by label for item.
Tool tip =	Followed by id for item as used in
id =	Followed by id for item as used in cl functions
Icon =	Followed by path to icon for item, or use Left, Right, Middle to refer to path of specific command. Follow icon path by ,n (including comma) to use nth icon in file, starting at icon 0.
Fontname =	Followed by name of font to use in text when display the command list item.
Fontinfo =	Followed by font keywords [same as FontInfo above].
Format =	Followed by item format keywords [see details below].

These are followed by the command assignment values, such as LCmd1= LParam1= LWork1= LHow1= which are described in the section on **Command Configuration** above.

details of the format= values

Width:	Followed by item width in pixels.
Height:	Followed by item height in pixels.
Text:	Followed by three numbers given red, green, blue for text color.
Back:	Followed by three numbers given red, green, blue for background color.

HideItem	The item is hidden.
HideIcon	The item icon is hidden.
Disable	Item is disabled if shown as a menu.
No3D	Item is drawn without 3D if shown as a button.
TextMulti	Multi line text box.
HoverClickMiddle	If hover clicking is selected for the bar, the middle command will be used.
HoverClickRight	If hover clicking is selected for the bar, the right command will be used.
NoHoverClick	If hover clicking is selected for the bar, it will not apply to this button.

example of a command list item

This command list Item (for a bar button) has an icon, label, no tool tip, multiple left commands, no middle command, format information, and a single right command.

```
[14]
Label = this is the label
Icon = c:\path to\icon,2
format = height: 22 back: 12 33 44
Lcmd1 = c:\path\to\prog.exe
LHow1 = max
LCmd2 = c:\path\to\prog2.exe
LParam2 = params for prog2
LHow2 = min
RCmd = window
RParam = close "=prog"
```

For further examples, use the *Export as Text* button on the setup tab to export existing command lists.

You can also use Menu ShowFile with .ini files which follow the text file configuration ini format: the file extension must be .ini

Configuring with CL functions

Overview of CL functions

Use the cl functions to work with command lists. You can create new lists, add or remove items, change list properties, read or change command information, icons, tool tips, and labels, and read or change item format such as color or visibility.

Calls for the cl functions take the form:

cl.ServiceName(expr1, expr2,...)

where ServiceName is one of the function services described below and expr1, etc, are expressions which supply the parameters of the function. The case of letters in the ServiceName is ignored. (Although cl calls look and function like plugin calls, in fact cl is not a separate dll like a plugin, but is implemented within the PowerPro exe.)

Many of the cl services require you to specify the item in the list that you wish to access. You have two choices to specify an item's location: you can specify the zero-based index of the item in the list as a number, or you can specify the id of the item.

The id is a text label for an item, used solely by the cl functions. You can specify the id using id=xxx in text configuration files or using the configuration dialog. The id cannot consist solely of digits; it must have at least one alphabetic.

The cl functions can access command lists from either the pcf file or text configurations. Changes to lists from text configurations are never saved in the original pcf. However, changes to pcf command lists may or may not be saved, depending on other PowerPro activities (e.g. if an alarm rings, the pcf file is updated with the new alarm and all changes are saved, including changed lists). You can force changes to be saved with cl.savepcf, but you cannot guarantee that changes to pcf command lists will not be saved. If needed, you can use the Startup events and the Reconfigure command list to set command list values to an appropriate state.

Information on specific functions can be found in these Help topics:

Working with whole command lists

Getting single item attributes

Setting single item attributes

Configuring with CL functions

CL functions for working with the entire command list

In each description, clname is used as an expression to represent a command list name and clindex is used to represent an expression giving the integer index of an item or its id, hlist is a handle to a command list obtained from Create or Get service.

creating lists and items

cl.Create(clname) or cl.Create(clname,1)

Creates a new Command List with name given by the clname expression. The command list must not already exist unless the second argument is specified and set to 1, in which case, if the list exists, it is re-used but all of its items are removed by the call to create. Returns 1 if successful, empty string otherwise. Returns handle which can be used to call other cl services if successful, empty string otherwise. For example

global hlist=cl.Create(clname)

would then allow calls like mylist.insert(clindex). Examples of this approach are shown in the syntax descriptions below.

cl.Get(clname)

Returns handle to existing clname which can be used to call other cl services if successful, empty string otherwise.

cl.Exists(clname)

Returns 1 if the specified command list exists, and the empty string otherwise.

cl.Remove(clname, clindex) or hlist.Remove(clindex)

Removes the clindex item of the specified command list. Returns a handle if successful (i.e. the element exists), and the empty string otherwise.

local anItem=cl.insert("list1", 0) or local anItem = hlist.insert(0)

Inserts a new item at the start of the list; assigns resulting handle to anItem.

cl.insert("list2", cl.length("list2"))

Inserts a new item at the end of the list.

cl.insert("thelist", "theId")

Inserts a new item before the one with id theId.

Note: If you remove or insert an item for a command list which is showing as a bar, the bar will be closed and then re-opened after the command completes. If you are changing many items, you may want to close the bar first with bar.close, then add/remove the items, then show the bar again with bar.show; this avoids many closes and shows of a bar.

Insert returns a handle which can be used to call services on the created item in its command list. The first example above sets anItem so that one can say, for example,

anItem.addLeftCmd("command", "params")

as a short form for

cl.addLeftCmd("thelist", 0,"command", "params")

Note that the handle returned by insert includes the absolute index of the item, so it will point to a different item if a new item is added to the command list before the one selected by the handle.

cl.RemoveAll(clname) or hlist.RemoveAll

Removes all the items in the command list. Closes any bar built from the clname. Does not destroy the list; just sets its length to zero. There is no cl.destroy function.

getting properties of command lists

cl.Length(clname) or hlist.Length

Returns the number of elements in the command list. Note that the index of the last element would be cl.length(clname)-1.

cl.GetLabelIndex(clname, string) or hlist.GetLabelIndex(string)

Returns the numeric index, which could be zero, of the item in clname with label equal to string. Returns the empty string if the label is not found.

cl.GetIdIndex(clname, string) or hlist.GetIdIndex(string)

Returns the numeric index, which could be zero, of the item in clname with id equal to string. Returns the empty string if the id is not found.

cl.GetItem(clname, string) or hlist.GetItem(string)

Returns a handle which can be used to call services on cl items (i.e. those with first two parameters clname, clIndex). The string can be either an id or a numeric index (but not a label). If it is a numeric index, the handle includes the absolute index of the item, so it will point to a different item if a new item is added to the command list before the one selected by the handle. This restriction does not apply if it is an id. Example:

```
anItem = aList.GetItem("id")   // aList is a handle to a command list set by create or get
anItem.addLeftCmd("command", "params")
```

cl.getBarHandle(clname) or hlist.GetBarHandle

Gets window handle for showing bar. This can then be used with win plugin.

cl.GetBackground(clname) or hlist.GetBackground

Returns the background file name.

cl.GetBackColor(clname) or hlist.GetBackColor

cl.GetHoverColor(clname) or hlist.GetHoverColor

cl.GetPressColor(clname) or hlist.GetPressColor

Returns the list background color for ordinary, hovered-over, and pressed buttons as three values separated by spaces: Red, Green Blue. If the list does not have its own color, returns the empty string.

cl.GetTextColor(clname) or hlist.GetTextColor

cl.GetHoverTextColor(clname) or hlist.GetHoverTextColor

cl.GetPressTextColor(clname) or hlist.GetPressTextColor

Returns the list text color for ordinary, hovered-over, and pressed buttons as three values separated by spaces: Red, Green Blue. If the list does not have its own color, returns the empty string.

cl.GetRoundBar(clname) or hlist.GetRoundBar

cl.GetRoundWhich(clname) or hlist.GetRoundWhich

cl.GetRoundBtn(clname) or hlist.GetRoundBtn

Returns the rounding for bars or buttons.

cl.GetHoverGradient(clname) or hlist.GetHoverGradient

cl.GetPressGradient(clname) or hlist.GetPressGradient

Returns the gradient settings as integer 0-7 representing selection index in configuration dropdown.

cl.GetHideAfter(clname) or hlist.GetHideAfter

Returns the interval for hiding a list.

cl.GetMaxtext(clname) or hlist.GetMaxText

Returns the maximum text length setting for a command list.

cl.GetTransparent(clname) or hlist.GetTransparent

Returns the transparency setting 0 – 255.

cl.GetGradient(clname) or hlist.GetGradient

Returns the gradient setting 0 – 255.

cl.GetHeight(clname) or hlist.GetHeight

Returns the default button height.

cl.GetHOffset(clname) or hlist.GetHOffset

Returns the horizontal offset.

cl.GetVOffset(clname) or hlist.GetVOffset

Returns the vertical offset.

cl.GetIconSize(clname) or hlist.GetIconSize

Returns the icon size, 0 for none.

cl.GetFolderButtons(clname) or hlist.GetFolderButtons

Returns the folder buttons file.

cl.GetNumActive(clname) or hlist.GetNumActive

Returns the maximum number of active buttons.

cl.GetPosition(clname) or hlist.GetPosition

Returns the position name using the same text values as the [Properties] section of the text configuration ini files, described in the section on command list properties.

cl.GetShowIfBump(clname) or hlist.GetShowIfBump

Returns the show-if-bump edge name using the same text values as the [Properties] section of the text configuration ini files, described in the section on command list properties.

cl.GetSlideFrom(clname) or hlist.GetSlideFrom

Returns the slide-from direction using the same text values as the [Properties] section of the text configuration ini files, described in the section on command list properties.

cl.GetClFlag(clname, flagname) or hlist.GetClFlag(flagname)

Returns 1 or 0 indicating whether or not the flagname is set for the configuration; flags are keywords such as "ToolTips" or "BarSize" the full list is given in the [Properties] section of the text configuration ini files, described in the section on command list properties.

setting properties of command lists

All of the setting services return a handle to the command list being set. This allows service calls to be chained, as in the following example

hlist.SetBackground(path, flag).SetHideAfter(value).AddProperties(props)

Some properties have specific setter calls; any property can also be set with NewProperties or AddProperties.

cl.SetHideAfter(clname,value) or hlist.SetHideAfter(value)

Sets the interval for hiding a list. Use 0 to stop hide. Use -n to alternate between n and no hide.

cl.LockTo(clname, onoff, clnameTarget, n1, n2, n3, n4, n5, n6, n7, n8)

If onoff is 1, locks clname to clnameTarget using eight position values n1 through n8 to specify locking points on target bar and source bar. n1 through n4 specify horizontal percentage, horizontal offset, vertical percentage, vertical offset for target (clnameTarget); n5 through n8 similar values for locked bar (clname). If onoff is 0, any existing lock is removed. In addition, all locks added by cl.LockTo are lost if bar is reconfigured using PProConf. Note that neither clname nor clnameTarget need exist when this command is issued.

cl.SetBackground(clname, path, flag) or hlist.SetBackground(path, flag)

Sets the background to the path. If flag is included and set to 1, does not close and re-open bar.

cl.SetAsTray(clname) or hlist.SetAsTray

Sets clname as the source for tray icons.

cl.SetAsAutorun(clname) or hlist.SetAsAutorun

Sets clname as the source for autorun command list.

cl.Refresh(clname) or hlist.Refresh

If clname is a bar, redraws it by opening and closing the bar. May be needed after NewProperties used.

cl.Redisplay(clname) or hlist.Redisplay

If clname is a bar, redraws it without opening and closing the bar. May be needed after NewProperties used.

cl.SetMaxtext(clname, ,newmax) or hlist.SetMaxText(newmax)

Sets the maximum number of characters shown on each button or menu item

cl.SetRect(clname, top ,left ,right, bottom) or hlist.SetRect(top, left, right, bottom)

Sets position for command list displayed as floating or locked bar. All four numbers must be specified. If command list is already showing as a bar, it is moved or resized (or both).

cl.NewProperties(clname, proplines) or hlist.NewProperties(proplines)

Replaces properties for clname with those given in proplines, which should be a multi-line string (using \r as the line separator) following the same format as the [Properties] section of the text configuration ini files, described in the section on command list properties.

Examples of cl.NewProperties

cl.NewProperties("thisone", "Background=c/a/file/path")

cl.NewProperties("another", "Format1=showasbar barsize \r active=20")

Note: If clname is a currently visible bar, it is closed and then reshown.

Note: If you are setting the properties for a newly created bar with floating position, you must always specify all four of Left:, Right:, Bottom:, Top:, even if BarSize is specified. You can use SetRect service as another way to do this.

Each time you call NewProperties all properties are reset. So, if you are making a series of calls in sequence to set many properties, the first one will normally be NewProperties and the remainder AddProperties.

cl.AddProperties(clname, proplines) or hlist.AddProperties(proplines)

Same as cl.NewProperties, except that properties are added to those already existing for the list. If you want to change the value of a keyword that it is not normally followed by a value, like ToolTips or TopMost, you can precede the keyword by ~ to clear the setting or by ^ to reverse it. For example

cl.AddProperties(clname, "^ToolTips") // reverses setting of tool tips flag; format=assumed

cl.CopyProperties(clnameFrom, clnameTo) or hlist.CopyProperties(proplines)

Copies all properties from first cl to the second.

working with configuration files

cl.Compact()

Compacts PowerPro internal memory used to store strings related to command lists. It is never necessary to use this function, but it may help to keep your memory usage down if you add or remove many items, or change the value of many commands, names, icons, tool tips. PowerPro automatically does a compact each time you remove all items in a command list or after importing a command list.

cl.Savepcf()

Writes current pcf, ensuring any changes to command lists from the original pcf are saved. Only needed if you want to ensure that changes made to command lists stored in pcf are saved. If this function is not used, the changes may or may not be saved.

This function only saves changes to command lists that appear in the original pcf. To save manually created lists or lists from text imports as well, use Configure.WriteAllToPCF(path_to_pcf).

cl.Import(filepath)

Imports command list(s) stored in filepath which much be in ini file format. Has same effect as Configure.ImportCL(filepath).

cl.Export(clname,filepath,appendflag)

Exports clname to filepath. If appendflag is present and set to 1, appends to end of file. Otherwise, overwrites file.

cl.TaskbarBarUpdate(flag)

If flag is 0, stops automatic updates of Taskbar bars from cl functions. If 1, restarts (and does an immediate update). If –1, reverses previous setting.

working with bars and menus

cl.ShowMenu(clname, pos) or hlist.ShowMenu(pos)

Show command list as menu. The pos argument can be omitted or set to any of the menu positions screen top left, centreundermouse, centerscreen, cursor.

cl.getBarHandle(clname) or hlist.GetBarHandle

Gets window handle for showing bar. This handle can then be used with win plugin.

cl.Show(clname) or hlist.Show

Show command list as bar.

cl.HideShow(clname) or hlist.HideShow

Reverses hidden/shown status of bar.

cl.Hide(clname) or hlist.Hide

Hides bar.

cl.Close(clname) or hlist.Close

Closes bar.

cl.Keys(clname) or hlist.Keys

Readies bar to receive keys.

cl.ToMouse(clname) or hlist.ToMouse

Moves bar to mouse. Shows it if not already showing.

cl.ToMouse(clname) or hlist.ToMouse

Moves bar to mouse. Shows it if not already showing.

cl.SelectSubbar(clname,"@subbar") or hlist.ToMouse("@subbar")

Shows named subbar; the "@" must be present. Services SelectSubbarToMouse and SelectSubbarToButton also exist.

cl.GetLastBar()

Returns name of bar where last button was pressed; returns "" if no bar button yet pressed.

cl.GetLastPressed("barname")

Returns the last pressed button.

cl.GetLastPressedType("barname")

Returns "P", "D", or "H" depending on whether the last accessed button was pressed, dropped to, or hovered over.

cl..getLastMouse("barname")

Returns "L","R", or "M" to indicate which mouse button pressed the last button on bar "barname".

cl.GetLastMenu()

Returns name of command list last displayed as menu

cl.GetLastMenuSel()

Returns index of last command list item selected from last displayed menu; returns "" if nothing selected.

cl.GetLastMenuMouse()

Returns "R" if right command selected from last menu; "L" otherwise.

Configuring with CL functions

CL functions for getting item attributes

In each description, clname is used as an expression to represent a command list name, clitem is used to represent an expression giving the integer index of an item or its id, and hitem is a handle to an item returned by cl.insert or cl.getitem.

You can use the following three services to translate between item index number, item ids, and item labels. However, note that you *can always use either an id or an index number for* clitem.

working with item label, ids, and index numbers

cl.GetLabelIndex(clname, string) or hlist.GetLabelIndex(string)

Returns the numeric index, which could be zero, of the item in clname with label equal to string. Returns the empty string if the label is not found.

cl.GetIdIndex(clname, idstring) or hlist.GetIdIndex(idstring)

Returns the numeric index, which could be zero, of the item in clname with id equal to idstring. Returns the empty string if the id is not found.

hitem=cl.GetItem(clname, clitem) or hitem=hlist.GetItem(clitem)

Returns a handle which can be used to call services on cl items (i.e. those with first two parameters clname, clIndex). The string can be either an id or a numeric index (but not a label). If it is a numeric index, the handle includes the absolute index of the item, so it will point to a different item if a new item is added to the command list before the one selected by the handle. This restriction does not apply if it is an id. Example:

```
anItem = aList.GetItem("id")   // aList is a handle to a command list set by create or get
anItem.addLeftCmd("command", "params")
```

Samples for hitem are only shown for the first two services but this format can be used in all cases.

working with item command entry elements

cl.RunLeft(clname, clitem) or hitem.RunLeft

Runs the left command of item clitem. Use cl.RunRight for right command and cl.RunMiddle for middle command.

cl.GetLeftCmd(clname, clitem, i) or hitem.GetLeftCmd(i)

Returns the ith command name from the Left Command of the specified command list. Returns the first command if i is 0 or 1, the second if it is two, and so on. You use cl.GetRightCmd for the Right command and cl.GetMiddleCmd for the middle command. Example:

cl.GetLeftCmd("mybar", 13, 1)

returns the first command.

cl.GetLeftParam(clname, clitem, i)

Returns the ith command parameter from the Left Command of the specified command list. Returns the first if i is 0 or 1, the second if it is two, and so on. You use cl.GetRightParam for the Right command and cl.GetMiddleParam for the middle command.

cl.GetLeftWork(clname, clitem, i)

Returns the ith work directory from the Left Command of the specified command list. Returns the first if i is 0 or 1, the second if it is two, and so on. You use cl.GetRightWork for the Right command and cl.GetMiddleWork for the middle command.

cl.GetLeftHow(clname, clitem, i)
Returns the ith how start from the Left Command of the specified command list. Returns the first if i is 0 or 1, the second if it is two, and so on. Possible return values are "normal", "max", "min", "hide", "traymin". You use cl.GetRightHow for the Right command and cl.GetMiddleHow for the middle command.

cl.GetLeftOnTop(clname, clitem)
Returns setting of on top checkbox for left command ("1" if set, "" otherwise). Use cl.GetMiddleOnTop, cl.GetRightOnTop for others.

cl.GetLeftSwitchTo(clname, clitem)
Returns setting of switch to if active checkbox for left command ("1" if set, "" otherwise). Use cl.GetMiddleSwitchTo, cl.GetRightSwitchTo for others.

working with other item properties

cl.GetId(clname, clitem)
Returns the id from the item clitem.

cl.GetLabel(clname, clitem)
Returns the label from the item clitem.

cl.GetLabelValue(clname, clitem)
Returns the label from the specified item; if it is an *info label, then the current actual value is returned.

cl.GetTooltip(clname, clitem)
Returns the tool tip from the item clitem.

cl.GetTooltipValue(clname, clitem)
Returns the tool tip from the specified item; if it is an *info tool tip, then the current actual value is returned.

cl.GetIconFile(clname, clitem)
Returns the Icon file name from the item clitem.

cl.GetIconNumber(clname, clitem)
Returns the zero-based icon number from the item clitem.

cl.IsVisible(clname, clitem)
Returns 1 if item is visible, 0 otherwise.

cl.IsEnabled(clname, clitem)
Returns 1 if item is enabled, 0 otherwise.

cl.IsIconVisible(clname, clitem)
Returns 1 if icon is visible, 0 otherwise.

cl.GetWidth(clname, clitem)
Returns item width.

cl.GetHeight(clname, clitem)
Returns item height.

cl.GetBackColor(clname, clitem)
Returns the color as three values separated by spaces: Red, Green Blue for the background of the item. If the item is not showing its own color, returns the empty string.

cl.GetTextColor(clname, clitem)
Returns the color as three values separated by spaces: Red, Green Blue for the text of the item. If the text is not showing its own color, returns the empty string.

cl.GetCtrlHandle(clname, clitem)
Returns the window handle of a control associated with the button on a toolbar and defined using *control You can use win.sendmessage to interact with the control using this handle.

cl.GetCtrlValue(clname, clitem)
Returns the text displayed by the control; returns 0, 1, 2 for checkboxes.

cl.GetFontName(clname, clitem)
Returns the font name if item has own font; returns "" otherwise.

See Also:

Overview of cl functions

Getting single item attributes

Setting single item attributes

Configuring with CL functions

CL functions for setting item attributes

In each description, clname is used as an expression to represent a command list name and clindex is used to represent an expression giving the integer index of an item or its id, hitem is a handle to an item returned by cl.insert or cl.getitem.

Samples for hitem are only shown for the first two services but this format can be used in all cases.

All of the following set services return a handle to the set item allowing the service calls to be chained, as in this example.

hlist.AddLeft(command, parameters, work, how).setLabel("label").setid("id")

When setting the attributes of an item, the specified item must always exist in the list. If you want to add a new item, use cl.Insert(name, cl.length(name)) to add an item at the end, if needed, before setting its attributes.

Tip: If you make an item visible or invisible for a command list which is showing as a bar, or change an item height or width, the bar will be closed and then re-opened after the command completes. If you are changing many items, you may want to close the bar first with hlist.close, then change the items, then show the bar again with hlist.show; this avoids many closes and shows of a bar.

cl.ClearLeft(clname, clindex) or hItem.ClearLeft
Clears left command information. Use cl.ClearMiddle and cl.ClearRight to clear other command information.

cl.AddLeft(clname, clindex, command, parameters, work, how)

or hitem.AddLeft(command, parameters, work, how)
Adds the specified command to the commands already present for command list clname, item clindex. The command, parameters, work, and how parameters are all optional, except that the command must be specified if any of the others are specified. The clindex must be less than the length. If there is no room for the command, null string is returned; otherwise "1" is returned.
Or use cl.AddRight for right command, cl.AddMiddle for middle.

Example of cl.AddLeft

cl.AddMiddle("mylist", 1, "Menu", "Show mymenu", "centerscreen")

adds "Menu Show mymenu" with *Work Folder* used for the option "centerscreen".

cl.SetLeftOnTop(clname, clindex, newval)
Sets OnTop checkbox for left command.
Or use cl.SetMiddleOnTop, cl.SetRightOnTop for others.

cl.SetLeftSwitchTo(clname, clindex, newval)
Sets SwitchTo checkbox for left command).
Or use cl.SetMiddleSwitchTo, cl.SetRightSwitchTo for others.

cl.SetLabel(clname, clindex, newlabel)
Sets the label for the specified item. You can change *info labels, and you can change labels to and from being *info labels.

cl.SetId(clname, clindex, newid)
Sets the id for the specified item.

cl.SetTooltip(clname, clindex, newtip)
Sets the tool tip for the specified item. You can change *info tool tips, and you can change tool tips to and from being *info tips.

cl.SetIcon(clname, clindex, path, number)
Sets the icon to the specified path and (zero-based) number. You can choose a .image file for icon in which case the whole button face will be tiled with the bitmap in the file.

cl.SetVisible(clname, clindex, vis)
If vis is zero or empty, makes the specified item invisible, otherwise makes it visible.

cl.SetEnabled(clname, clindex,enable)
If enable is zero or empty, makes the specified item disabled, otherwise makes it enabled.

cl.SetIconVisible(clname, clindex, ico)
If ico is zero or empty, makes the icon from the specified item invisible, otherwise makes it visible.

cl.SetMultiline(clname, ,clindex, flag)
Sets multi-line setting for button label according to flag.

cl.SetWidth(clname, clindex, newwidth)
Sets the item width.

cl.SetHeight(clname, clindex, newheight)
Sets the item height.

cl.SetHover(clname, clindex, newhover)
If newhover is 0, sets to no hover click, if 2, sets to middle, if 3 sets to right, else sets to left.

cl.SetBackColor(clname, clindex, ColorParameters)
Sets the item's background color (only affects buttons, not menu items)

cl.SetTextColor(clname, clindex, ColorParameters)
Sets the item's test color (only affects buttons, not menu items)

There are five ways to specify the **ColorParameters** in cl.SetBackColor and cl.SetTextColor:

- specify the red, green, and blue components as three separate parameters between commas;

- specify the red, green, and blue components in a single blank-separated parameter string;

- specify the red, green, and blue components as a single integer with red in the lowest 8 bits (as returned by win.getpixel and inputcolor);

- specify "current" to use the currently set color (presumably instead of the bar color).

- leave the ColorParameter blank to use the bar's background color

Numbers are decimal unless they start with 0x in which case they are taken to be hexadecimal.

Examples of ColorParameters:
All of the following specify the same color for item number 5, with red=4, green=128, blue=255:

```
cl.SetTextColor(mybar, 5, 4, 128, 0xff)
cl.SetTextColor(mybar, 5, "4 0x80 255")
cl.SetTextColor(mybar, 5, 4 + 256*(128+256*255))
cl.SetTextColor(mybar, 5, 0xff8004)
```

For both SetBackColor and SetTextColor: if the third argument is omitted or empty (""), the button reverts to the bar background color or text color. If the third argument is "c", the button uses is own background color or text color as previously set.

cl.SetFont(clname, clindex, fontname, fontsize)
cl.SetFont(clname, clindex, fontname, fontinfo)
Sets the font as specified. If fontname is "", the own font is removed. Use a plain number fontsize to specify just the font size; use a fontinfo string to specify information in the same format as the .ini file fontinfo= keyword and values.

cl.SetCtrlValue(clname, clindex, newtext)
Sets the text displayed by the control; use 0, 1, 2 for checkboxes.

cl.SetCtrlURL(clname, clindex, url)
If an ie control is attached to the specified button, it is navigated to the specified URL. Use "+" for URL to go forward, "-" to go back.

cl.AddCtrlListItem(clname, clindex, "text", pos)
Adds the specified text at position pos of the combobox control dropdown. Omit pos or set to –1 to add to end.

cl.ClearCtrlList(clname, clindex)
Removes all items in the combobox control dropdown.

See Also:

Overview of cl functions

Getting single item attributes

Setting single item attributes

Built in Commands
PowerPro built in commands

Powerpro comes with a set of built in commands which let you manipulate running programs and control the Windows interface. You find the built in commands in the Command drop down control of the Powerpro command entry controls command entry controls.

After choosing a built-in command, the command entry controls dialog automatically prompts you for the actions and information required for each built-in command.

Following are the built-in commands:

*Bar	Work with bars.
*Clip	Clipboard extender and history.
*Configure	Access configuration dialog.
*Desktop	Work with desktop icons, Taskbar, resolution, windows.
*Exec	Miscellaneous commands
*File	Move, copy, delete files.
*Format	Change look and layout of menus and bars.
*Keys	Send keystrokes to other windows
*Macro	Used only as a hotkey command to setup a list of keyboard macros.
*Menu	Display menus built from command lists, folder contents.
*Message	Display a message.
*Mouse	Send a series of mouse actions to another window.
*Note	Use sticky notes
*ScreenSaver	Start, stop, enable, disable, change the screen saver.
*Script	Runs a list of commands.
*Shutdown	Shutdown Windows or PowerPro
*Timer	Start, stop, clear, reset PowerPro timers.
*TrayIcon	Activate or hide tray icons of other programs.
*Vdesk	Work with virtual desktops.
*Wait	Wait for an event or a certain amount of time.
*Wallpaper	Change the wallpaper (desktop background).
*Window	Close, min, max, traymin, rollup, etc any window.

In addition to these built-in commands, many of the plugin calls act as commands; so do many of the cl.service calls.

This page describes the alternative syntax formats for running commands, either using literal text for the command's arguments, or using expressions for the arguments.

Built in Commands : *Bar
Bars and the *Bar command

You can access the items in a command list as buttons on a bar.

Some features of PowerPro bars:

You can use left, middle, and right clicking to execute different commands.

Configure bar positions and other features by Ctrl-right clicking the bar.

Move bars by left dragging (position must be floating or caption/beside active). Optionally you can require that Ctrl be pressed to drag a bar.

Bar size can be set to sum of buttons or can be manually set using a sizing border (*"Bar size to sum of buttons"* must be unchecked to use border).

Bars can be automatically hidden when the mouse pointer is moved off them

Bars can use skinsskins.

Bars can be positioned in the caption of or beside/above/below the active window

Bars can be positioned as screen bars which reserve screen space like the Windows Taskbar.

One bar can be positioned on the Windows Taskbar.

Bar buttons can be pressed using the keyboard

Bar visibility can be based on the active window

Files can be drag/dropped onto bar buttons from explorer; left drag/drop starts a button command and right drag/drop allows buttons to be configured.

Different parts of bars can be shown using subbars.

configuring bars

Bars are configured using command lists and their look is set using Bar Properties. You can also ctrl-right click on a bar to access many of the configuration options from a menu. To show bars automatically, check *"Auto Show as Bar"* on the Command Lists dialog; bars can also be shown manually using the *Bar show command.

You can put buttons on bars in various ways:

- as Items in a command list,
- files and sub folders from a specified folders,
- a button for each windows, useful for task switching,
- a button for each icon in the tray.

dragging a bar

If the bar's position is *floating* or *caption* or *near active window*, you can move the bar by clicking and dragging any button. If you find that you are moving the bar when you do not want to, you can use the Command Lists Setup dialog to require that ctrl be down for dragging a bar to move it.

You can also position bars by assigning the *Format Drag command to any button and then clicking and dragging that button. After moving a fixed position bar, you can return to the fixed position quickly by Ctrl-right clicking bar and selecting "Last Fixed".

separators on bars

You can force new rows on non-vertical bars with the *Format NewBarRow command. You can start a new row and show a horizontal separator line with *Format NewBarRowLine. You can insert a vertical separator line with *Format BarVerticalLine.

using the *Bar command

*Bar Show barname

Shows a bar. barname is the name of the command list

*Bar Show *keyword barname

Show a bar with slide animation (W98/2000). *keyword is one of:
*vertical, *horizontal, *fromtop, *frombottom, *fromleft, *fromright
(see below for details of *Bar Show keywords for animation)

*Bar Show *move barname

Shows a bar and positions the mouse over the bar.

*Bar Hide barname

Hides a bar but keeps it in memory (for faster reshow).

*Bar HideShow barname

Hides a bar if visible; shows it otherwise.

*Bar Close barname

Closes a bar and removes it from memory.

*Bar ToMouse barname

Temporarily moves bar to mouse and shows it if hidden.
Usually used with hotkey. Floating position only.

*Bar ToCursorbarname

Temporarily moves bar to text cursor and shows it if hidden.
Usually used with hotkey. Floating position only.

*Bar Keys barname

Readies bar to receive keys.

*Bar ToButton number barname

Readies bar to receive keys and moves mouse cursor to given button number.

*Bar SelectSubBar barname

Shows a subbar.

*Bar SelectSubBarToMouse barname

Shows a subbar at the mouse

*Bar SelectSubBarToButton barname

Shows a subbar aligned to the bar button just pressed.
Also see s = subbarname which returns name of current subbar

*Bar Format barname keyword

Changes bar background, hide interval, or position.
The resulting new bar configuration is always saved in the .pcf file.

barname is the name of the bar's Command List

keyword is one of the following:

back "file.bmp" changes the background to file.bmp Or use:
back none to remove background

back2 "file.bmp" changes the background to file.bmp Or use:
back2 none to remove background
Use both back and back2 to alternate between two backgrounds each time the command is executed.

autohide n Changes the interval before automatic hiding to n millisecs.
Use 0 for no autohide. Use -n to alternate between no autohide and autohide after n milliseconds.
Example: autohide -1000 alternates between no autohide and hiding after 1000 milliseconds.

position n Sets the bar position to the nth position. Use the menu accessible from the button beside the keyword edit box to set the number. Use a negative number to alternate between floating position and the nth position.

refresh Closes and reopens bar. Could be used, for example, to manually refresh folder buttons.

animating with *Bar Show parameters

For Win98/2000 and later, when showing a hidden bar, you can specify that slide animation be used by preceding the bar name with one of *vertical, *horizontal, *fromtop, *frombottom, *fromleft, or *fromright. For *vertical and *horizontal, Powerpro will select the direction depending on which half of the screen the mouse is positioned. The *Command Lists >Setup* checkbox *"use slide animation"* does not have to be checked. If it is, you can also use *none to override any default slide animation.

examples of animation

*Bar

Show

*vertical MyBar

For *Bar Show and *Bar HideShow, you can also request that the mouse be moved to be position over the bar by preceding the bar name with *move, e.g.

*Bar

Show

*move mycommandlist

This can be used to prevent an autohide bar from disappearing when it is shown if the mouse is not over it.

You can combine slide hints like *fromtop and *move using the words in either order.

You can also use the *Window commands hide, hideshow, show, position, and close to access a bar. Use the bar name as the caption to access a single bar or use c=PowerProToolBar as the class name caption to access all bars.

new syntax

Like most built in *commands, the *Bar command now accepts the new alternative syntax which is similar to plugin calls

Instead of:	*Bar Show *keyword actualBarname
you can use:	bar.show("*keyword", barnameExpr)

which requires an expression for barnameExpr or a literal bar name in "quotes".

Built in Commands : *Bar

*Bar format command

Use *Bar format to change the background, autohide interval, or position of a bar.

To configure the command, select the *Bar command, Format action, and set the command list drop down to the bar to be changed. Then enter format keywords to specify the new bar attributes. Use the button at the right of the format keywords dialog to select the keywords. The keywords are:

back "file.bmp"	changes the background to file.bmp or use back none to remove background; put file name in double quotes if it contains blanks.
back2 "file.bmp"	changes the background to file.bmp or use back none to remove background; put file path in blanks if it contains blanks. Use both back and back2 to alternate between two backgrounds each time the command is executed.
autohide n	changes the interval before automatic hiding to the number n. Use 0 for no autohide. Use -n to alternate between no autohide and autohide after n

	milliseconds. For example, autohide -1000 alternatives between no autohide and hiding after 1000 milliseconds.
position n	Sets the bar position to the nth position. Use the menu accessible from the button beside the keyword edit box to set the number. Use a negative number to alternate between floating position and the nth position.
refresh	Closes and reopens bar. Could be used, for example, to manually refresh folder buttons.

The resulting new bar configuration is always saved in the .pcf file.

Example:

```
*Bar Format xxx back2 "*" back none position -1
```

alternates bar xxx between wallpaper and no background and between floating and locked position.

Background and Button Images

Overview

If your system has gdiplus.dll installed, PowerPro supports many image files types to be used for button images and bar and menu backgrounds. Supported images types include BMP, PNG, JPEG, GIF, TIFF. Gdiplus comes with Windows XP; it can also be used on earlier versions of Windows. You can download a copy from the files section of http://groups.yahoo.com/group/power-pro if you do not have one.

Since PNG files support blending into the background, it is important to understand how PowerPro draws images on bars and buttons to understand how your bar will look. The following sections explain this. Two cases are covered: depending on whether or not the bar is set to be partially transparent using the Transparent setting on the Command List properties for the bar.

No Transparency

If the transparency is zero (ie no blending with background windows under the bar), PowerPro follows the following steps to draw the bar:

1. Fill the bar with the background color. If there is no background specified, use the system default.

2. If there is a background image, blend it with the background.

3. For each button:

3a: If the button has a background color, fill the button rectangle with that color. Otherwise do nothing, which leaves whatever resulted from the first two steps in the button rectangle.

3b: If there is a button image, blend it with whatever is in the button rectangle from the above steps.

3c: Write any button text on the button rectangle.

Transparency

If transparency is greater than zero, then the way PowerPro draws the bar depends on the Pixel-Blending checkbox:

If Pixel Blending is unchecked, first the bar is drawn as under No Transparency. Then the transparency value is used. If it is between 1 and 254, the bar is made partially transparent, with greater values leading to greater transparency. If it is 255, then background color of the bar is set to be transparent.

If Pixel Blending is checked, then any bar background color or gradient is ignored. Each button bitmap, text, and icon is drawn over any bar background. Then the resulting image is pixel blended into the underlying windows. If the transparency value is 1, it is blended as is; if it is between 2 and 254, further transparency is added; if it is 255, then the bar background color only is made transparent and no pixel blending is done.

If Pixel Blending is gray-checked, then drawing is the same as ordinary check, except that button images are blended into the bar background image, rather than replacing that image.

Note that the mouse will always click through any transparent portions of a drawn bar.

For pixel-blended bars, no 3d frame is drawn, but any non-transparent portions of the bar border can still be used to size the bar.

Built in Commands : *Clip

*Clip command

clipboard manipulation, tracking and copying

purpose

The *Clip command is used to work with the clipboard. Powerpro has commands to copy text or a file to the clipboard, to copy the clipboard to a file, and to clear the clipboard.

Furthermore, Powerpro has a clipboard history function: it can track text as you Copy it to the clipboard and can subsequently display a list of such text items on a Clip Menu. If you select an item from this menu, the selected item is recopied to the clipboard and optionally pasted.

The built-in clipboard commands work with the first line of text on the clipboard only. The clip plugin, described below, can work with the full clipboard.

*Clip actions

MenuPaste	Show menu of recently captured clips history; select an entry to put on clipboard and then automatically paste selected entry using Ctrl-V.
MenuIPaste	Show menu then paste selected entry using Ctrl-Ins. (Note: this keyword contains an added letter i)
Menu	Show menu ; select an entry to put on clipboard.
Delete	Delete selected entry from clipboard history.
File	Copy file to clipboard.
FilePaste	Copy file to clipboard then paste using Ctrl-V.
FileIPaste	Copy file to clipboard then paste using Ctrl-Ins
CopyThenToFile	Send Ctrl-C then copy clipboard to file. See Manual Copy below for more
ToFile	Copy clipboard to text file.
ToFileAppend	Append clipboard to text file.
ClearClipboard	Clears clip board.
ClearRecent	Clear recent list of captured clips.
ShortDate	Put date on clipboard in short format. (put *Keys ^v in More Commands to paste).
LongDate	Put date on clipboard in long format.
Time	Put time on clipboard.
Text	Put following text on clipboard. Single line of text only.
TextAppend	Append following text to clipboard. If no text, a new line is appended.
TextPaste	Put entered text on clipboard and then automatically paste with Ctrl-V.

Note: *Clip TextPaste text to paste will often be faster than a *Keys command for long text.

Copy	Sends Ctrl-C to foreground window for copy ; normally follow by wait.for.
Cut	Sends Ctrl-X to foreground window for cut ; normally follow by wait.for.
Paste	Sends Ctrl-V to foreground window for paste ; normally follow by wait.for.

Reattach	Puts PowerPro at front of applications tracking clipboard

Note: The clip actions Text, TextAppend, and TextPaste commands can only work with a single line of text. Use the Clip plugin, described below, to work with multiple lines of text.

commands relating to the clipboard history tracking features

Clip Capture	follow by off, on, or reverse to change the current state. You can follow on by a folder name, say c:\path, to start capturing clips in folder c:\path. You can replace the foldername by the word refresh (i.e. use *Clip capture on refresh) to have PowerPro reset its internal memory copy of clip files and dates; do this if you insert a new text file into the clip folder without using clip capture.
Clip write off	follow by off, on, or reverse to change the output of clips to a file. You would turn off writing if you had implemented your own clip management by using the clipcaptured command list.

manual copy to clipboard

You can manually copy selected text to the clipboard and then to a specified file with

Command	*Clip
Parameter	CopyThenToFile filepath

Powerpro will send the keystroke Ctrl-c to copy selected text to the clipboard, and then will copy the clipboard text to the filepath. Provide the full filepath with the extension. PowerPro will always copy plain text format.

To implement multiple clipboards, create a set of hot key pairs. For each pair, one hotkey has a Copy and Clip.ToFile; the other hotkey has a Clip.FromFile and Paste, using the same file name assigned to the other hot key.

clip plugin

The clip plugin extends the built-in clip commands and offers the ability to work with multi-line text on clipboard as well as other functions. You must use the new syntax clip.service(arguments) to access the functions of the plugin. Following is a list of those functions.

Get - returns the text on the clipboard. Example, var = clip.get.

GetRTF - returns rtf text on clipboard (as PowerPro text string)

GetHTML - returns HTML format on clipboard (as PowerPro text string)

HasFormat("xxx") - The argument "xxx" can be a clipboard text standard name (see below) or an integer; returns 1 if the specified format is available on the clipboard. Example:

```
if (clip.hasformat("html"))
  varhtml = clip.gethtml
```

ListFormats - Returns a blank-separated list of the formats on the clipboard. May include names listed below as well as integers for private clipboard formats.

SetSaveLoadFolder(folderpath) - Sets default folder for clip.save and clip.load. If folderpath omitted, default is set to system temporary folder, which is also what it is initially.

GetSaveLoadFolder() - Gets default folder for clip.save and clip.load.

Save(filename, folderpath) - Saves contents of clipboard in all formats to file named filename in folder folderpath. The folderpath parameter may be omitted, in which case the system temporary folder is used. The filename may be omitted, in which case "0.PowerProClip" is used. If the filename is present but without an extension, then "PowerProClip" is the extension.

Load(filename, folder) - Loads clipboard contents saved by clip.save. Same conventions apply to filename and folder as for save.

Set(var, cap) - sets the clipboard; example clip.set("line1\r\nline2"). If cap is present and set to 1, then this clip will be captured by PowerPro

SetHtml(var, plain, cap,onlyhtml) - sets the clipboard to the html text in var. A second variable plain can optionally be present to give plain text version of html. If cap is present and set to 1, then this clip will be captured by PowerPro. If onlyhtml is present and set to 1, only the html element of the clipboard is changed; the remaining elements are untouched.

Append(var) - appends text in var to the clipboard

Setpaste(var) - sets text to the clipboard, then sends Ctrl-V to paste to foreground window. Usually will need to be followed by wait.for((50) so that paste key Ctrl-v can be processed by receiving window.

Clear - clears the clipboard

Length - returns length of clipboard; example, lclip = clip.length

IgnoreNext - Next item added to clipboard will not be captured by PowerPro clip capture

FromFile(fpath) - copies contents of file to clipboard

FromFileAppend(fpath) - appends text in file to clipboard; does not support rich text format.

ToFile(fpath,NoErr) - puts clipboard into file fpath; no error message if NoErr is 1.

ToFileAppend(fpath,NoErr) - appends clipboard to file fpath; no error message if NoErr is 1; does not support rich text format.

BMPToFile(fpath) – writes the bitmap on the clipboard to the file path.

Standard Clipboard Formats

The standard clipboard formats that can be used with HasFormat or appear in ListFormats are:

"BITMAP"

"DIB"

"DIBV5"

"DIF"

"ENHMETAFILE"

"HDROP"

"LOCALE"

"METAFILEPICT"

"OEMTEXT" T

"PALETTE"

"RIFF"

"SYLK"

"TEXT"

"WAVE"

"TIFF"

"UNICODETEXT"

Built in Commands : *Desktop

*Desktop command

Use the *Desktop command to control various aspects of your desktop layout. Use these action keywords:

HideIcons	Hides all icons on desktop
ShowIcons	Shows desktop icons
HideShowIcons	Hides icons if visible; shows them if hidden.
HideTaskbar	Hides the Windows Taskbar
ShowTaskbar	Shows the Windows Taskbar
ShowTaskbarautohide	Shows the Taskbar; re-hides it when mouse is moved off and the Taskbar is not the foreground window. Assign this command to a **screen bump** hot key to show an Autohide Taskbar but prevent inadvertent shows by movements near screen bottom. You may also need *Desktop hideTaskbar as a startup scheduled event.
HideShowTaskbar	Hides Taskbar if visible; shows it if hidden.
HideShowWindows	Hides all desktop windows; shows them when next executed.
MinShowWindows	Minimizes all desktop windows; shows them when next executed.
SaveIcons	Saves desktop icon positions
RestoreIcons	Restores desktop icon positions
SaveIconsGrid	Aligns desktop icon positions to a grid and then saves them.
Resolution	Changes resolution
TransIconText	Makes a transparent background for desktop icon text; use keyword auto to reset transparency when background changes
IconTextColor	Set color of text under desktop icons; use the TransIconText auto function to reset color when background changes
SetWorkArea	Sets size of work area on desktop screen: that is the size of a maximized window. Specify four numbers left, top, bottom, right. Relies on system routine built into Windows which controls the effect of this command.

Built in Commands : *Desktop

Changing screen display resolution

You can change the display resolution, color depth, and refresh frequency (NT and later only) with the built-in *Desktop Resolution command.

If you use this command with nothing in the parameters edit box, Powerpro will present a menu of valid screen resolutions to choose from. Select one to change and save the new setting in the registry (hold down shift while selecting to avoid saving the new setting).

For Windows 95, if you change the color depth or refresh frequency, you will be asked if you want to restart windows for the settings to take effect.

To set a resolution without the menu, specify:

Command:	*Desktop Resolution
Parameters:	x1 y1 depth freq

where **x1** gives the new horizontal pixels, **y1** gives the new vertical pixels, **depth** gives the new color depth (4, 8, 16, 24), and **freq** gives the new refresh frequency (NT only). Depth and freq are optional. For example, to change to 1024 x 768:

Command:	*Desktop Resolution
Parameters:	1024 768

You can alternate between two settings by the following command format:

Command:	*Desktop Resolution
Parameters:	x1 y1 x2 y2

When this command is executed, the display resolution is set to x1 by y1 unless it is already that value; in this case it is set to x2 x y2.

Normally, the new settings are saved in the Registry; if you do no want this to happen put the word nosave after the settings in the parameter field.

Built in Commands : *Desktop

Saving and restoring desktop icon positions

Assign these commands to a button or menu, and execute them to save and restore your desktop icons positions.

*Desktop SaveIcons	Saves the relative positions of desktop icons
*Desktop RestoreIcons	Restores the relative positions of desktop icons
*Desktop SaveIconsGrid n1 n2 [filename.iconpos]	Aligns icons according to a grid and [optionally] saves them into c:\path\powerpro\filename.iconpos

Positions are stored as numbers which are independent of screen resolution. If you save positions under one resolution and restore under another, the relative positions of icons on your physical screen will not change.

You can align icons to a grid before saving by using SaveIconsGrid n1 n2 in the parameters box of the *Desktop command, where n1 is horizontal grid spacing and n2 is vertical grid spacing. The top left corners of icons are moved to the nearest grid point.

Example: *Desktop SaveIconsGrid 30 20 mydesk1.iconpos

You can specify the name of the file used to save/restore the icons by putting the file name in the Parameters edit box (after the grid numbers, if you are using them). Do not specify a path; all files must be in the Powerpro folder. Use the extension **.iconpos**. This allows many different configurations to be kept.

You can use the advanced dialog to specify that PowerPro should always restore saved desktop icon positions when the screen resolution is changed; however, this option may cause Explorer aborts on some systems.

Built in Commands : *Exec

*Exec commands

Note: Several of the following *Exec commands are described in more details in subsequent pages.

Use the *Exec commands to access various functions. Many of these commands require parameters which are explained in the pages linked to. The *Exec commands are:

Autoscroll	Starts automatic scrolling
AutoPress	Used to learn new types of windows for mouse stop-press
Alarms	Suspend or re-activate checks for scheduled programs.
BrowseRun	Shows a file open dialog; the select file is immediately executed.
Calendar	Shows a calendar. Use mouse or arrow keys to navigate. Unavailable on early Win95 versions unless IE3 or later has been installed.
CalcCalendar	Shows two dates/calendars along with day number, week number, and differences between dates. Changing any value updates the others. Unavailable on early Win95 versions unless IE3 or later has been installed.
CD	Play a CD track or eject any CD.
ChangeConfiguration filespec	Changes to configuration stored in a different pcf file; the new file path can be entered in the command or it will be prompted for if no path is provided.
ClearRecent	Clears recent command folder on Start Menu.
ClearRecentExplorer	Clears list of recent windows shown by *Menu Explorer.
CommandLine	Shows a tiny command line to enter a command to run.
ContextMenu	Shows right click menu of window under mouse; usually followed by *Keys {to menu}… to select an entry from the menu. Or you can specify a file or folder to get the context menu for that file. Finally, you can specify an object from Desktop or My Computer virtual folders, such as a local drive, to get its context menu. You must use the same name that appears for the object when you view it with Explorer.
Dos	Starts Dos, runs a command line, restarts Windows (not for NT).
EmptyRecycleBin	Empties the recycle bin. Use checkboxes to control confirmation and whether sound is played and animation shown. Unavailable on early Win95 versions unless IE4 or later has been installed.
Disable	Disables PowerPro until mouse is moved over a bar or a hot key is used.
Explorer	Shows the contents of the specified folder in explorer; both file folders and special folders like control panel may be used. Use find button to browse

	for folder when configuring. Use * for current working folder of active program.
Explorer2	Shows the contents of the specified folder in explorer in 2-pane window; both file folders and special folders like control panel may be used. Use find button to browse for folder when configuring. Use * for current working folder of active program.
FindFiles	Shows the Windows find files dialog. You can use the edit box to enter a single starting folder. Or use *Keys in More Commands to initialize dialog fields; for example, *Keys %lc:\path;d:\p2%n*.txt sets Look In to **c:\path;d:\p2** and Named to ***.txt**.
FindComputer	Shows find computer dialog.
HideWindow	Shows a dialog allowing you to pick a window to hide.
Hotkeys on / off / reverse	Suspend or re-activate hotkeys. Note: even if you suspend hot keys, a hot key which runs this command will still work so you can tie *Exec HotKey reverse to a hot key to control whether hotkeys are enabled. You can also specify a list of blank-separated hot key names after on/off/reverse to act only on those hot keys. The hot key name can be found by exporting the hot keys to an .ini file with Setup >Export; it is the same as the name which appears for the key in the pproconf.exe tabbed configuration dialog. You can also specify the hotkey by id=xxx in the Exec Hotkeys command with ;<id=xxx> in the target field of the hotkey configuration (note ; followed by <). Example: Exec Hotkey Reverse id=thisone.
IgnoreExplorerRestat *1/0*	If 1, explorer restarts do not cuase a PowerPro restart. Access with exec.ignoreexplorerrestart(n)
LogKeys	Log keystrokes to a file.
MoreCommandsAsScript	Executes commands in the More Commands edit box as a script, which allows you to use loops, jumps, if-elseif in More Commands. You must use Enter to separate command lines.
Monitor	Suspend or re-activate repeated running of a "monitor" command list, if checked in the "Special Lists" dialog.
Mute	Mutes sounds; run again to reverse.
NewFolder	Creates a new file folder.
OnError	Changes where error messages go and whether HookErrors list is run.
Print	Print a file using its associated program (usually Notepad for text files)
Prompt	Sets a flag according to a user prompt dialog
QuoteEscape	Use on, off, or reverse to determine whether single quote is an escape character in strings in expressions; only works if Setup >Advanced >Characters "Escape character" is set to single quote.
RefreshEnvironment	Refreshes all environment variables from registry (NT or later only); does not delete variables which are deleted.
RefreshImage	Followed by name of image file used in skin or as bar/menu background. It refreshes any in memory copy of the image information. Use when the file is changed while PowerPro is running.
RestoreLastMin	Restores last minimized window.
SchedulerAdd	Adds a new scheduled event. Specify date, time, and either a command or text (which will be set as a *Message event). The date and time must each be one number, separated by a blank (no blanks or colons or slashes within date or time). The date can be yyyymmdd or a number less than

1000 to specify that number of days from now (zero for today). The time can be hhmm to specify the time using 24 hour clock or +hhmm to specify that number of hours and minutes from now.

Examples: *Exec SchedulerAdd 20021225 0900 Merry Christmas

*Exec SchedulerAdd 0 +100 "c:\program files\myprog\mprogra.exe" runs myprog one hour from now.

You cannot use exec scheduleradd in a script called from a scheduled alarm unless you precede the Exec SchedulerAdd by wait 1.

StandardConfiguration	If >*Setup >Advanced >"use standard configuration"* is not checked, outputs an error message and stops all scripts.
ScrollwithMouse	Starts manual scrolling
ScrollWindow	Scrolls the window under the mouse.
SetBrowser brwsr	Sets default browser for browserurl function.
SetEnv text	Set environment variable to provided text. Use the env(...) function in expressions to read environment variables.
Tile	Tiles active windows.
TraceKey	Shows each key press and release in a debug window; displays virtual key code, scan code, extended key bit, key name. Run the command again to stop trace.
ToFile	Writes a single line of text to a log file. You can use the file plugin for file input and output to handle multiple lines.
VolumeAll	Set volume for all playback; enter number 0 (mute) to 255 (loudest). Use + or - in front of number to adjust relative to current setting.
VolumeWav	Set .wav volume; enter number 0 (mute) to 15 (loudest). Use + or - in front of number to adjust relative to current setting.
WindowInfo	Shows/hides a small window showing mouse position and window size.

Built in Commands : *Exec

Showing folders in Explorer

using *Exec Explorer / *Exec Explorer2

exec explorer folderpath

Shows the contents of the specified folder in explorer; both file folders and special folders like control panel may be used. Use find button to browse for folder when configuring. Use * for current working folder of active program.

exec explorer2 folderpath

Shows the contents of the specified folder in explorer in 2-pane window; both file folders and special folders like control panel may be used. Use find button to browse for folder when configuring. Use * for current working folder of active program.

using Explorer's command line

syntax

explorer.exe [/n][/e][,/root,object][[,/select],sub-object]

switches

/n Opens a new window in single-paned (My Computer) view for each item selected, even if the new window duplicates a window that is already open.

/e Uses Windows Explorer view. Windows Explorer view is most similar to File Manager in Windows version 3.x. Note that the default view is Open view.

/root,object Specifies the root level of the specified view. The default is to use the normal namespace root (the desktop). Whatever is specified is the root for the display.

/select,sub-object Specifies the folder to receive the initial focus. If "/select" is used, the parent folder is opened and the specified object is selected.

examples

To open a Windows Explorer view to explore only objects on \\server name

explorer /e,/root,\\server name

To view the C:\WINDOWS folder and select CALC.EXE

explorer /e,/select,c:\windows\calc.exe

see also

The *Menu Explorer command.

Built in Commands : *Exec

Hiding windows with *Exec

You can use the *Exec HideWindow built-in command to hide windows. You might use this if you do not want a window to appear on the Windows Taskbar, or the Powerpro list of active windows or the active task buttons.

When you execute an *Exec HideWindow command, the cursor changes to a cross. Left click on the window you wish to hide. This window, its top-level parent, and all the parent's children will be hidden.

You cannot hide a Powerpro window or the Desktop window. Some other programs will also refuse to be hidden.

If you execute HideWindow but then decide you do not want to hide a window, left click the mouse on the desktop or on a Powerpro window to cancel the operation.

If you want to show a hidden window, configure a Powerpro list of active windows to show hidden windows, assign this command to a menu, then execute the menu and select the hidden window from the list.

alternatives

Window Hide: The above method is is an interactive way to hide a Window by clicking it. To hide a named window using a command (from a script, hotkey, menu or bar item) use the *Window Hide command.

Tray min: If you are hiding windows to remove them from the Taskbar, you may want to use tray minimization instead.

Automatically Hiding Windows: You can specify that Powerpro should automatically hide any windows, should they become visible.

Put the comma-separated captions of the windows you want to autohide in the Auto Hide edit box on the main Setup dialog.

Built in Commands : *Exec

Prompting for Yes/No

Use *Exec Prompt to prompt for a Yes/No answer and set a flag with the result. For example:

Command	*Exec
Action	Prompt
Parameter	14 Any text

displays a message box with "Any Text" and sets flag 14 according to whether the result is yes or no.

Or you can prompt for a Yes/No/Cancel result by using a variable instead of a number:

Command	*Exec
Action	Prompt
Parameter	myvar1 Any text

Displays a yes/no/cancel dialog and sets **myvar1** to 0 for no, 1 for yes, 2 for cancel.

Built in Commands : *Exec

Logging keystrokes

You can log keystrokes into a file with *Exec LogKeys.

To start logging keys to file c:\path\mylog.txt, use

*Exec
Logkeys
c:\path\mylog.txt

To stop logging, set the file name blank.

To switch between logging and non logging, put an = in front of the file name:

*Exec
Logkeys
=c:\path\mylog.txt

will start logging the first time the command is run, stop it the next time, and so on.

You can write a heading to the logging file by checking that option on the *Exec Logkeys command configuration. You can also write a heading using *Exec ToFile with filename set to log to refer to the currently open logging file.

If you omit the path, the key logging file is assumed to be in the same folder as the PowerPro configuration file (usually pproconf.pcf).

The configuration checkbox option *"All Keys"* lets you determine how invisible keys like Alt and PageDown are handled. If *All Keys* is **not** checked, then only visible keys, spaces, tabs, and Enters are written to the file. If *All Keys* is checked, then all keystrokes are written to the file, using the *Keys notation for special keys, e.g. {alt} or {pgdn}. In addition, {alt}, {ctrl}, {shift}, and {apps} (Win key) will be written twice: once for key press and again for key release. Files logged with *"All Keys"* checked can be played back using *Keys {from …}

You can use the *Info keyword keylog to display an X on a button label if logging is active or the keyword keylogfile to view the file name of a logging file. You can also access this information with the built in functions keylog and keylogfile.

Built in Commands : *Exec

Tiling windows

You can tile active, top-level windows with the *Exec Tile command. It re-arranges active windows as follows:

- If the Parameters edit box starts with the letter c or the Control Key is held down, windows are arranged in equal-sized columns.
- If the Parameters edit box starts with the letter t or the Shift Key is held down, windows are arranged in equal-sized rectangles.
- Otherwise, windows are arranged in equal-sized rows.

You can restrict the affected windows to only those containing "text" in their window captions by putting *text in the Parameters edit box. For example, *explor means that only Explorer windows are affected.

To start two instances of Explorer and tile them use the following:

Command:	Explorer.exe
More Commands:	explorer.exe *wait 1 *Exec Tile *explor

You need to ensure that the *"Switch To If Active"* box is unchecked.

Built in Commands : *Exec

See mouse cursor position and window information

using *Exec WindowInfo

You can have Powerpro display a small window with the mouse screen position and the size and position of the window under the mouse as well as the caption, window class, and exe name of the window under the mouse. This display can be activated manually or it can be automatically shown whenever you move or size a window.

To manually show the information window, execute this command:

Command:	*Exec
Action:	WindowInfo

The window will be displayed until you execute the command again (i.e. to stop display, execute the command again).

To show the information automatically whenever a window is moved or sized, use the check box on the GUI Control dialog.

The information window has six lines of text:

1. mouse screen coordinates, both Absolute (point 0,0 is top-left of screen) and Relative (point 0,0 is top-left of window under mouse)
2. window coordinates: (left, top) - (right, bottom) of window under mouse
3. total window size: width x height , client window size and aspect ratio
4. window caption
5. main window handle in hex, window class and exe name
6. child window handle, id, and class name; handle and id are shown in hex preceded by 0x.

The client window excludes the border, caption, menu bar, tool bars, and status bar. The aspect ratio is the width of the client window divided by its height.

If the mouse cursor is over an Edit box, the contents of that box are shown as the caption. This can be useful to see password fields.

The display window uses the same colors and font as the tool tip window.

For the automatic display, to have the information in the display updated dynamically as you move or size a window, you must have the Windows Control Panel, Appearance, Effects option "Show Window Contents While Dragging" activated; this option is available in NT4 and Win 98 or in Win95 with MS Plus!.

Built in Commands : *Exec

Tiny Type and Run box

using *Exec CommandLine

purpose

If you want an easily accessible but unobtrusive command line, use the built-in command *Exec CommandLine. It creates a small window consisting of a single drop down edit box. You can type any command into this box and press enter to have the command executed. Or, if you have a three-button mouse, you can execute the command by middle-clicking on the edit box.

usage

You can select the command from the drop down which stores the last 50 commands entered.

You can use auto completion with the command line by putting the auto complete keywords after the action command: Exec CommandLine keywords

Put the *Exec CommandLine command as a Powerpro start up alarm if you want the run box to appear when Powerpro starts.

syntax in the command line box

If your command's filepath contains blanks, you must surround it by double quotation marks:

"c:\path to\notepad.exe" "c:\docs folder\favorite quotes.txt"

Powerpro will send any command line starting with www. or containing //: to a running browser to be used as a URL (the browser must be Netscape or IE and must already be running).

You can execute a "Dos" command by specifying *Dos immediately followed by the command line. The command is written to file ppro.bat file (.cmd in NT) in the PowerPro folder and then that .bat file is executed. Use explorer to set the properties of the .bat file to change its configuration (e.g. full screen versus window). Precede the Dos command by *hide to run it in a hidden window. For *Dos commands, if you use CurrenntDirectory = "drive:/folder" to pre-set the special variable CurrentDirectory to a drive:\folder, PowerPro will do a cd /d to the directory stored in that variable before running the command.

You can create your own special processing of the command line by creating a command list script called HookCommandLine. When this script exists, PowerPro executes it just before running the command line. The variable x0 will be set for the script to the contents of the command line, and whatever the script sets x0 to will be executed. If x0 is set to "", nothing is executed. For example, the script could scan the command line in x0 and handle command aliases.

configuration

After you first start the Tiny Run Box, drag and resize its width to the desired dimensions. Powerpro will remember the location and width the next time the run box is started.

You can further configure the run box by right-clicking on the edit box (not the caption). You can then:

- specify that the run box should shrink when inactive (see below for details)
- specify that the run box should close when inactive for ten seconds
- specify that the run box should/should not be always on top
- specify that all commands expect those starting with "win " should be prefixed by *dos (useful if you use the run box mainly for dos command line commands)
- specify whether or not the caption and resizing window frame should be shown
- pick a background color for the window
- browse for a file to execute
- execute the command in the run box
- save the current size to be used as the shrunk size
- specify whether the run command should switch to another instance, if it is active
- specify whether Powerpro should try to interpret the command as a URL to send to a running browser

even tinier

To keep the run box out of the way when not in use, you can specify that it should shrink when not active. Follow this sequence of steps in the order given:

1. Set the caption on.
2. Move to position so that left of window is at desired location.
3. Resize the width to desired shrunk width.
4. Select "save shrunk width" from configuration menu.
5. Resize to desired large width.
6. Select "shrink if inactive" from configuration menu.
7. Turn caption off, if desired.

tip

If you use the keyboard extensively, you may want to configure a hot key to activate the tiny run box, by setting the hot key command to *Exec CommandLine

Built in Commands : *Exec

Date Calculator

Use *Exec CalcCalendar to show a dialog with two calendars and with calculations for day number, week number, and difference between dates in days, weekdays, and weeks. As detailed below, changing any field on the dialog refreshes all the others.

The dialog is shown at the current mouse position. To center it instead, put *Window Center Active in the *More Commands* box of the command entry controls.

To change either of the dates, you can click on the day, month, or year subfield within the date and then use the arrow or number keys to enter a new value. Or you can click on the drop down arrow, and then select day, month, or year by clicking on that field.

using the dialog

To display the week number or day number of a date: set the top date and read the day and week number.

To display difference in dates in days, weekdays or weeks: set both dates and read the difference. The difference excludes the day of the later date (e.g. there is a two day difference between July 2 and July 4). Differences will be negative if the second date precedes the first date.

To show the date for a given week number or day number: set the week number or day number and read the first date.

To find the date a given number of days, weekdays, or weeks before or after a given date: set the first date and set the difference (which can be negative for before) then read the second date. Remember that the day of the later date is excluded from difference calculations.

The week number definition follows the ISO standard: Week 1 of any year is the week that contains 4 January, or equivalently Week 1 of any year is the week that contains the first Thursday in January.

To just display a simpler calendar, you can also use *Exec Calendar.

Built in Commands : *Exec

Error Handling

You can change the way PowerPro displays error messages with the Exec OnError command. You can control how errors messages are displayed and whether or not a command list called HookErrors is called whenever an error occurs.

The command takes one of these forms:

Exec OnErrors keywords

Exec.OnErrors("keywords")

The keywords can be any of the following:

none - Do not display any error messages

file - Write errors messages to file ErrorLog.txt only (in same folder as .pcf)

display - Display messages on screen only

both - Write error to file and display on screen hook - run command list HookErrors whenever an error occurs

unhook - Do not run command list

clear - Set _LastError_ global variable to "" (You can also say _LastError_ = "");

For example, the following writes errors to files only and activates the hook list:

```
Exec.OnErrors("hook file")
```

The global variable _LastError_ is set to the error message whenever any error occurs.

If an error occurs while the HookErrors script is running, the script is not called again.

Built in Commands : *Exec

Working with CDs

Use *Exec CD to control your audio CD player. Enter one of the following commands in the parameters box.

Play n	Plays tracks starting at number n. Omit n to play starting with track 1.
Next	Plan next track.
Previous	Play previous track.
Stop	Stop CD.
Eject	Ejects (opens door for) default audio CD.
Close	closes door for default audio CD.

You can display the current CD track and maximum track with *Info.

Built in Commands : *Exec

Sound volume and mute

using *exec commands

You can mute sound volume with this command

Command	*Exec
Parameter	mute

Each time this command is executed, the mute setting is reversed.

You can set the volume for .wav files only with *Exec VolumeWav n where n is a number between 0 and 15. Use +n or -n to adjust volume relative to current setting.

You can set the volume for all playback with *Exec VolumeAll n where n is a number between 0 and 255. Use +n or -n to adjust volume relative to current setting

Built in Commands : *File

*File command

PowerPro has internal file manipulation commands for renaming, copying, and deleting files. You can also select a file at random from a folder and copy it over a specified file.

The *File actions are

copy	copies one file path to a second path; you can use wildcards to copy many files
rename	renames one file path to another; can be used to move files to another folder; you can use wildcards to move/rename many files
move	same as rename
extchange	change, remove, or add a file extension.
delete	deletes a file; you can use wildcards to delete a set of files
deletenorecycle	deletes a file without putting it in the recycle bin;
deleteold	deletes files in folder older than specified number of days.
copyrandom	copies a randomly selected file to a specified file path
runrandom	runs a randomly selected file
commandrandom	runs a command with a randomly selected file as a parameter.

Put double quotation marks around file paths which contain blanks.

For Copy and Rename, if the second path is a folder, then the file name for this target is taken from the file name of the first path.

If you check *"Confirm"* in configuration, then deletes or overwrites of a file or creation of a new directory will be confirmed first. If you check *"Folders"*, then folders will be included in *.* wild card operations.

For DeleteOld, you must provide a number of days then a folder path, optionally including wild cards in a file name. All files older than the specified number of days are deleted to the recycle bin. You can use recent as the folder name to access the folder of recently accessed documents.

Extchange works as follows: First put a file path to be changed, possibly with wildcards (or use | as the first file name if the *File Extchange command is in a context menu). After this first file path, put a single dash (-) to remove the extension, or +xxx to add .xxx as the extension, or yyy to replace current extension with yyy. For example, if a context menu contained

*File Extchange | +jpg

adds .jpg to the files selected in explorer when the context menu is clicked.

examples

*File

Copy

c:\mypath\in.txt c:\output\out.txt

copies in.txt to out.txt.

*File

Rename

c:\mypath\in.txt c:\output

moves in.txt to folder c:\output.

Built in Commands : *File

Writing entries to a file

You can use the following command to append a line of text to a file:

Command	*Exec
Action	ToFile
Parameter	"filepath" text

writes the single line of text to the end of the file given by filepath. Enclose the filepath in quotes if it contains blanks. A single blank after the filepath is ignored and then the text after this blank is written.

Use a file name of log to write the text to a currently open key logging file.

examples

```
*Exec
ToFile
c:\logs\log1.txt this is the text
```

writes this is the text to c:\logs\log1.txt

```
*Exec.ToFile("c:\\logs path\\log1.txt", date ++ "date included")
```

writes the date then the phrase "date included" to c:\logs pth\log1.txt.

Built in Commands : *File

Working with a randomly selected file

The commands *File RunRandom *File CopyRandom *File CommandRandom can be used to select a file at random using a file path with wild cards that you provide.

configuration

*File CopyRandom filepath outfile

selects a file at random from the filepath (which must contain wildcards like *.bmp) and copies it over outfile.

*File RunRandom filepath

selects a file at random from the filepath (which must contain wildcards like *.bmp) and runs it using the program associated with the file extension.

*File CommandRandom commandpath filepath args

selects a file at random file filepath (which must contain wildcards like *.*), then runs the command given by commandpath -- using a command line consisting of the commandname, the selected file, and finally the args. If you want the command to be run invisibly, put *hide after the args at the end of the parameters edit box. If the command being run is a .bat file, you may want to use explorer to set its properties to include close on exit. This is especially important for commands run invisibly.

One use of this command is to set up your own randomization routines for system media files. For example, you can randomize the Windows shutdown screens by creating .bmp files with the appropriate size and color depth, putting the files into a folder, and using the command to copy a randomly selected file over c:\windows\logow.sys. Take a backup copy of logow.sys before experimenting with this.

To implement randomization on a schedule, put the command as a scheduled command.

examples

Command	*File CopyRandom
Parameters	"c:\my logo files*.bmp" c:\windows\logow.sys

copies a random bmp file from c:\my logo files over the logow.sys file.

Command	*File RunRandom
Parameters	"c:\zounds\*.wav"

plays a random wav file from c:\zounds.

Command	*File CommandRandom
Parameters	"c:\program files\bat\exec.bat" "c:\random\*.*" arg2 arg3 *hide

selects a random file from c:\random, then executes the exec.bat file with the selected file as the first argument, then arguments arg2 and arg3. The command is run in an invisible window.

Built in Commands : *Format

Formatting menus and bars with *Format

Use these *Format command to control the look of menus and bars.

The list indicates which Format commands can only be used with Menus, or only with Bars, or with both.

configuration

Use the following actions with *Format:

Drag	Bar	Assign to a bar button and then click-drag that button to move bar.
NewBarRow	Bar	Starts a new row
NewBarRowLine	Bar	Starts a new row and draws a separator line; check "no3d/disable" for flat look.
BarVerticalLine	Bar	Draws a vertical line between buttons; check "no3d/disable" for flat look.
Separator	Menu	Inserts a horizontal separator
NewColumn	Menu	Starts a new column
NewColumnLine	Menu	Starts a new column with a separating vertical line.
MaxColumn n	Menu	Specifies maximum number entries in column as number n. May not be effective with embedded submenus. Use explicit *Format newcolumn in this case.

StartSubMenu	Menu	The following items in the list appear in a submenu.
EndSubMenu	Menu	Ends the submenu. You can nest submenus up to 4 deep.
Context	Menu and Bar:	Starts a menu section, or shows a whole bar, depending on the active window.
ContextIf	Menu and Bar:	Starts a menu section, or shows a whole bar, depending on an expression being true
EndContext	Menu	Ends portion of menu depending on active window.
Item	Menu and Bar:	Changes the colors and text of a menu or bar item.

Note: Unlike most built in commands, Format commands cannot use the alternative Expression Syntax format.

Built in Commands : *Format

Changing the look of an item with *Format Item

The format item command is obsolete. Use the cl.xxx functions instead.

You can execute the *Format Item command to change the look or text of an item on a command list used for a bar, menu, or tray icon item.

When you configure this command, use the button beside the format keywords edit control to access a dialog which will set the keywords needed to change item text or color and to specify whether or not the item should be visible and should use its own colors. You also specify the command list and starting and ending item number (starting at 1) of the item to be changed. Instead of specifying a bar and item number, you can specify an item number of 0 to indicate the last button pressed on any bar.

There are restrictions on changing the text associated with special Info labels. You cannot change the text from an ordinary label to an *Info label or the reverse. You cannot change tool tips which use *Info at all.

When you set new item features with the dialog, you can use a checkbox to indicate whether the new item values are to be written into the configuration file. If they are not written, then the values will be reset if you use the configuration dialog and save the new configuration. If this is not what you want, you could put the *Format Item to reset the desired values in the Reconfigure command list, as set by the advanced setup dialog. This command list will be run after each time you reconfigure so you need to program it to run the appropriate *Format Item values.

Built in Commands : *Format

Window-specific menu and bar contents

*Format Context

Using the *Format Context captionlist and *Format EndContext built in commands, you can specify that portions of a menu should only appear only if a specified window or program is active.

This is useful, for example, to set up menu entries attached to hot keys where different parts of the menu appear depending on which program is active when the hotkey is pressed. The menu could contain *Keys commands to send keystrokes to activate program features. *Format Context would be used to display the *Keys commands which were appropriate for the active program.

The *Format Context captionlist function can also be used on a button bar to show or hide the bar depending on the active program. (Such a bar could be attached to the active window as its bar Position setting.) Different bars would then appear depending on which program is active. Each bar could contain commands relevant to the active program.

*Format ContextIf

Using the *Format ContextIf expression and *Format EndContext built-in commands, you can specify that portions of a menu should only appear only if a specified expression is true.

ContextIf expression: A single ContextIf expression can also be used as the first command of a bar to control whether or not the bar is visible.

configuration

to create a program-specific portion of a menu:

Insert a *Format Context command into the menu. In the parameters edit box, put a list of window captions and exe file names. Use *xxx for captions ending in xxx, yyy* for captions starting with yyy, and =exename (no path or extension) for all windows from the program whose .exe file is exename. Follow this command by the program-specific menu entries. End with the *Format EndContext command.

Or insert a *Format ContextIf command into the menu. In the parameters edit box, use an expression as the condition for showing this portion of the menu. For example,

Command	*Format ContextIf
Parameter	(modem or a>0)

displays the menu portion only if the modem is connected or variable a is greater than 0.

to create program-specific bars:

With bars, the *Format Context captionlist or ContextIf expression command applies to the entire bar; it must be the command for left clicking the first button. Do not use a *Format EndContext command.

example

The following illustrates a set of menu entries to send control-I (view images) and Ctrl-arrow-left (go back) only if Netscape Navigator (netscape.exe) is active.

Item Name	Netscape only
Command	*Format Context
Parameter	=netscape

Name	Images
Command	*Keys
Parameter	^i

Name	Back
Command	*Keys
Parameter	%{al}

Name	EndContext
Command	*Format EndContext
Parameter	

You cannot use these commands in menus attached to clicking on the Desktop as the Desktop will always be the active program in this case.

Built in Commands : *Keys

*Keys command

sending keystrokes to other windows

Use the *Keys command to send keystrokes to other windows, either for sending text into the document being edited in the receiving window or to automate functions of the receiving window. For example, functions on menus can often be accessed by Alt-ab where a is the first character of the menu name and b selects an item on that menu.

the keystrokes

When configuring the command with the pprofconf.exe tabbed dialog, the keystrokes to be sent are entered in the parameters edit box. You can use the find button to select a special key or to record keys.

Type letters, digits, special characters in the parameters edit box. Special characters like function keys or the date/time can be entered using {xx} abbreviations, such as {f1} or {enter}{back 3}. The full list of these special {codes} is in a topic below.

To specify an Alt-prefixed key, prefix it by %; similarly use ^ for Ctrl key, + for Shift, and combine as needed (e.g. %^ for both Ctrl and Alt). You can often simulate menu selections by sending % followed by a set of characters; e.g. %fn sends Alt-fn which does a File|New menu selection in many programs. Note: for most reliable operation, use ^ or + or % rather than {ctrl} or {shift} or {alt}.

Beware of these characters which could have a special meaning for Powerpro:

%	use {pe}	or {%}	(% alone signals Alt)
^	use {ca}	or {^}	(^ alone signals Ctrl)
+	use {pl}	or {+}	(+ alone signals Shift)

If the keystrokes start with a double quote, then that double quote and any ending quote are removed. So to send keys start with a double quote, add a quote to the start.

the target window

Normally keys are sent to the currently active window, but you can switch to another window first by preceding the sequence of keys with {to xxx} where xxx specifies the new target window.

timing 1

If your sequence of keys causes the window receiving the keys to open a new window or menu to receive subsequent keys, you may need to insert a wait in your key sequence to allow the new window to open and be readied to receive the keys. Use {w1} to insert a wait on one tenth of a second.

Starting a program and then sending it keys requires special care; see Sending *Keys to a new window for information.

timing 2

PowerPro has three methods of sending keys, the "fast" method (journal hooks), a "slow" method (keybd_input), which is the default, and the sendiput method.

Use *>Setup >Advanced >Other* to set the default to {fast}. The default Slow method works for most keys, but the Fast method is needed to send shift in XP and W2K.

If you have set the Slow method, you can temporarily select the Fast method for a single *keys command by preceding the keys to be send with {fast}

*Keys {fast}{home}+{end}

If you have set the Fast method, you can temporarily select the Slow method for a single *keys command by preceding the keys to be send with {slow}

*Keys {slow}{home}+{end}

The sendinput method works like the slow method, but sends all keys in a block, rather than one-by-one, making it less likely that manual keyboard input will be intermixed. You must put {sinp} at the start of the keys sequence to use the sendinput method.

If you cannot get send keys to work, try the other method from your current setting. Also make sure you are using ^ for control and + for shift (not {ctrl} or {shift})

keystrokes from a file

If you have a large number of keys to send, you can store them in a file (say c:\path\filename.txt) and then use *Keys {from c:\path\filename.txt} to send the keys.

You can use many lines in the file to make it easier to enter and check the keys; all line ends are ignored. You can also put a comment at the start of the file by putting ** at the start of the file, then any number of lines of comment text, then ** at the end of a line.

Example:

One line comment

Example:

**

Multi-line

Comment

**

If you specify a file name without a path, then the file is assumed to be in the same folder as the PowerPro configuration file.

If you use {from ...} it must be the only item in the parameter; you can put other items such as {to ...} inside the file; or you can use a *Window Show command before the *Keys command to activate the target window.

keystrokes from a menu

You can select keys to be sent from a file based menu with *Keys {filemenu c:\path\items.txt}

other uses

You can use a command list of *Keys commands, for example to have a list of favorite folders to select from for use in open/save dialogs.

To send mouse clicks, use *mouse commands.

example

Command	*Keys
Parameter	hello, world

Sends **hello, world** to the active window.

Built in Commands : *Keys

*Keys target window

Specifying the Window to Receive the Keys

The *Keys command normally sends keystrokes to the foreground (active) window. You can reset the foreground window before sending the keys; or cause PowerPro to change the receiving window by putting {to xxx} or {toany xxx} at the start of the keys to be sent. The difference between the two is that {to xxx} only works with visible windows and always leaves the focus at the window receiving the keys whereas {toAny xxx} works with both hidden and visible windows and returns the focus to the window which had it before keys were sent.

For both, xxx indicates the target window and can be:

{to none}	does not check if window is available; {to none} makes keys available in keyboard buffer only
{to *}	sends keys to the current active window
{to captionlist}	sends the keys to the first window found matching the captionlist for example the following items:
{to =File Path}	sends keys to program run from that "File Path"
{to Title}	sends keys to window with caption "Title"
{to PartTitle*}	to window with caption starting with "Part Title" (Note asterisk at end)
{to *PartTitle}	to window with caption ending with "Part Title" (Note asterisk at start)
{to *PartTitle*}	to window with caption containing "Part Title" (Note asterisks at start and end)
{to autorun}	to last window matched by autorun command list
{to activebar}	to last window referenced by an Active bar button

prefixes

Except for {to none}, if the {to xxx} window is not found, you will normally get an error message. Precede the window ID with the character ^ to avoid the error, e.g.

keys {to ^*notepad}abc

avoids the error message if no Notepad window is open.

To wait for the specified window to be ready:

keys {to +captionlist}abc

waits up to 3 seconds for the window specified in captionlist

keys {to -captionlist}abc

sleeps a short time before sending the keys

Built in Commands : *Keys

*Keys codes for keystrokes

specifying the keystrokes to be sent

This page is about specifying the keystrokes to be sent.

literal text

Send letters, numbers, and other keyboard characters simply by typing them as you want them to be sent: *Keys text to send

prefix modifiers

which don't need braces: % for Alt ^ for Ctrl + for Shift

Those three prefixes affect the next one key only, so: *Keys +abcd sends: Abcd

sticky modifiers

To modify several keys in succession, use:

{alt} or {at} ; {shift} or {sh} ; {ctrl} or {co} ; {win} or {wi}

before and after the characters to toggle the up/down status of Alt, Shift, Ctrl or Win

Example: *Keys a{shift}bcde{shift}f sends: aBCDEf

ascii characters

To send Alt+xxxx keys (such as: alt+0181 =µ) use:

{alt}{pad0}{pad1}{pad8}{pad1}{alt}

You may have to uncheck *"Use fast send keys"* in *Setup >Advanced >Other* -- or precede the keys with {slow} -- for this to work on your system.

list of codes for the *Keys command {in braces}

Note: some of these show two or three alternative codes

{+} {plus} {pl}	Plus sign
{%} {percent} {pe}	Percent sign
{^} {caret} {ca}	Caret ^
{{} {brace} {br}	Left brace {

{up} {au}	Up arrow
{down} {ad}	Down arrow
{left} {al}	Left arrow
{right} {ar}	Right arrow

{enter} {en}	Enter
{space} {sp}	Space
{quote} {qu}	double quote character single quote does not need { }
{question} {qn}	question mark
{greater} {gt}	greater than sign >
{less} {lt}	less than sign <
{tab} {ta}	Tab character

{printscreen} {ps}	PrintScreen Key
{ins} {in}	Insert Key
{del} {de}	Delete Key
{back} {ba}	Backspace
{home} {ho}	Home
{end} {ed}	End

{pgup} {pu}	Page Up
{pgdn} {pd}	Page Down
{esc} {es}	Escape
{pad+} {p+}	Numeric Pad +
{pad-} {p-}	Numeric Pad -
{pad*} {p*}	Numeric Pad *
{pad/} {p/}	Numeric Pad /
{pad0} {p0}	Numeric Pad 0 [similar for Pad 1 through 9]
{scrolllock} {sl}	Scroll lock

["Use fast send keys" on Advanced Setup must be **un**checked -- or precede the keys by {slow}]

{capslock} {cl}	Caps lock (toggle) (needs slow method)
{apps} {ap}	Send the Apps key, as pressed down then up
{dateshort} {ds}	date in Windows short format
{prevshort} {xs}	previous day's date in Windows short format
{nextshort} {ns}	next day's date in Windows short format
{datelong} {dl}	date in Windows long format
{time} {ti}	time in Windows format

[To send the time without seconds use {time}{back}{back}{back}]

{f1} ... {f12}	Function Key F1 to F12
{wn}	Wait n tenths of a second (eg {w1} to wait one tenth of a second)
{nnn}	Send character with decimal ascii code nnn (first n cannot be 0).
{alt} {at}	Toggle Alt down or up (see note re Modifiers above)
{shift} {sh}	Toggle Shift down or up (toggles state)
{ctrl} {co}	Toggle Ctrl down or up
{win} {wi}	Toggle Win down or up
{cmdsep} {cs}	command separator character [obsolete]
{param} {pp}	command line prompt character [obsolete]
{clip} {cc}	clipboard character [obsolete]
{var} {sv}	expression variable character [which is & by default]

[&() is not favored as best practice. See do( ) usage or the new Expression Syntax]

{fast}	following keys will be sent using the fast method
{slow}	following keys will be sent using the slow method

[to contradict your default setting in *Setup, Advanced, Other*]

other special {codes}

*Keys {to folder}c:\path

Sends the folder path c:\path to an open/save dialog. The {to folder} keyword tells PowerPro to automatically select the file edit box to receive the keys, save the contents of that box, send the keys to change to c:\path, and then restore the previous contents of the file edit box.

See here about favourite folders for open/save dialogs

*Keys {filemenu c:\path\items.txt}

The user selects the keys to be sent from a file menu

See here about using FileMenu with *Keys

*Keys {from c:\path\filename.txt}

Sends the contents of the specified file. With this form of the *Keys command, you cannot add any literal text nor {codes} as a parameter to the actual *Keys command. Instead you can include them in the special text file of keys to be sent.

notes

The keystrokes to be sent are entered in the *Parameters* edit box. You can use the find button to select a special key or to record keys.

If the keystrokes start with a double quote, then that double quote and any ending quote are removed. So to send keys starting with a double quote, add a quote to the start.

You can change either of the two special characters used to delimit the special codes, which are { and } by default. Change the { or the } or both to any non-alphanumeric using the *Setup >Advanced >Characters* dialog. They can both be set to the same character.

Assuming you are using the standard { and } delimiters: To send the opening brace, use {{}. To send the closing brace, just use }.

limits

You can send at most 1000 keys. However if sending more than a few characters, it is faster to use Paste commands such as Clip TextPaste some text

Built in Commands : *Keys

Examples of *Keys commands

examples

Command	*Keys
Parameter	^{ed}

Sends Ctrl+End to the active window. This key combination often tells the program to go to the end of the information being displayed.

Command	*Keys
Parameter	this text contains spaces

Sends this text contains spaces to the active window.

Command	*Keys
Parameter	{to =prog}^{ho}abc

Sends Ctrl-Home followed by abc to window started from prog.exe.

Command	*Keys
Parameter	{to *Notepad}%fo

Sends Alt-f followed by o to the window with caption ending in Notepad. This would select the "Open" command from the menu.

Built in Commands : *Keys

Sending *Keys to a new window

sending keystrokes to programs when they are started

Since Windows is a multitasking system, starting programs and sending them keys requires care. You must make sure the program you are starting is ready to receive them.

To start a program and send it keys at start up, use multiple commands. For example, to start c:\ql\myprog and send alt-g n, specify

Command	c:\ql\myprog.exe
More Commmands	*wait ready
	*keys "%gn"

The sequence *wait ready causes Powerpro to wait until the program is ready to accept input before sending the keys.

If the *wait ready does not work for some reason, try *wait 2 (or some other digit) to wait 2 seconds.

You can also wait for up to 3 seconds until a window with a specified caption appears by preceding the caption with a + and using the {to xxx} option:

Command	*Explorer
More Commands	*keys { to+Exploring*}"%vd"

Starts Explorer, then waits for up to 3 seconds for the window with a caption containing Exploring to appear, then sends Alt-v followed by d to the window with caption starting with Exploring. This could set the Details view for Explorer. This is especially useful with Explorer, since Explorer is always running if you use the standard Windows shell. You must use the *Keys command for this approach.

Built in Commands : *Keys

Selecting *Keys from a menu

selecting the keys to be sent from a menu

When sending keys with *Keys, you can display a menu of selections to determine some of the keys to be sent. The selections are stored in a file which specifies pairs of: the menu item text and the corresponding characters to be sent.

To show the menu, use:

```
*Keys {filemenu c:\path\items.txt}
```

Each line in c:\path\items.txt is of this form

```
name=keys
```

There may be many lines in the file. Each line defines a menu item with the item label set to name. If you select a menu item, the corresponding keys will be used. If you do not select a menu item, then no keys will be sent at all.

The keys can contain special sequences like {tab} or modifier keys alt or ctrl (% or ^). You can also use name= with no following keys to allow a selection of nothing without canceling the entire *Keys command.

The name field may contain an & which will cause the following character in the name to be a menu mnemonic selecting that menu item from the keyboard.

You can put blank lines in the file. You can put a line consisting of the word sep in the file for a horizontal menu separator and a line of colsep to start a new menu column. Use startsubmenu sss to start a submenu called sss and endsubmenu to end a submenu.

You can conditionally include portions of the menu by using

```
Contextif (expression)
label1=result1
label2=result2
Endcontext
```

Normally, tabs at the start of lines are ignored. But if you include the line

```
tabs keep
```

in the file, then tabs will be used to indent menu entries. To starting ignoring tabs again, use

```
tabs ignore
```

The title=result lines are only included if the expression is true. You cannot nest Contextif.

You can specify other keys to be sent, beside what is chosen from the {filemenu}; for example:

```
*Keys abc{filemenu key.txt}yz
```

will send abc, then the selection from key.txt, then yz.

You can use more than one filemenu in a single *Keys command.

If you specify a file name without a path or with a relative path, then the file is assumed to be in the same folder as the PowerPro configuration file. If the filename contains a *, the * is replaced by the exe filename of the foreground window, which allows the selected file to depend on the active window. Finally, you can use two file names with {filemenu} by separating the file names with a comma. For example

```
{filemenu keys\_common.txt,keys\*.txt}
```

with Microsoft Word in the foreground would create a single menu from the entries of

c:\program files\powerpro\keys_common.txt

followed by

c:\program files\powerpro\keys\winword.txt

Use a semi-colon (;) instead of a comma to separate the file entries with a column separator.

other uses for filemenu()

You can also use filemenu(filespec) in expressions in a command's parameters. When the command is run, a menu will be displayed. The result of the users menu choice is substituted into the expression before the command is executed.

For example: `Bar.Show(filemenu("c:\tests\MenuOfBars.txt"))`

works if MenuOfBars.txt contains lines which pair each menu item label to a Command List name

```
Show my small blue bar=bluebar1
Show my large blue bar=bluebar2
Show no bars=
```

Built in Commands : *Macro

*Macro command

setting up keyboard macros

purpose

Keyboard macros let you replace one set of typed characters by others. You can also use keyboard macros to run Windows programs or to execute Powerpro Windows configuration features or built-in commands.

For example, you could define .me to be replaced with your name. Or you could define Alt-tm to minimize the current window.

configuration

To define a set of keyboard macros, you need to do two things: define the macros and define the macro signal character.

You define the macros and the corresponding actions by creating a command list. Enter the macro as the item name and enter the macro command as the corresponding left command. For the name, you can use letters, digits, spaces, and the lower case characters from your keyboard (`-=[];'\/.,). You can use digits from the numeric keypad as well.

Use the *Keys command to send keystrokes if you want to define a macro abbreviation for the corresponding keystrokes.

After defining the macros, you need to define a hot key character which is used to signal that a macro may follow. You do this by defining any hot key with the command *Macro in the command entry edit box. Put the name of the command list with the macros in the parameter edit box of the command.

You can temporarily disable a macro in the command list by checking the *hidden* check box.

When you type a macro such as .me, PowerPro normally sends backspace characters to erase the .me: three backspaces would be sent for this example. If you do not want PowerPro to send backspace characters for a macro, check the "Disable/no3D" check box for the command list entry. If you find that the backspaces are not functioning properly, try checking "Hide Icon" which will use a different way of sending backspaces.

examples

For example, suppose you define a command list mymacros with these four entries:

Item Name	me
Item Command	*Keys
Item Parameters	yourname@yourdomain.com

Item Name	new
Item Command	*Keys
Item Parameters	%fn

Item Name	sq
Item Command	*Keys
Item Parameters	{sp}{ba}²

Item Name	xp
Item Command	c:\windows\explorer.exe
Item Parameters	

Also suppose that the period is defined as a hot key as follows

Hot Key	. [that is a point]
Hot Key Command	*Macro
Hot Key Parameters	mymacros

When you type .me, Powerpro would replace the .me by yourname@yourdomain.com. Similarly, .new would be replaced by Alt-fn, and .sq would be replaced by the superscript 2 (²). Finally, typing .xp would cause Windows Explorer to be started.

If you type a period followed by any other sequence of characters, nothing will happen - the typed characters will not be changed.

further information

Be careful when you define macros: Powerpro will execute the shortest macro that applies. For example, if you define one macro ab and another one called abc, then the abc macro would never be executed since the ab macro would always be matched first. To help avoid this, you can put spaces in macros, including spaces at the end. The space then has to be typed for the macro to be executed.

You can have as many combinations of macro signal characters and menu tables for macros as you want.

You can use program-specific hot keys to limit macro expansion to certain windows or to avoid checking for a macro with certain windows.

The *Macro command can only be used with hot keys. You will get an error message if you use it in any other context (e.g. as a button command).

Built in Commands : *Menu

*Menu command

Use *Menu to display a menu. It can be built from a command list (*Menu Show), the files in a folder (*Menu Folder), recently executed commands (*Menu Recent), captured Explorer folders (*Menu Explorer), or the Windows Start menu (*Menu StartMenu).

*Menu commands

Show	Shows a command list. as a menu. Use *Format to insert submenus and separators..
ShowFile	Shows a file as a menu; this file can contain *Format commands to format the menu.
ShowAtButton	Shows a command list as a menu aligned to last button pressed. Use *Format to insert submenus and separators.
ShowAtCursor	Shows a command list as a menu aligned to test cursor. Use *Format to insert submenus and separators.
Folder	Create a menu from a folder.

Recent	Shows a menu of recently executed commands. You must check *"Track recent commands"* on *Command Lists >Setup*.
Explorer	Show folders recently accessed with Explorer. You must check *"Track explorer"* on the *Setup* dialog. Set maximum number of entries and set number of entries per menu column using the *Setup >Advanced >Limits* dialog.
StartMenu	Show Windows start menu at mouse cursor. Alternatively, you can assign *Menu Folder StartMenu to a hotkey.
Tray	Show tray icons as a menu.
UnderMouse	Show the menu of the window under the mouse. Only works in Win95/98 and only shows standard menus (not toolbars like the menu tool bar in Explorer). Must be assigned to a hot key or as part of menu shown by hot key.

Note that there are two steps to showing a command list as a menu: defining the command list, called say "MyMenu1", and then executing a *Menu show MyMenu1 command. For example, to show a menu by pressing a button, assign *Menu Show MyMenu1 to a bar button command; see the default bar menu button for an illustration. To show a menu by activating a hot key, assign the *Menu Show MyMenu1 command to a hot key

If you have a bar configured to show a menu with *Menu Show xxx, you can quickly configure menu xxx by Alt+clicking the bar button with the *Menu Show command.

The global variable _pickedline_ is set after a Menu command to 1 if item picked; 0 if menu dismissed without picking an item.

tips

To create a menu of favorite folders, create a command list with a set of commands like this

Command	*Exec
Action	Explorer2
Parameter	c:\pathto\folder1

Then display it with *Menu Show. Use *Exec Explorer for a single pane window.

To create a menu of text clips, create a command list of *Keys or *Clip TextPaste commands.

Built in Commands : *Menu

Showing menus

You can use *Menu Show or ShowAtButton or ShowAtCursor to show a command list as a menu.

*Menu ShowAtButton is meant for use on bar buttons and shows the menu aligned with the button used to display it. For horizontal bars, the menu is shown below or above the button, depending on which half of the screen the bar is in; for vertical bars, the menu is shown to the left or right of the button.

*Menu ShowAtCursor ishows the menu at the text cursor.

*Menu Show gives more control of the menu position. Use the edit box *"Blank or enter position"* to enter a keyword specifying the menu position; the find button displays a menu of valid keywords:

centerundermouse	centers the menu under the mouse

centerscreen	center the menu on the screen
offset n1 n2	shows menu n1 pixels to left and n2 above the mouse (n1 or n2 can be negative)
screen n1 n2	shows menu at screen position n1 (from left), n2 (from top)
horbutton	shows menu aligned under/above last button pressed on horizontal bar
horbuttoncenter	shows menu centered under/above last button pressed on horizontal bar
verbutton	shows menu aligned right/left with last button pressed on vertical bar
cursor	shows menu at text cursor

The keywords horbutton and verbutton perform the same alignment as *Menu ShowAtButton, except that by using *Menu Show, you can choose whether horizontal or vertical bar alignment is used.

With *Menu Show, you can display a subset of a command list as a menu as follows. Start display at named item xxx by putting @xxx in the edit box under the one used to select the name of the command list in the *Menu Show command. In the command list, insert an item *Format Subbar labeled xxx at the start of the menu subset and end the subset by putting a *Format Endsubbar at the end. This allows a command list to be used for both a bar with subbars and to contain a packed series of menus.

Built in Commands : *Menu

Working with Explorer windows

Use the built-in *Menu Explorer command to re-open a folder that you recently used with Explorer.

You must check the Windows Explorer option *"Display the full path in the title bar"* on Explorer's View menu >Options.

You can select the sort order for the menu and optionally specify a path to a file manager to use instead of explorer.

If you then check *"Track Explorer"* on the Setup configuration dialog, Powerpro will remember the last 32 file folders that you open with Explorer, PowerDesk, or 2XExplorer. Activating the command *Menu Explorer displays a menu of these recent folders sorted by path, last accessed, or drive.

Right clicking on the menu with shift or ctrl down will remove the selected entry from the recent explorer list.

PowerPro also puts a text mirror copy of the tracked recent folders in the file explorer_windows.txt. This file can be used for *Keys {filemenu c:\program files\powerpro\explorer_windows.txt} to access the tracked folder names.

To clear the list of explorer windows, use *Exec ClearRecentExplorer.

To create a menu of favorite folders, rather than recent folders, create a command list with a set of commands like this

Command	*Exec
Action	Explorer
Folder	c:\pathto\folder1

Then display it with *Menu Show. You can also add the *Menu Explorer command to this command list to combine the menu of favorite folders with the menu of recent folders.

Instead of *Exec Explorer2, you may prefer

Command	c:\windows\explorer
Parameter	/e,/select, c:\pathto\folder1

Omit the /e for a single pane explorer window (sometimes referred to as a "My Computer" window). This will produce the same result as *exec explorer.

see also

The *Exec explorer command and the explorer.exe command line.

Built in Commands : *Menu

File menus

You can use file menus to store the entries and layout information for menus in a file instead of in a command list. These menu files are displayed with one of:

- the *Menu ShowFile command,
- the filemenu("c:/path/name.txt") function in expressions
- the {filemenu c:/path/name.txt} argument in the keys command.

For *Menu ShowFile, lines take one of these two forms (the only difference is how the label is specified):

```
label="command " parameters !‘work directory ::index,iconfile
@"label" command parameters !‘work directory ::index,iconfile
```

For filemenu, lines take one of these two forms

```
label=text ::index,iconfile
@"label" text ::index,iconfile
```

Lines hold text for keys for the filemenu function or its keys counterpart. Lines hold commands for Menu ShowFile.

Lines can be continued by ending each continued line with ;;+ up to a maximum of 530 characters per line.

Blank lines and any line starting with a semi-colon are ignored.

Here is a description of each item in a line

label	The label which appears on the menu, in double quotes if @ format used and the label contains spaces. Optional.
command parameters	The command and then any parameters to be run (for *Menu ShowFile) The command must be in double quotes if it contains blanks. You can use ‘r to generate a CR to separate multiple commands (note always ‘r regardless of the quote character you set on configuration).
text	The keys to be sent for filemenu.
!‘work directory	If using a command (not text) path to work directory or keywords if command uses the field for keywords. Do not use quotes. Optional.

::index,icon	Zero-based index number (optional) of icon in file followed by path to icon file. Optional field. You must use Properties iconsize for icons to be shown. If you want to omit icons for some items, set those items to have an icons field of ::(none)

example lines

```
Send Name=My name here

Calculating Calendar="Exec CalcCalendar" ::c:/pathto/icons/calendar3.ico

@"myEditor in work/directory" "c:/program files/myEditor" -optionstring ;;+
!‘c:/work/directory::2,c/my library/icons.icl
```

quoting

For *Menu ShowFile, the command must be in quotes if it contains blanks. If you are using *Menu ShowFile to show a file which consists of file paths, then these file paths would usually contain blanks. Rather than putting quotes around every one, you can specify that there are no command parameters and so all commands should be treated as though quoted by specifying

```
autoquote
```

at the start of the file. You can turn autoquote off if needed with

```
autoquote off
```

menu properties

You set the look of the menu as a whole with Properties lines.

Note that you must use Properties iconsize for icons to be shown.

You can use *Format commands to create submenus, specify separators, specify maximum rows per column, conditionally include menu entries. Example

```
@main this on main menu

@mysubmenu Format Startsubmenu

@submenu this on submenu

Format endsubmenu
```

You can include other files, for example common submenus or command Properties information, with a line starting with the include keyword:

```
include c:/path to file/name.txt
```

If the full path is not given, the file is assumed to reside in PowerPro\config folder. Includes can be nested to a maximum depth of 4.

alternative file format

You can also use Menu ShowFile with .ini files following the text file configuration format used for import. The file extension must be .ini.

Built in Commands : *Menu

Properties in file menus

Properties lines are used in file menus to provide the layout information which is provided on the Properties dialog of the configuration program. You can use as many Properties lines as you want and they can appear anywhere and in any order. The Properties apply to all elements in the command list or file menu.

The Properties lines have this form:

Properties keyword value keyword keyword value

Note that only some of the keywords are followed by a value.

list of supported Properties keywords

iconsize	followed by one of: 0, 8, 12, 16, 32, 64, 128, or 256. Default is 0 (no icons)
tooltips	show tool tips (no following value; keyword only)
maxtext	followed by integer 0-255 giving maximum length of label
back	followed by three integers gives red green blue color for background
back	followed by path in quotes gives .image for background
back *	means use desktop wallpaper for background
text	followed by three numbers giving the red, green, blue components of text color
fontname	followed by name of font to use, enclosed in double quotes if it has blanks
fontsize	followed by size of font in pixels
fontbold	set font to bold
fontitalics	set font to italics
fontwidth	followed by integer 0-255 giving "boldness" of font

Note that the fontname must be specified for the other font keywords to be actioned.

examples

Properties back "c:/file/back.bmp" text 128 64 0 tooltips

Properties iconsize 16

Properties fontname "Time New Roman" fontsize 12 fontbold

Built in Commands : *Menu

*Menu Folder

Using the built-in *Menu Folder command, you can show a menu listing the files from a folder with subfolders shown as submenus. Left clicking an entry runs the file; right clicking an entry shows the explorer context menu for that entry. You can run the file using its default action, or you can specify a command to use on the selected file using the format keywords. You can also choose to execute all the files in the folder instead of displaying a menu to select one.

*Menu Folder can process all three types of folders:

- ordinary file folders (either the whole folder or specified files using wild cards)
- virtual shell folders, like control panel, printers, my computer
- special folders like start menu programs, recent files, desktop (these special folders are actually folders of shortcuts, usually under your c:\Windows folder)

configuration

Select the *Menu command and the Folder action. In the *"Enter folders"* box, type the folder to be displayed or browse for it using the find button. You can select special folders from the drop down box. You can display more than one folder by listing the folders with a comma after each folder. You can use the word "Separator" to show a horizontal menu separator. You can use the word "ColSep" to start a new column in the menu.

Example: *Menu Folder c:\textfile,colsep,desktop

shows the files and folders in c:\textfile, starts a new column, then shows the contents of the Desktop.

The *Format Keywords* edit box is used to hold keywords which control which files are displayed and how they are displayed; for example, you can change the sort order and you can change the number of entries per menu column. You can also specify the command to be used for a left click instead of the default file action; or you can specify that all files in the folder should be processed (this is one way to enumerate files in a folder with PowerPro).

You usually do not enter the keywords directly; instead use the find button beside the enter keywords edit box to set these keywords with a dialog.

menu display

You can display the folder as an independent menu, by attaching the Menu Folder command to a hotkey or a bar button, or to an item in another menu without using the embed keyword.

Or you can include a *Menu Folder command in the command list of a menu, so it will be displayed as a submenu of that command list's menu, by checking *"Embed *Menu Folder in outer menu"* in the format dialog. Do not use the plugin-style command format for embedded menus; use menu folder xxx, not menu.folder("xxx").

special processing

You can program your own processing for the files from *Menu folder by creating a command list script called HookMenuFolder. This script will be executed for each selected file with x0 set to the full path to the file, x1 set to any command from the dialog, and x2 set to the folder used to run *Menu folder. Note that x1 is the command you specified before substitution of the selected file (e.g. |s are not processed yet). You can change x0 and x1 in any way. If you set both x0 and x1 to "", PowerPro does nothing. If you set only one of x0 or x1 to "", PowerPro runs the command in the other.

large folders

You can use *Menu Folder to explore a large tree of files and folders, but if there are more than 1000 folders or 13000 files in the folder and its subfolders, you need to use a special approach. You could exclude files or folders using the Format Keywords. Or you can navigate one folder at a time.

examples

Command: *Menu Folder

Folder: Desktop

to display a menu of the shortcuts on your desktop.

Command: *Menu Folder

Folder: c:\work\monthly report

to display a menu of the files in c:\work\monthly report.

Command: *Menu Folder

Folder: c:\work\monthly report\new*.xl?

display only files matching the wild card filename new*.xl?

Command: *Menu Folder

Folder Control Panel, c:\ut\myfiles, Sep, Programs Startup to display a menu of your Control Panel, all files in c:\ut\myfiles, programs file Start up, with menu separator after c:\ut\myfiles.

settings

You can show a tool tip with the full path for a *Menu Folder entry by setting a non-zero value to "Milliseconds before tool tip appears for *Menu Folder on the *Setup >Advanced* dialog.

The *Menu Folder command will try to calculate the appropriate number of entries per menu column based on screen resolution and menu font; if you are unhappy with the choice you can set it with an Advanced dialog option.

Built in Commands : *Menu

*Menu Folder formatting

Use the *Menu Folder command to display the contents of one or more file folders or special folders as a menu, with submenus for subfolders.

You can control the look of the menu, how subfolders are processed, the contents of the menu, sorting, and the command executed by using the dialog which is accessed by pressing the find button beside the edit box "enter format keywords or use find button for dialog".

Although you can edit the keywords directly, it's simpler and safer to always use the dialog accessed by pressing the find button.

controlling menu look

Exclude icons: check to avoid icons on menu. You may also want to check *"Add … to folders"* to easily distinguish folders in this case. Note: You must check *"Show icons for *Menu Folder"* on *Command Lists >Setup >All Menus* to show icons on *Menu Folder menus.

Add … to folders: adds … to folder names making it easier to distinguish them from files if icons are not used.

No file extensions: Removes .xxx filename extensions from files shown in the menu

Exclude files: Only includes folders in menu; useful with "Add explorer entry" to navigate folders and then explore one.

Exclude folders: Shows only files; no submenus or subfolders appear.

No submenus: All files from all folders appear on the top menu.

Add explorer entry: Add an entry to the start of each menu and submenu. Clicking it will open that subfolder in Explorer.

Assign menu mnemonics: Adds up to 36 menu mnemonics 0, 1, …a, …z to top level menu entries to allow them to be easily selected from the keyboard.

Menu position alignment: Select position for menu: at mouse, centered on screen, centered under mouse, aligned with last pressed button (beneath or on top), aligned with last pressed button (to the left or right). Alignment with last pressed button left/right or bottom/top depends on which half of the screen it is in.

Start new column after this number of entries (middle of dialog): starts a new column each time this number of entries is placed in a menu. Use checkboxes to determine whether this applies to the main menu only or all submenus and to control whether a line is drawn between columns. Set to 999 for a single column menu which will scroll under Win98/2000.

Show Tool tips: Tool tips are shown even if tool tip configuration indicates they are to be shown for clip menus only.

Tool tips show .txt files (gray all): Check to show content (first 6 lines) of .txt file in tool tip when mouse hovers over entry in menu. Gray check to show for any type of file.

Limit text to this number of characters: use to limit file names to keep the menus a reasonable width. Use *Setup >Advanced* to specify that tool tips will be shown; the tool tip shows the entire file name.

Default background: Use default background from *Command Lists >Setup*.

submenus and subfolders

Expand subfolders to max depth: If unchecked, or if checked and a depth greater than 0 is specified, submenus will not be created for subfolders; instead, clicking on a subfolder displays its entries in a menu. This option is useful for navigating large menu trees which take a long time to load into memory if all submenus are loaded at once.

Add back entry: Useful with previous "Show subdirectories" option to provide for navigation back up the folder tree.

Embed items in outer menu: If you create a command list to display as a menu and put a *Menu Folder in that command list, then the *Menu Folder menu is not generated until you click on the command when the command list is displayed. If you prefer that the *Menu Folder be generated and displayed as part of the command list manual menu, then check this option.

Make submenus from folders: Useful when you have several folders listed in a *Menu Folder command. Normally, the entries in the folders listed in the *Menu Folder command are added to the main menu. If you prefer, submenus can be created for each entry by checking this option.

Expand folder shortcuts: Normally shortcuts to folders which appear in the folders scanned by *Menu Folder are not expanded into submenus and clicking on them displays the folder contents using explorer. Check this option to expand the shortcut instead as a submenu.

sorting

Use the dropdown to select the sort order and the check box to specify that all folders should be sorted to the start.

specifying files and folders to include/exclude

You can limit the menu to files with certain extensions by including these extensions separated by blanks in the edit box; for example

```
.xls .doc .ppt
```

includes Microsoft Excel spreadsheets, Word documents, PowerPoint presentations. Alternatively, you can omit files by preceding extensions with a hyphen, e.g.

```
-.exe -.dll
```

omits exe and dll files.

You can omit certain folders by listing names separate by commas in the edit box.

zeroing variables

You can zero variables before the menu is displayed or before processing the files with `*all` by listing the variable names, separated by blanks.

specifying another menu item name for add explorer entry

You can enter a new string to use instead of "Explorer" as the menu item name when you have checked *"Add Explorer Entry..."*

specifying command to execute

Powerpro normally runs the file or folder selected from the menu by running the associated command; you can instead specify the command and any parameters by putting the command and parameters in the edit box:

c:\windows\notepad

would cause the file to be opened in notepad. PowerPro will create a command line consisting of the command you specify followed by the selected file enclosed in quotation marks. You can use PowerPro built-in commands too:

*File Delete

in the edit box would delete the selected file.

If you want to run a file path with blanks as a command, enclose it in double quotes:

"c:\program files\myprog.exe'"

Normally, the selected file is placed in double quotes after a space. If you want more control over where the selected file appears in the command line, you have two options: use the _file_ variable or use the character |

Using _file_ The selected file is assigned to the variable _file_ which you can then use in a command, such as

do("notepad.exe", _file_)

which runs notepad on the selected file.

Using | You can also use a | to indicate the desired position:

*script assign x1 length "|" -1

would assign x1 the length of the file path less 1. Note that you must include the quotation marks if appropriate when using |.

Use || to get the folder excluding the file name.

Note that with | and ||, PowerPro does not insert any spaces.

If you have selected *"Add explorer entry"*, you can use another file manager (or any command) instead of explorer by specifying it, e.g.

'c:\program files\2x\2xExplorer' '|*'

Note single quotes which will be replaced by double when run.

Built in Commands : *Menu

Entering format information for *Menu Folder command

You can use the work directory edit box to control the files displayed in the menu. Use the dialog accessed by **...** to set the format keywords; or enter them directly as follows:

Columns

autocol n automatically starts a new column every n entries; this gives the menu a toolbar look (applies to top level menu only, not submenus).

autosoftcol n automatically starts a new column every n entries without including a bar between the columns (applies to top level menu only, not submenus).

autocolall n automatically starts a new column every n entries; this gives the menu a toolbar look (applies to top level menu and submenus).

autosoftcolall n automatically starts a new column every n entries without including a bar between the columns (applies to top level menu and submenus).

Text labels

maxtext n limits text labels to n characters.

omit deletes the phrases in the *"Omit strings..."* edit box on the *Command Lists >Setup >All Bars and Menus* config tab; omit is applied before maxtext.

mne means Powerpro will assign single character menu mnemonics to the first 36 items on the main menu to allow them to be easily selected with the keyboard.

noext means file extensions will be removed from menu item names.

back to use default background from *>Command Lists >Setup*.

Position
offset n1 n2 shows the menu offset n1 characters to the right and n2 characters below the mouse cursor; n1 or n2 can be negative.

Sorting
nosort items will not be sorted.

sortext sort by file extension.

sorttime sort most recently change files first.

Subfolders
folderdots means "..." is added to folder names; this is useful with NoSubDir if you do not use icons in menus.

folderstart sorts menu entries with folders at start.

folderback adds Back (previous folder) entry when NoSubDir specified.

nofolders to omit all subfolders.

foldershortcut expands all folder shortcuts in the menu

foldershortcuts2 expands only those foldershortcuts with names ending in _x.

nosubmenu means all files from subdirectories will be listed in the main menu.

empty means empty folders will be included in the menu (normally, they are excluded).

nosubdir means no subdirectories will be included. The names of subdirectories are still shown; if selected, a *Menu Folder is shown for that subdirectory.

Explorer
nofiles means only folders will be shown and not files; useful with the explorer option to traverse large folder trees.

explorer adds a menu entry "Explore" at the top of all submenus; left clicking on it will open a single-pane Explorer window on the selected directory and right clicking will show a *Menu Folder menu for the folder (useful with nofiles). Uncheck *"Switch to if active"* to allow a new Explorer window to open if explorer is already running. You can use another file manager (or any command) instead of explorer by specifying it with cmd, e.g.

```
cmd "'c:\program files\2x\2xExplorer' '|*'"
```

Note single quotes which will be replaced by double when the command is run.

explorer2 adds a menu entry "Explore2" at the top of all submenus; left clicking on it will open a double-pane Explorer window on the selected directory and right clicking will show a *Menu Folder for the folder.

Icons
noicons to omit menu icons (only works if the Folder Contents menu is not embedded in another menu).

Execution
*all to execute all commands, rather than displaying a menu.

*allclose to close all commands, rather than displaying a menu.

*allclosefoce to force closed all commands.

Embed

embed is used if the *Menu Folder command appears in a menu: it causes the menu entries to be embedded within that menu rather than appearing when the *Menu Folder command is selected (embed must be in lower case).

Position

center to center menu on screen.

under to center menu under mouse.

Tool tips

tiptext shows up to 6 lines of .txt file contents on tool tip.

tipall shows up to 6 lines of any type of file contents on tool tip.

tool tips shows tool tips regardless of configuration setting on tool tip setup dialog.

File date

Putting a number n in the work directory edit box means that only files accessed more recently than n days ago will be included.

Exclude

exclude this keyword should be followed by a list of folders, separated by commas, and enclosed in double quotes. These folders will be excluded. For example, Exclude "c:\window, c:\program files" excludes the Windows and Program Files folders.

Fileman

fileman this keyword should be followed by a string to replace "Explorer". The explorer keyword must also be included.

Extension

To **include** only files with only certain extensions, list the extensions separated by blanks including the initial period.

To **exclude** files with certain extensions, list the extensions to be excluded, separated by blanks, and include a - in front of the period of each extension.

Command

cmd Follow the keyword cmd by the command to be executed, enclosed in double quotes. Use a | to indicate where the selected file is to be placed in the command line or use the _file_ variable with the expression format for commands. If the command to be executed is a file path containing blanks, you must enclose it in single quotes. Use single quotes instead of double quotes throughout.

Zero

zero should be followed by a list of one or more variable names, enclosed in double quotes, and separated by blanks. These variables will be initialized to zero before the files are processed.

examples

autosoftcol 2 offset -15 0 maxtext 5

Start a new column every 2 entries; limit labels to 5 characters, and offset 15 characters to the left of the cursor.

nosubdir .exe 15

Include .exe files accessed less than 15 days ago from main directory

.xls nosubmenu

Include Excel spreadsheets from all subdirectories on one menu.

-.dll -.bak

Exclude dll and bak files.

```
*Menu
Folder
c:\path\to\files
*all nofolder cmd ".DoFiles(_file_)" zero "FCount"
```

runs the script file DoFiles for each file. No menu is displayed. Variable FCount is initialized to zero.

Built in Commands : *Menu

Special folders for *Menu Folder

Using the built-in *Menu Folder command, you can also display a menu of the Special Folders used by Windows. To access special folders, the parameters edit box for this command can contain one or more of the following (separated by commas).

desktop	shortcuts on your desktop
recent	recently accessed documents
templates	standard document templates
personal	personal favorites folder
start menu	start menu entries
programs	menu of all Programs folders (current user for NT4)
programs xxx	menu of Programs folder xxx (e.g. Accessories)
Allprograms	menu of all Programs folders
Allprograms xxx	menu of Programs folder xxx for All Users profile (NT4 only)
AllStartMenu	start menu for All Users profile (NT4 only; note no space after All)
AllDesktop	desktop for All Users profile (NT4 only; note no space after All)

Built in Commands : *Menu

*Menu Folder with a large folder tree

PowerPro limits the menu and submenus shown by *Menu Folder to at most 13000 files and 1000 folders. To access folder trees with more files, use one of the following approaches.

To show an entire disk, select in work directory:

Command: *Menu
Action: Folder
Parameter: c:\
Format: nosubdir autocol 16 folderback

The nosubdir keyword is also selected by checking "Show subdirectory only when parent entry clicked" on the *Menu Folder format keyword dialog.

Shows a menu of all files/folders for top level of drive C; selecting one folder shows that folder as menu. Or, if Shift key held down when selecting from menu, shows entire folder as explorer Window. (Autocol 16 automatically starts a new column in the menu every 16 entries).

Also shows a back entry in each menu to allow you to go back up the folder tree. You can enhance this technique with maxdepth keyword to set the maximum depth of the menu.

Another approach for large directory trees:

Command:	*Menu
Action:	Folder
Parameter:	c:\
Format:	explorer nofiles

Shows a menu of all folders for drive C with single explore entry in the menu for each folder. Left click on this entry to show files for that folder in an Explorer window.

The first technique shows the menu faster, but requires clicks to go up or down the folder tree.

Built in Commands : *Message

*Message command

Use the *Message n message text command to display a message.

If you assign this command to a scheduled alarm, the message will appear as a reminder at the specified time. You will be able to update the interval until the message is shown again, or discard the message, or create a copy of the message to be shown again after 5/15 minutes while the original message is reset to be shown again at the standard interval.

You can automatically close the message box after n seconds by including the number n with the command.

Be careful with *Message xxxx; if xxx starts with a number, PowerPro will use the number as the number of seconds to keep the message box open rather than as the message.

If you find that the *message box is not taking the focus when you want it to, try using the **On Top** checkbox or putting *takefocus at the end of the message.

See also

To pause a script with a message, see wait message n,expression and wait message expression described in the "Other wait commands" section.

Built in Commands : *Mouse

*Mouse command

sending a sequence of mouse clicks and moves

You can use the *Mouse command to send a sequence of mouse clicks, mouse positions, and mouse moves to the active window. You can combine alt, win, shift, ctrl with these mouse clicks.

The parameters field of this command contains a series of two letter commands which indicate the mouse actions to perform. The commands which move or position the mouse are followed by two numbers giving the move or position value in pixels.

Here are the commands. You can use either the long form (e.g. leftclick) or two letter short forms (e.g. ll):

leftclick	lc	left click (both left down and left up)
leftdown	ld	left down
leftup	lu	left up
leftdouble	ll	double click left (note: lc lc will not work)
middleclick	mc	middle click (both middle down and middle up)
middledown	md	middle down
middleup	mu	middle up
middledouble	mm	double click middle (note: mc mc will not work)
rightclick	rc	right click (both right down and right up)
rightdown	rd	right down
rightup	ru	right up
rightdouble	rr	double click right (note: rc rc will not work)
save	sa	save current position
restore		restore saved mouse position
alt	al	reverse alt key (i.e. press if up, release if down)
shift	sh	reverse shift key
win	wi	reverse win key
ctrl	ct	reverse ctrl key
move x y	mo x y	move mouse x pixels right, y down (x or y can be negative)
screen x y	ab x y	set mouse to absolute position x y (absolute means 0 0 is top left of screen)
relative x y	re x y	set mouse to relative position x y (relative means 0 0 is top left of active window)

alternative syntax form

instead of: *mouse re 20 50 rc

you can use win.mouse(expr)

where expr must resolve to one or more of the Mouse parameters listed above.

example: win.mouse("re 20 50 rc")

or: myvar = "re 20 50 rc"

win.mouse(myvar)

examples

Command *mouse

Parameter ll

sends double left click.

Command *mouse

Parameter ctrl leftdown leftup ctrl

sends ctrl-left click.

Command *mouse

Parameter re 20 50 rc

position mouse at 20 50 with respect to active window then sends right click

Command	*mouse
Parameter	ld mo 30 -40 lu

sends left down, move 30 right, 40 up, left up (e.g. will draw a line in MS Paint).

Command	*mouse
Parameter	ab 40 60 lc
More cmds	*keys abc

moves mouse to absolute position 40 60, sends a left click, then sends letters abc

Built in Commands : *Note

*Note command

purpose

Use PowerPro's Notes to capture any type of text: reminders, scripts, text for copying into other documents, and so on. Notes can hold up to 64K of text. Notes can be placed in categories. Notes can be assigned a date and will be shown automatically by PowerPro on that date. You can set the color and font of notes, you can resize and hide notes, and you can drag and drop selected text or files from notes to other windows from windows to notes.

commands

*Note Open params	Create new notes or open existing notes
*Note ShowHideOpen	toggle: Hide or Show open notes
*Note OpenCategory cat	Open all notes in a category
*Note OpenOneFromMenu cat	Open one note in a category
*Note CloseCategory cat	Close and save all notes in a category
*Note ShowOpen	Shows all hidden notes
*Note OpenMenu	Shows a menu of all open notes. Brings selected one to the top.
*Note OpenToday	Shows notes with date categories today or before
*Note DeleteOpenCategory	Deletes open notes from a given category
*Note Search	Shows a dialog allowing you to search the last active note. You can also type Alt-S into a note to show or switch back to this dialog.

When you create a new note by executing *Note Open, you can optionally set its colors, text, tab size, source file, and category. To access these features, use the "find" button to the right of the edit box when configuring the *Note command.

Notes menu

Right click on a note text or use the Win key to see a configuration menu which lets you set the note category, set note color, set note tab size, close the note, run the entire note as a PowerPro script, and perform other functions. You can use this menu to fix the notes opening position and size: when the menu option is first checked, the current note size and position is remembered and the note will re-open there after it is closed.

Notes usage

If the option is selected on *>Setup >Advanced >Notes*, double click anywhere in the note to run the double clicked line as a command (else double click selects word). Use ctrl-double click to run the clicked item and the next one.

Resize a note by dragging its border and move a note by dragging the caption.

H Hide a note by clicking on the **H** in the caption or by showing menu with Apps key (beside right Ctrl) and pressing H. Show hidden notes with *Note OpenMenu or *Note OpenShow.

X Close the note by left clicking the **X** at the right of the caption. Left clicking the X closes and saves; right clicking the X closes and deletes. You can also close with Alt-F4. Use *Setup >Advanced >Notes* to control whether PowerPro prompts before deleting notes.

A Select all text in a note by clicking the **A**. Left click and drag from the A to drag-and-drop all note text.

R Rollup a note to its caption by clicking **R** or using Rollup selection from right-click menu or typing Alt-R. Reverse rollup status by using R again. Rolled up notes normally show only the caption, but you can show more by setting height using *Setup >Advanced >Notes* You can reverse the rollup state of all open notes in a category with *Note RollupCategory. You can also use this command to unroll all notes in a category by preceding category name with a plus (+) and to rollup all notes by preceding the category name with a dash (-).

configure Notes

You can control the default size and position, color, and tab size, font of newly opened notes with the *Setup >Advanced >Notes* dialog. This dialog also controls whether PowerPro closes open notes when it is shut down and whether PowerPro shows notes assigned a date category before today. In addition, this dialog lets you control whether R or H are shown in the caption, whether the caption shows the first line of text or the category, and whether rolled up notes are temporarily unrolled with the mouse stops over them for more then half a second. You can also specify whether notes appear on all virtual desktops and whether shown notes get the keyboard focus. You can specify that the right click menu for notes should be split into two columns. Finally, you can specify that note file names should not include date and time created; however, this means that notes with the same first line will be overwrite each other.

showing Notes when specified programs or documents are active

You can arrange for notes to be shown only when a program or a document is the foreground window. To do this, you first check *"Show !captionlist category matching active"* on the *Setup >Advanced >Notes*. You should then restart PowerPro. You then have to create and assign a special category for notes you want to depend on the active window.

The category name must start with an exclamation mark (!) and take the form: !captionlist where the caption list is set to match the programs or documents for which the note is to be shown.

Document matching assumes the document name is placed in the window caption and the caption list is used to match that caption. Use a + instead of an * for the caption wildcard (since categories are stored as folder names and Windows does not allow * in folder names).

For example, notes with category: !=winword,=notepad
would appear whenever MS Word or Notepad programs were active.

For example, notes with category: !Microsoft excel+mysheet.xls
would appear only when a caption starting with Microsoft Excel and ending in mysheet.xls appeared on the active window.

accessing saved Notes by name

Saved notes are stored in a file named by the first line of the note followed by the date and time to make the note file name unique. The files are stored in the note folder under your PowerPro folder, with a subfolder equal to the category name.

If PowerPro is running, you can double click on a note file in Explorer to show the note. You can also open a given note in category "MyCat" with first line starting with "first line of text" by running the command

*Note

OpenCategory

MyCat\first line of text*

where MyCat is the note category and first line of text is the start of the first line of the note's text. For example, you could assign this command to a scheduled event to show the note at a certain time.

more Notes uses

You can use notes for future ToDo's by assigning a date category to them and configuring PowerPro to show notes from today or before when it starts (using >Setup >Advanced >Notes) or by running *Note ShowToday (e.g. in a StartUp alarm).

You can capture clipboard entries or automatically create notes for drag/dropped text by using the autoclone note category; click here for details.

summary of usage

The following list summarizes the user interface to notes and how to use it:

Create new note
Execute *Note Open from bar, menu, hot key, tray icon, etc

Create notes in different categories each with own color
Configure a *Note Open cat for each category, using *"find"* button to assign color and category.

Copy note text to/from other windows
Select text and drag and drop. Or use right click menu for copy/paste.

Select and copy all text
Left click **A** at upper left of note. Drag to copy all text.

Open a note at an exact time
Use Scheduler to open a specific note at a specific time by scheduling the command *Note OpenCategory cat\start of note* where cat is the note category and start of note are the initial characters of the first line (up to 63 characters).

Close note and save it
Left click **X** at upper right of note or right click note text and select close from menu.

Close note and delete it
Right click **X** at upper right of note, or right click note and select *delete*, or use explorer to locate a note's file in its category folder and delete it.

Set category of note
Specify initial category in *Note Open cat, or right click note and select *Category* entry.

Create new category
Right click note and use *Category New* or use explorer to create new subfolder of Notes folder (under PowerPro main folder).

Resize note
Drag border or corner.

Hide note
Hide note with **H** button; reshow with *Note OpenMenu or *Note ShowCategory, or show menu with Apps key and press H.

Show hidden note
Execute *Note MenuOpen and select note or use *Note ShowOpen.

Delete note
Open note and right click **X**, or use explorer, or right click any note and select *"Delete from Menu"*.

Built in Commands : *Note

Auto cloning Notes

You can use the **autoclone** category with *Note to automatically capture clipboard text in new notes or to create cloned notes for text which is drag/dropped onto a note. This lets you create a set of notes with one piece of text per note so that you can easily rearrange or copy a set of captured pieces of text.

To use either for these features, you must check the appropriate checkbox on *Setup >Advanced >Notes*. For automatic clip capturing you must also enable **clip capture** on the *GUI control* tab. Finally, there must be a visible note with category "autoclone": you could use a hot key, bar button etc to create this initial note using a *Note Open command with category autoclone and the desired position, size and colors.

If you drag/drop text to a note with category autoclone, PowerPro will check to see if the note is empty. If it is, the text is pasted into the note. If it is not, PowerPro will clone a new note and put the dropped text in that new note.

If you activate autocloning of notes for clip capture, and you put text on the clipboard when an autoclone note is visible, PowerPro will check to see if this note is empty. If it is, the clipboard text is pasted into the note. If it is not, a new note is cloned and the clipboard is pasted to this new note.

Each cloned note is placed directly beneath the preceding one so for best viewing you should consider setting a small height and initial top right or top left position on the first *Note command. You may also want to set it as "always on top".

You can use the **A** box at the upper left of the note to drag drop the text from autocloned notes. You can also use standard copy operations by right clicking the note.

When you are finished with a set of autoclone operations, you can delete all open autoclone notes with *Note DeleteOpenCategory or by right clicking a note and selecting *Delete Open this Category*.

Built in Commands : *ScreenSaver

*ScreenSaver command

Use the *ScreenSaver command to start, stop, enable, disable, or change the screen saver. You can also change the saver from the media dialog.

The *ScreenSaver actions are:

Enable	enables the saver
Disable	disable the saver
TempDisable	disables the saver until the mouse is moved. The TempDisable command is normally used with a screen corner hotkey. Moving the mouse to the hotkey screen corner which activates the command will disable the saver until the mouse is moved again.
Start	starts the saver
Stop	stops a running saver
Change	changes saver to a saver (.scr file) in same folder, either random or sequential
ChangeTo file	changes saver to specified file

ChangeTimeout	changes saver timeout to specied value (in minutes)
ChangeRestart	set, clear, or reverse setting of restart saver when changed setting on GUI Control dialog

Built in Commands : *Shutdown

*Shutdown command

shutdown Windows or PowerPro

Use the *Shutdown command to exit PowerPro or Windows. When you configure this command, you can also use checkboxes to specify whether a confirmation dialog should be shown, and whether open programs should be forced to close (possibly losing information) for a Windows shutdown.

The *Shutdown actions are:

PowerPro	PowerPro exits. If command list PProShutdown exists, it is run.
Dialog	Shows the windows exit dialog
Reboot	Shuts down window and reboots system
Restart	Shuts down system with warm windows restart
Logoff	Logs off current user
Windows	Shuts down windows
Suspend	Put computer into suspend/standby power mode
Hibernate	Put computer in hibernate power mode
LockWorkStation	Sign off user and lock workstation

Built in Commands : *Skin

Creating skins

This topic and the following two topics tells you how to create a skin.
You may find it convenient to print these three topics for study and reference.

Note: *Skin is not a PowerPro built in command to be used in the normal way in command editing dialogs or in scripts. *Skin is one of several special commands relating to skins; these are only used in "skin files" as described in this and the following topic.

skins only rarely needed

Skins were originally implemented in PowerPro to give greater flexibility than was available from using the configuration dialog. However, for versions 4.6.03 and later, many of the formatting features that originally required skins have now been implemented in the configuration dialog. For example:

Non-square bars can be implemented using .png background file with transparent sections.

Global settings for different colors can be specified for buttons when hovered over and when pressed (and cl function allow these to set individually for each button).

A different font per button can now be set.

Hence skins are only required in rare circumstances: for example, if you want to use sounds or special cursors with buttons.

general structure of skins

A skin consists of a skin .txt file and associated image, cursor (.cur or .ani), font (.ttf or .fon), and wav files. All these should be installed into the same folder.

The skin text file consists of a series of lines of the form:

```
name keyword value keyword value keyword value
```

The name is one of the words *Skin, *Bar, *Font, *Buttondefault, or *Button.

The first line in a skin .txt file must start with the name *Skin. The keywords depend on the name. Keywords can appear in any order. The values depend on the keyword. Some keywords are not followed by values. If a value contains blanks, the value must be enclosed in double quotation marks. You can extend a name entry to the next line by ending a line with a dash (-). For example,

```
*Bar height 200 width 150 -
shape "background for panache.png"
```

sets the bar with keyword height, value 200, keyword width value 150, and shape to "background for panache.png".

Blank lines and any lines in the .txt file starting with a semi colon (;) are ignored.

The case of letters in keywords is ignored: e.g., *SKIN or *Skin or *skin are all the same.

Many keywords are followed by numbers, including numbers used for colors. These numbers are assumed to be based 10 unless preceded by the letter x, in which case they are assumed to be hexadecimal. For example 254 is the same as xfe.

Some keywords use a color value. Colors are specified as either one integer or as three integers between 0 and 255. The three number format color has Red, Green, and Blue components, each between 0 and 255. Or you can use any HTML color values as a single integer. If you use three integers, the list of three numbers must be in double quotations: e.g., "244 0 0" is bright red. You can use the Paint program's *Colors >Edit Colors >Custom colors* dialog, to see various colors and their Red Green Blue components. Or you can use many available HTML-support programs to find the single integer. Remember to put the letter x in front of any hexadecimal values. For example, "255 255 255", "xff xff xff", and xffffff all specify white.

Skins files will often refer to other files: image files, wav files, font files (.fon or .ttf), cursor (.cur or .ani) files. These should be located in the same folder as the skin and be referred to by filename only, without any path. You must always include the file extension (e.g. .bmp).

When building or modifying a skin, you can quickly re-apply a new skin text file by ctrl-right clicking a bar and selecting "Re-apply Skin" from the menu.

layout of skin .txt files

Skin .txt files follow this structure

```
*Skin keywords values
usage notes structured as any number of lines with any text
these lines describe the usage of the skin
they are reproduced in the Set Skin dialog in the list box at the bottom of the dialog
*Bar keywords values
*Font 1 keywords values
*ButtonDefault keywords values
*Button id 0 repeat "count columns h.gap v.gap" keywords values
```

The *Skin line must be the first line. It is followed by a usage notes.

Next comes the *Bar line to give the overall shape of the skin.

Up to 4 *Font lines can optionally be provided to define fonts for use on buttons.

*ButtonDefault lines can optionally be used to set default characteristics for buttons.

Finally, a series of *Button lines appear to define position, size, shape, cursors, sounds, etc of buttons. Often (but not always) a single *Button line will be used to create an array of buttons by using the repeat keyword. *Button lines refer to the corresponding command list items in the pcf file through the id keyword. You should make sure there is a *button command for each item in the command list, usually by using a large repeat value on the last *Button command.

models of skin configurations

When you build your skin .txt file, you must have a model of the corresponding configuration (pcf file) in mind. Two common models are the button array model and the section/subbar model.

the button array model

This consists of an *Info button (optional), followed by an array of buttons which are used to run commands. Its skin .txt file would look like this example:

```
*Skin
This skin file uses the button array approach.
The first command list item should have an *Info label.
*Bar
*Button id 0
*Button id 1 repeat "1000 6 2 3"
```

The first *Button command gives a special layout used to display information (using *Info). The next example *Button command uses the repeat keyword to create an array of buttons with 6 buttons per row, 2 pixels between buttons horizontally, and 3 pixels between buttons vertically. The repeat count of 1000 is a large number chosen to process all remaining buttons on the command list. See file SkinTemplate1.txt in PowerPro folder for a template of this skin .txt file structure.

the section/subbar model

This is used to create a series of section buttons; pressing any section button displays a set of command launch buttons. Subbars are used on the pcf file for the button in each section.

Using subbars: See Sections/subbars for skins for more information on configuring the section/subbar model in the pcf file. The *Skin .txt file has this structure:

*Skin

This skin file uses the section/subbar approach.

The first command list item should have a *Info label.

The next command list items should provide subbar selection buttons.

A series of subbars corresponding to the selection buttons should appear at the end of the command list.

*Bar

*Button id 0

*Button id 1 repeat "toSubbar 1 2 0"

*Button id next repeat "1000 1 2 0"

The first *Button command gives the layout for a *Info button. The next *Button command gives the layout for the selector buttons; the command list items corresponding to these buttons should either be ordinary command buttons or buttons which select subbars. The repeat "toSubbar…" keyword says that this *Button layout should apply until a *Format Subbar is encountered. Then the *Button id next line will be processed, and it will apply to the subbars which end the command list. PowerPro automatically arranges to format the subbars so that they all start at the button position given by this *Button command. See file SkinTemplate2.txt in PowerPro folder for a template of this skin .txt file structure. (LaunchKaos skins use the section/subbar approach).

*Skin line details

The *Skin line must be the first line in all skins text files and can include these keywords and associated values:

Keyword	Purpose and Value [default]
author	Followed by author name (in double quotes if it contains blanks) [none]
created	Followed by created date [none]
title	Followed by any text for title [none]
thumb	Followed by name of .bmp file to use as thumbnail display in skins configuration dialog [none]

The *Skin line is followed by a set of text lines with any text which gives usage notes for the skin.

*Bar line details

The *Bar line follows the usage notes.

If you want to bar to take on the shape of the background bit map, you specify the shape keyword followed by the .image file name as the value.

In addition, when drawing the bitmap for non-rectangular shapes, you must use the transparent color magenta "255 0 255" for the portions of the bitmap which you do not want to appear on the screen. Be careful to use a pure color for the transparent portions; many advanced painting programs will use anti-aliasing which will mix colors together. MS Paint uses only pure colors.

Bars using "shape" are never resizable. Use the "background" keyword instead of shape to allow resizing (depending of course of the pcf settings).

PowerPro shape bmps are compatible with the bmps created for the LaunchKaos program; you can find many such bmps at www.skinz.org.

The following table summarizes the keywords and values for *bar. The default value gives the value used if the keyword is not present.

Keyword	Purpose and Value [default]
shape	The shape keyword is followed the filename of the image which determines the shape, size, and look of the skin. [none]
background	Followed by file name for .image for background. Does not affect the bar size or shape. Use either Shape keyword or Background keyword, but not both. *none*
marker	Followed by a image file to show when the bar is hidden if the pcf file includes the option "Marker" for autohide bar. [none]
width	Width of bar in pixels. Ignored if shape specified. [none]
height	Height of bar in pixels. Ignored if shape specified. [none]
sound	sound file to play when bar is first shown. [none]
soundshow	sound file to play when bar shown after autohide. [none]
soundhide	sound file to play when bar hides. [none]
cursor	Cursor to use if no button cursor applies (following value must be .cur or .ani file). [none]
backcolor	Background color. Ignored if background bmp specified. [gray]
otherback	Default background bitmap file for menus and other bars (to give menus and other bars compatible look to skinned bar); followed by .bmp file name. [none]
othertext	Default text color for menus and other bars. [none]
minmenuwidth	Minimum menu column width in pixels. [none]
maxmenuwidth	Maximum menu column width in pixels. [none]
menuwidth	Fixed menu column width in pixels. [none]
menuheight	Fixed menu item height in pixels. [none]
menuindent	Fixed menu item indent in pixels. [none]
menusepcolortop	Color of top line of separator. [system color]
menusepcolorbottom	Color of bottom line of separator. [system color]

Example:

```
*Bar shape "fancy shape.bmp" marker mymarker.bmp cursor mycursor.cur
```

creates a bar with shape given by "fancy shape.bmp", a background for the marker window given by mymarker.bmp, and the default cursor mycursor.

*Font line details

The *font line can be used to create up to ten fonts to be referenced in *Button and *Buttondefault lines. The word *Font must always be followed by a space then a single digit 0, 1, 2, ..., 9 to specify which font is being defined. Note that font 0 is predefined to the font set for the command list by the .pcf file, but you can override this font, if you want.

Keyword	Purpose and Value [default]

name	Font name, as it appears in a font dialog. [Arial]
size	Followed size, as it appears in a font dialog. [10]
install	Followed by name of .fon or .ttf file to install; only needed if you are not using a standard Windows font and you include a font file with your skin. [10]
weight	A number in the range 0 through 1000. For example, 400 is normal and 700 is bold. [400]
escapement	A number in the range 0 through 3600. Specifies the angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text. [0]
orientation	A number in the range 0 through 3600. Specifies the angle, in tenths of degrees, between each character's base line and the x-axis of the device. [0]
charset	Need for some non-English fonts; followed by a number between 3 and 255. [400]
bold	Same as specifying weight 700. [N/A]
italics	Selects italics font. [N/A]
underline	Selects underlined font. [N/A]

Some charset values are: 128 for JIS, 129 for HANGEFUL, 177 for Hebrew, 178 for Arabic, 161 for Greek, 162 for Turkish, 163 for Vietnamese, 222 for Thai, 238 for East European, 204 for Russian, 186 for Baltic.

Escapement and orientation may not work with all fonts or all versions of Windows. Try experimenting with each separately until you get the effect you want.

Example

```
Font 1 name "Times New Roman" size 12 italics
```

sets font 1.

*ButtonDefault line details

The *Buttondefault line uses the same keywords and values as the *Button line, described below. It provides default values for all keywords for any button commands which appear after the *Buttondefault line in the skin .txt file. You can use many *Buttondefault commands throughout the skin file to change the defaults.

Example:

```
*ButtonDefault textcolor "0 0 0" Facebmp buttonback.bmp pressbmp "button pressed.bmp" height 20 font 1
```

sets the default background and pressed bitmaps, default font number and the default height. If these keywords are omitted from following Button lines, the defaults will be used.

If you want to stop using defaults from *ButtonDefault, use

```
*ButtonDefault reset
```

to remove all defaults.

*Button line details

*Button lines give the size, position, and appearance of the button. A *Button line can refer to one or more items in the command list.

Keyword	Purpose and Value [default]
id	Must always be the first keyword, to specify the button to work with from the command list; the keyword can be followed by one of these three types of values:

- a number specifying the item number in the command list; the first item is item 0. Hidden items and *Format items are included when determining the item number.
- the word next for the next button, skipping any buttons with left command *format
- any other string, which specifies the id for the button, or if the id is not found, then the first item in the command list which has an item name beginning with the same characters as the string

[Default is next]

left	Position of left of button, relative to top left of bar, in pixels. Use 0 for top left of bar. [Default is right of previous]
top	Position of top of button in pixels relative to top of bar; top of bar is 0 and lower positions have higher numbers. [none]
width	Width of button in pixels use pcf width or text [icon width]
height	Height of button in pixels. [pcf height or icon height]
no3d	No special drawing effects are used when the mouse hovers over the button or when the button is pressed (not followed by a value). You would normally specify this keyword if you specified the pressbmp or hoverbmp keywords. [N/A]
notext	The item name text is not shown. This could be used, for example, to relate the button to the pcf using the item name and idname but not show the item name text. N/A
textover	Text is only shown if mouse over button (no following value). [N/A]
iconover	Icon is only shown if mouse over button (no following value). [N/A]
soundhover	sound file to play when mouse moves over button [none]
soundpress	sound file to play when button is clicked [none]
cursor	cursor to use when mouse over button (following value must be .cur or .ani file, or a standard name Ibeam, cross, help, wait, no) [none]

font	Number of font (0 to 9). [0]
text	Text color when mouse is not over button. [pcf setup]
texthover	Text color when mouse is over button [none]
textpress	Text color when button is pressed [none]
textall	Sets all of above three text colors [none]
textpos	Follow by right for right justify or center for centering, or top for top-alignment, or multi for multi-line text broken at end of words, or bottom for bottom-aligned text. You can use textpos more than once to specify both horizontal and vertical justification, e.g. textpos right textpos bottom. [none (left, vertical center)]

textleft	Followed by number giving the offset to the text rectangle from the left of the button. Note: textpos setting gives the justification of text within this rectangle. [0]
textwidth	Followed by number giving the width of the text rectangle. Note: textpos setting gives the justification of text within this rectangle. [width of text]

texttop	Followed by number giving the offset to the text rectangle from the top of the button. Note: textpos setting gives the justification of text within this rectangle. [0]
textheight	Followed by number giving the height of the text rectangle. Note: textpos setting gives the justification of text within this rectangle. [height of text single line]
iconpos	Follow by right for right of text, center for center if no text, or above for above text [none (left of text)]
icontop	Offset to icon from top of button; overrides iconpos.
iconleft	Offset to icon from left of button; overrides iconpos.
face	Background color when mouse is not on button. Omit to let base bmp from bar show through for button. [pcf setup]
facehover	Background color when mouse over button [none]
facepress	Background color when button is pressed [none]
faceall	Background color for button for all cases [none]
facebmp	Background image file when mouse is not on button [none]
hoverbmp	Background image file when mouse is over button [none]
pressbmp	Background image file when button is pressed [none]
allbmp	Sets face, hover, and press to same image file [none]
repeat	Creates an array of evenly spaced buttons. Must be followed by four numbers in quotes: the first number is the repeat count, the second gives the number of columns, and the third and fourth give gap in pixels between buttons horizontally and vertically. Use ToSubbar for first number to repeat until *Format StartSubbar encountered in pcf [none]

notes

The first keyword must always be id. If a string is specified as the following value, the *Button line refers to the first item in the command list with a name which starts with the text characters. If a number is specified, then the *Button line refers to the item number given by the value. If next is specified, the *Button line refers to the item following the item used by the previous Button line. (The first *Button line defaults to item 0). *Format commands are skipped by Button lines. But note that when counting items for id followed by a number, all command list items are included.

You should always specify top, left, width, and height.

You can make the same *Button line refer to multiple command list items by using repeat. The id specifies the first item; subsequent items in the command list for the repeat follow this initial item in the command list. The repeat value must be at least 1 or use ToSubbar to make repeat apply until *Format Subbar encountered.

There can be more button commands than items in the command list; such button commands (or repeats) are ignored but this is not an error. Also, if the id refers to a command list item which does not exist, the Button command is ignored but again this is not an error.

If you specify both a color and a image for background, hover, or press, the image takes priority.

To let the base background from the bar show through, omit both face color and facebmp.

See the sample skins for examples of *Button usage.

Built in Commands : *Skin

Using skins

purpose of skins

If you want more options and flexibility in specifying the look of a bar, you can use skins. Skins give you more control than the formatting options in the configuration file: skins let you specify that a bar uses a background image drawing of any shape and they let you specify the size, position, font, text/icon position, and look of any button. You can also use skins to specify cursors and sounds for buttons.

Skins are defined by files which you keep separate from your pproconf.pcf file (which is in your Powerpro folder and stores configuration data). The pcf file specifies what you want buttons to do. The skin files specify the look of bars and buttons. But there is still an interaction between look and configuration meaning that some skins expect certain features in your pcf file and that not all skins work with all pcf files. For example, many skins display time, date or other data and expect you to define a *Info button at the top of your command list. Or if you use subbars, you will usually need to have all the subbars defined in a series at the end of the command list. Certain skins works best with the section/subbar configuration, described below. The usage notes on the skins configuration dialog will describe constraints on the pcf configuration for the skin.

installing skins (including the sample skins)

Before installing your first skin, make sure that you create a folder called Skins under your main powerpro folder. For example, if PowerPro is stored in C:\Program Files\Powerpro, create a folder called C:\Program Files\PowerPro\Skins using explorer.

Skins are distributed as zip files. To install any skin, create a subfolder of your skins folder with the same name as the zip file (or anything else if that name is already in use). Then unzip the skins .zip file into that subfolder. You will find sample files SkinSample Kaos1, SkinSampleZlk, SkinSampleNewbie, and SkinSampleKaos in your PowerPro folder and can unzip these to create sample skins to experiment with.

Note: you should be able to use Kaos1 and zlk on your current bar, but before using sample Kaos or Newbie, please review the help on section/subbar configurations below.

Sample skins are based on those created for the LaunchKaos program. See www.PocketKaos.com for more information on this program and www.skinz.org for skins which can be used as a basis for creating PowerPro skins.

See Creating skins to learn about how to create new skins for yourself.

using a skin

To configure a skin, Ctrl-right click any bar and select "Configure Skin for Bar" from the resulting menu. You can also access this dialog by pressing the "Set Skin" button on the command list configuration dialog.

Then select the skin file you want to use from the drop down at the top of the dialog. Review the usage notes at the bottom of the dialog for information on how to set up the command list items in your bar for best use of the skin. You will also be able to use the checkboxes to enable or disable the sound, menu/other bar background, font, and marker bitmap features of the skin, if the skin uses these features. If you check "use menu/otherbar background" to use the skin's background for other bars and menus, you can still override the background for individual bars or menus by putting none in the >Command List >Properties background for these bars or menus.

Although skins control the overall look of your bar and buttons, you still set the bar position and the autohide approach using Command List >Properties.

The "last for setup" feature of active buttons does not apply if you use a skin; the settings provided by the skin take priority.

Each time you reconfigure a skinned bar, it will take a few seconds for the bar to reappear.

section / subbar configuration

Some skins work best with a section/subbar structure in your pcf file. This will be indicated in the usage notes. The Kaos and Newbie samples use this structure.

This skin structure is meant to show a command list which has a series of subbar selection buttons at the start and a series of subbars at the end. The idea is to use each subbar for a category of commands or documents and to use a subbar for each category.

To see a sample of this type of configuration and to test Kaos and Newbie, ctrl-right click on any bar, select "Change configuration" menu item, and then select "subbars" from the resulting submenu.

This will display a new bar with subbar selection buttons at the start and subbars at the end. To see its layout, Ctrl-right click bar and select configure from menu. For best effect, you should view this bar with a skin by pressing Set Skin from command list configuration and selecting Sample Kaos or Newbie.

When you are finished with the demo, ctrl-right click on the bar and select "Change configuration" menu item, and then select "pproconf" from the resulting submenu to restart your configuration.

Built in Commands : *Skin

Sections / subbars for skins

Some skins configurations require a section/subbar approach to the pproconf.pcf file. This type of bar has a set of (so called) section buttons to select a subbar; each subbar consists of a set of launch items. For example, the LaunchKaos program and its skins use this approach. The idea is to use a subbar to group launch items with a common purpose, e.g. editors or internet, and use the subbar selection button to select that category. In general the list of items in a command list for this type of bar will look like this:

Name	Command	Action	Parameters
Editors	*Bar	SelectSubbar	@editors_
Internet	*Bar	SelectSubbar	@Internet_
Utilities	*Bar	SelectSubbar	@Utilities_
[maybe other items here]			
Editors_	*Format	StartSubbar	
[items for editors subbar here]			
	*Format	EndSubbar	
Internet_	*Format	StartSubbar	
[items for internet subbar here]			
	*Format	EndSubbar	
Utilities_	*Format	StartSubbar	
[items for utilities subbar here]			
	*Format	EndSubbar	

For convenience, the subbar name in the above example has been chosen to be the bar category label followed by an underline.

You will likely want to check "Show *Bar SelectSubbar as pressed" on Command Lists >Setup which will cause PowerPro to show the selector button corresponding as the visible subbar as pressed.

You can quickly create a subbar and a button for selecting that subbar from the command list configuration dialog by clicking Quick Add, or right clicking the list box and selecting Quick Add, and then selecting "Selector and new subbar" from the menu. The selector is added after any currently selected button in the list, and the subbar is added to the end of the list.

For a sample of such a bar, ctrl-right click on any bar, select "Change configuration" menu item, and then select "subbars" from the resulting submenu. If you have installed the sample skins, you can see how Skin Sample Kaos and Skin Sample Newbie display this pcf configuration. When you are finished with the demo, ctrl-right click on the bar and select "Change configuration" menu item, and then select "pproconf" from the resulting submenu to restart your configuration.

This approach to configuration can take a lot of screen space. If you prefer to use less screen space, you may wish to replace the *Bar selector buttons with a button or hot key which displays a menu of *Bar Selector commands.

Built in Commands : *Timer

*Timer command

purpose

Powerpro has 26 timers that you can control and optionally display as button labels. The timers are identified by the single-letter labels a, b, c, ..., z.

Timers can be used to launch commands at three different times: when the timer starts, when it stops, and at a specified reset interval.

Timers can also be used to track time spent online or using a specific program. You set that up by using the "Timers" dialog. Powerpro can produce a timer log to detail this tracking information.

configuration

You can change timer settings using the Timers dialog; select the Timers tab on the main configuration dialog.

Or you can manipulate timers using these *Timer commands:

*Timer commands

Note: For Start, Stop, StartStop and Clear, you need to enter the single letter identifications of the timers to be affected. You can enter more that one timer, but do not put blanks between the letters of the timers, thus: *Timer Start dgk

*Timer Start a	starts the indicated timer(s)
*Timer Stop a	stops the indicated timer(s)
*Timer StartStop a	start the timer if stopped; stops it if it is running.
*Timer Clear a	zeros the timer(s)
*Timer Autosave n	PowerPro normally autosaves running timers to the pcf file every 300 seconds. This changes the interval at which timers are autosaved to the pcf file, where n is the new number of seconds between saves. Specify 0 to stop autosaves.
Timer Set [+/-/][@/$]a h m s	starts, stops or toggles the timer(s) and sets the value

parameters for *Timer Set

[+/-/*]	If it starts with +, the timer is started; with - the timer is stopped, and with * the timer is toggled. Use of one of these characters is optional: if omitted, the timer state is unchanged.
[@/$]	Next, optionally, comes the single character @ if you want to add, or $ if you want to subtract the value, relative to the current value. Omit the @ and $ to set the timer to an absolute time.
a	Next come the single letter timer IDs of the timers to be adjusted, with no blanks: adgk
h m s	Finally, the new timer value is indicated as three numbers: hours, minutes, seconds, separated by blanks.

examples

*Timer Start dgk	Starts the three timers d g k
*Timer Set +a 0 0 0	Clears timer a and starts it.
*Timer Set +a 0 0 120	Starts timer a at 120 seconds.

*Timer Set be 0 10 20	Resets timers b and e to 10 minutes, 20 seconds leaving their running/stopped state unchanged.
*Timer Set -c 1 0 0	Stops timer c and sets its value to one hour.
*Timer Set @ q 2 3 0	Adds 2 hours and 3 minutes to timer q
*Timer Set $ q 2 3 0	Subtracts 2 hours and 3 minutes from timer q.

further information

A running timer is displayed in the form **hhhh.mm** (hours, then a period, then minutes).

A stopped timer is displayed in the form **hhhhxmm** (hours, then an **x**, then minutes).

To automatically clear a saved timer once per day, set up a scheduled alarm with these characteristics (using timers c and g for example):

Time:	12:01 AM
Interval:	Alarm again in 1 day
Command:	*Timer Clear cg

The *"Ring Missed Alarms"* checkbox on the Setup dialog must also be checked for this to work (unless you start Powerpro each day at 12:01!).

You can use a similar technique to clear timers once per month (ring on first of month at 12:01); that could be used, for example, to keep a monthly running total of time spent online, by checking *"Run when dialog of same name is active"* in the timer editing dialog.

logging timers

You can ask Powerpro to log all timer events in a file. Check the *"Timer Log"* checkbox on the *Timers* tab to log all timers. To log only some timers, check the *Log* check box on the individual timer configuration dialog.

showing timers on buttons

Using the item editing dialog in Command Lists, you can have Powerpro place a timer's value as the label on any button. Use Info timer a (where a is one of the timers, a to z) in the item Name field to indicate which timer is to be displayed.

see also

For more precise timing, try these built in functions:

n64 = perfcount	Returns the Windows performance counter.
n64 = perfreq	The frequency that perfcount is incremented.

These are both 64 bit integers, so you have to use the int64 plugin to work with them. You can use these functions to do very accurate timings.

Built in Commands : *TrayIcon

Working with tray icons from other programs

purpose

You can use the *TrayIcon command to simulate mouse clicks on the tray icons from any other programs . You can also use this command to hide these tray icons and still access the commands by simulated mouse clicks.

You can also access tray icons with *Menu Tray or active buttons.

configuration

Starting with version 4.2.04 of PowerPro, it is recommended that you install PowerPro tray support before using the TrayIcon command. To install PowerPro tray support, use the button on the Configuration Setup tab (if you get an error message, see below for information on tray support installation). It is also possible to use the TrayIcon command without installing tray support; see the last section of this help topic for details.

Once you have installed tray support and rebooted your machine, you can execute a tray icon command with

Command	*TrayIcon
Acton	command
Parameters	caption_list (id)

Or use the new syntax: trayicon.command("caption_list (id)")

Here "command" is one of left, leftdouble, right, rightdouble, hide, show, or dump. The hide command hides the icon from PowerPro and the system tray; show re-shows it. You can still send commands to hidden icons. (You cannot use show in Win Me or earlier).

The caption list is used to pick out the tray icon window to work with. You use the caption list to specify the window via its caption (*caption*) or exe name (=exename) or window class (c=windowclass) for the tray icons hidden window. The (id) is normally not needed. If used, it indicates which tray icon to use for a window in the case where the window controls more than one tray icon.

Tray icons are controlled by special, hidden windows. How can you find out the window corresponding to each of your tray icons? You use the trayicon.dump command for this. Execute the command trayicon.dump (eg from a button) after you have installed tray support and PowerPro will produce a list of all tray icons in a file called trayicons.txt in your PowerPro folder. Each item in the list contains the following information:

exe - the exename of the tray icon window

id - the id number of the tray icon for this window

fl - four flags: h means hidden, s mean shared icon, c means child window, u means Unicode (you can ignore these)

class - the window class for the tray icon window

cap - the window caption for the tray icon window

tt - the tooltip for the tray icon window: to help you identify which tray icon this window is for

Here is a sample line produced by trayicon.dump:

```
exe=ccApp id=200  fl=....  class=NAVAP Wnd Class  cap=Norton AntiVirus  tt=Norton Running
```

indicates a tray icon from program *ccApp.exe*, id is *200*, window class is *NAVAP Wnd Class*, window caption is *Norton Anti-virus*, tooltip is *Norton Running*.

As long as it is unique to the icon program you want to use, the exename is the best item to use in the caption list for the TrayIcon command. For example,

Command	*TrayIcon
Acton	left
Parameters	=ccApp(200)

would left click the icon from the above sample. The (200) is only needed if there are other tray icons from ccApp in the list.

However, sometimes, tray icons use a common exe name like explorer or rundll32. If the id is also duplicated, another approach is needed. In this case, you can try the class or the caption, such as (using the new syntax)

trayicon.left("c=NAVAP Wnd Class (200)") trayicon.left("Norton AntiVirus (200)")

However, note that some tray icon programs change their class or caption each time they run; you should tray the trayicon.dump commands after a reboot to see if this is happening for tray icon if the class or caption approach fails.

You can put an "!" in front of the window identifier in the parameter field to suppress an error message if the window is not found.

Command	*TrayIcon
Acton	command
Parameters	*!*caption_list (id)

configuration without using tray icon support (not recommended)

If you do not use the installed tray icon support, you must train Powerpro on how to access the icon. See below for information on how to do this. You have to train Powerpro once for each icon you want to access. Note that this approach does not work reliably for some tray icons.

Once you have trained Powerpro, you send mouse clicks to the icon with the following command:

Command	*TrayIcon
Acton	click
Parameters	icon_name keystrokes

click is one of left (left click), leftdouble (left double click), etc.

icon_name is the name you assigned to the icon when you trained Powerpro; put it in quotes if it contains blanks.

keystrokes is optional; if present, it is a set of keystrokes to send to a menu resulting from the click (if a normal window results from the click, use *Keys with multiple commands instead).

You can also hide the tray icon with the command:

Command	*TrayIcon
Action	hide
Parameters	icon_name

You can still send mouse clicks to a hidden tray icon.

examples

Command	*TrayIcon
Action	left
Parameters	modem {ad}{ad}{en}

sends a left click to the tray icon named **modem**, and then sends two arrow downs and an enter to the resulting menu.

If the icon_name is not found, you will normally get an error message. Precede the icon name with ^ to avoid the error.

Command	*TrayIcon
Action	hide
Parameters	^icon_name

Built in Commands : *TrayIcon

Tray icon support for menus and active bars

You can access all icons in the system tray on a PowerPro menu or a bar.

Before using this feature, you must install tray support by using the Install Tray Support button on the Configure > Setup tab. (If you get error messages, see below or other advanced information on installation.)

Once the tray support is installed, you use the Menu Tray command to display a menu of tray icons and an active bar to display tray icons on a bar.

For the menu or a button, left or right click to activate the correspond tray icon function; middle click (or left+shift) to access the double click function of the tray icon. You must also check *Command List >Setup >"Right selects its own entry on menus"* for right clicking the menu to work.

The text on the menu or button is set from the tool tip of the tray icon.

If you want to display only certain tray icons on a bar, you can use the edit box on the *Command List >Properties >Active Buttons* tab to specify the exe file name or window caption for the tray icon. You can use the *TrayIcon dump command to find this information. For example, the Windows 98 dial-up tray icon shown when a dial-up is active is shown by the program rnaapp.exe, so putting

#=rnaapp

will show this tray icon only (the # means display of active windows is not affected).

Or as an alternative way of identifying that icon:

- #*connect*

works for the dialup icon both in 95/98 and also in Win2000/XP.

To display all tray icons except those specified, add a ~

#~*connect

will show all tray icons except the dialup connected icon.

If you logoff, you will lose tray icon support. You must reboot to restore it.

You can also use the *TrayIcon command to access or hide individual tray icons.

If you find tray icons periodically stop functions, try using the trayicon.refresh command.

Built in Commands : *TrayIcon

Advanced Information for Tray Icon Support

The following information is intended only for users who want to understand how tray icon support works or who get an error message with automatic tray install. To install tray support, PowerPro must arrange for PProtray.exe to run before any other program.

For Win95/89, PowerPro tray icon support involves installing pprotray.exe to run as a service. This cannot be manually overridden and should not cause errors.

For Win NT/2000/XP, Powerpro arranges to run pprotray.exe from the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\

Before PowerPro tray support is installed, the subkey UserInit in this key is set as follows:

Userinit=C:\WINNT\system32 \userinit.exe,

After PowerPro tray support is installed, this value is set as follows:

Userinit=C:\WINNT\system32 \userinit.exe,C:\Progra~1\PowerPro\pprotray.exe,

The path would change to wherever you have installed PowerPro.

For safety, PowerPro will not install tray support unless if finds UserInit in the expected "before" state. For example, if you installed Windows into a different folder, PowerPro will not install tray support automatically. You can install manually by adding

c:\program files\pprotray.exe,

to whatever is there. Note the comma at the end. To de-install, delete only what you added.

Built in Commands : *TrayIcon

Training PowerPro to recognize tray icons

An obsolete approach to accessing tray icons from other programs using the TrayIcon built-in command is to first train Powerpro to recognize the hidden window and internal codes that this icon uses. This approach does not work for all cases.

1. Make sure the tray icon to be accessed is visible in your tray. It is helpful to shut down other windows, but this is not necessary.
2. Start the configuration dialog and activate the command entry controls for the command list, hot key, alarm, or timer that you are configuring. Select the *TrayIcon command.
3. Press the search button (binoculars) and select *Add...* from the resulting menu.
4. You will get a message box prompting you to left click on the tray icon. Press OK on this message box and then left click on the tray icon.
5. If Powerpro is able to capture the information, you will get another message box reporting success and asking you to help confirm that the information was correctly captured. Press OK and Powerpro will simulate a right click on the icon as a test.
6. If the right click test succeeds, Powerpro will ask you to enter a name for the icon information. This is the icon_name field used in the *TrayIcon command or selected from the drop down in the command wizard.

If Powerpro cannot capture the left click on the icon, or if the right click test is not successful, try again once or twice to ensure that this was not just a transient problem.

Note: If you are using an alternate shell, instead of the default explorer.exe shell, training may not work with the alternate tray module. It may be necessary to run the explorer shell to do the training, then switch back to your usual shell. After that PowerPro should be able to recognise the tray icons when displayed in the alternate tray.

Built in Commands : *Vdesk

*Vdesk command

using PowerPro's virtual desktop features

purpose

Use virtual desktops if you run many programs at the same time and want to reduce desktop clutter. A virtual desktop is a collection of windows which you show and switch-to as a group using the *Vdesk commands. Only windows on the currently active virtual desktop are visible.

Note: When you shutdown Powerpro, all desktops are lost. If you have a set of programs you always run as a group on a desktop, you can create a command list with those programs and then activate the desktop and these programs with the *Vdesk CreateOrSwitchTo or *Vdesk ReplaceByList actions or by using the *Initial list* box in a desktop's Edit dialog.

configuration

You can define and switch between virtual desktops in several ways:

- with the built-in command *Vdesk described below
- or through the Desktops Menu which you access by Shift+right-clicking anywhere on a Powerpro button bar. You can use the Desktops Menu to switch desktops, create new desktops, lock/unlock windows on desktops, move windows between desktops, close and rename desktops.
- or using the *Desktops* tab of the main configuration dialog. Here you can specify an initial name, an initial command list to populate the desktop, a command list to be shown as a bar for the desktop, and a wallpaper file for the desktop
- on the *Desktops* tab, click Setup for general settings about vdesks.

*Vdesk command

You can associate this command with a button, menu item, hot key, and so on. Use the action and parameter fields as follows:

Arrange	Displays a window showing all desktops (see below for more information)
Clear	Clears the selected virtual desktop.
Consolidate	Move all windows to current desktop.
ClearAllClose	Move all windows to current desktop and then closes them.
CloseTo	Move all windows to indicated desktop and then closes current desktop.
Menu	Displays the virtual desktop menu.
Next	Activates the next virtual desktop.
Previous	Activates the previous virtual desktop.
MoveActive	Moves active window to name desktop (which must already exist).
MoveAutorun	Moves last autorun match to named desktop
SwitchMenu	Show a menu of desktops and windows; select one to activate it.
ShowMenu	Shows a menu of desktops and windows; select a window to move it to this desktop
SwitchTo	Switches to the indicated desktop.
New	Creates a new desktop; you can specify its name.

CreateOrSwitchTo	Creates a new desktop named after a command list and runs the commands on the list to populate the desktop. If the desktop already exists, switches to it.
NewFromList	Same as CreateOrSwitchTo
ReplaceByList	Clears the current desktop and renames it to the command list and runs the commands on the list to populate the desktop.
Lock	Followed by a caption list; locks all visible windows matching the caption list.
Unlock	Followed by a caption list; unlocks all windows matching the caption list and leaves them on the current desktop.
Recover	Meant to be used only after a PowerPro crash when windows are hidden on desktops. If the"Save vdesk contents" option has been checked on desktop setup, then executing vdesk recover will show all windows hidden on desktops.
MoveAll	Followed by a source desktop, a destination desktop and a caption list. All windows on the source desktop matching the caption list are moved to the destination desktop. You can optionally use * for desktop name to indicate the current desktop. Put desk names with blanks in quotes. Example: *vdesk moveall otherdesk * =notepad,*explorer* moves all windows matching either =notepad or *explorer* to current desktop.

notes

*Vdesk ReplaceByList will rename the desktop. If you want to add programs to the current desktop without renaming it, you can *Script run a command list which starts each of the programs you want to add.

Use *Vdesk MoveAutorun with autorun command lists to move windows of a specified type to a desktop when the windows first open.

The command *Vdesk Arrange shows all nine potential desktops and allows you to drag/drop windows among desktops, create/delete desktops, and lock/unlock windows. You can access a control menu by right clicking on the Arrange window. The active desktop name is shown in bold; the active (foreground) window is also shown in bold. You can also double click on the list of windows in a desktop to close the Arrange dialog and switch to that desktop or double click on the name of a desktop to switch desktops without closing the Arrange window. If you work with fewer than nine desktops, you can change the arrange dialog's height (but not its width). The window normally has three columns and up to three rows, but if you use fewer desktops, you can use the right click menu to select a 3 by 2 version, which can be shrunk to show just 4 or just 2 desktops.

further information

You can have up to 9 virtual desktops.

You can show the name of the current desktop as a button *Info label.

You can show all windows on all desktops on the Taskbar; control subbars with virtual desktops; specify windows to be locked on all desktops; show a button for the active desktop with a special color, label, or as pressed; by using virtual desktop Setup.

You can initialize the name of desktops, specify the wallpaper for desktops, and specify a command list to be run to initially populate a desktop using the Desktop tab.

It is possible to show a different Powerpro bar for each desktop. Create new bars and start them with the desktop you want them to be associated with (using *Bar Show). Make sure *"All Vdesks"* on the Bar's Properties dialog is not checked.

You can define a command which will display a menu which depends on the currently active virtual desktop:

Command	*Menu Show
Parameter	*desk

will display the menu with the same name as the currently active virtual desktop.

desktop icons and vdesks

If you like using desktop icons (instead of PowerPro menus or bars), here is how to create a different desktop icon configuration for each virtual desktop.

Use *Desktop SaveIcons filename.iconpos which lets you specify a file name to save the icon positions in. Create the desired icon configuration for each desktop in turn and save them in a separate file; move icons you are not interested in for a desktop out of the way in a corner for that configuration.

Then write a command list script to load the icon positions you want. This script should check which desktop is about to become active using a statement like var = desktop and if statements to check for each desktop and then execute a *Desktop RestoreIcons command to restore the icon positions for that desktop. Set this script name on *Desktop >Setup* (script will be executed after a new desktop is loaded).

Built in Commands : *Vdesk

Virtual desktop setup

using the Desktops Setup dialog

Normally, Powerpro only shows windows from the current virtual desktop on the Taskbar. If you prefer, you can arrange to show all windows on the Taskbar and use the Taskbar to switch among desktops by checking the **"Show all windows from virtual desktops on task bar"** box. Shutdown and restart PowerPro after changing this option.

If **"Rerun script with desktop name each time desktop is activated"** is checked, each time you switch to a virtual desktop which is already running, Powerpro will execute the script of the same name for the desktop.

If you activate a window which is on a hidden desktop (e.g. via tray icon), Powerpro can be configured to show and switch to the hidden desktop. If you want this feature, check **"Show desktop if any of its windows is activated"**.

You can specify that PowerPro should move a window which is activated by the "switch to if active" feature by checking **"Switch to if Active moves window to current desktop"**. If you leave this unchecked, then the windows will be copied to the current desktop as well as remaining on hidden ones.

Check **"Show bar button ... as pressed"** to have PowerPro show the button corresponding to the active desktop as pressed. Assign the command *Vdesk SwitchTo xxx or *Vdesk NewFromList xxx to any mouse click on a button to have that button shown as pressed whenever the virtual desktop named xxx is active. You can set checkboxes to use this approach to show the icon only from the button corresponding to the active desktop or the own color only from the button for the active desktop. Note: you cannot use the plugin-style command format for this situation (avoid vdesk.switchto("xxx"))

You can **"Show subbars with the same name"** as the active virtual desktop, on bars which are shown on all desktops, by checking Show subbar of same name as vdesk.

You can specify the name of a command list to be **run before** (or **run after**) each time you switch to a new desktop. The command list can access the desktop name, if desired, with:

*Script if (veskdname=="desktopname") or *Script set v desktopname.
This list is not run when PowerPro starts; use a startup up scheduled event to run the list at startup if you want to do this.

You can specify a caption list of **"windows to be shown on all desktops"**, aka locked windows.

You can specify that PowerPro save a list of desktops in the file vdeskdump.ini to be used by the vdesk recover command in case of a PowerPro crash.

Built in Commands : *Vdesk

The Virtual Desktop Menu

The following items appear on the Virtual Desktop Menu:

List of Defined Desktops
Select one of the desktop names on the menu to show the windows on that desktop.

New Desktop
Hides all the windows on the current desktop and creates a new one. You can name the new desktop with the Rename menu entry, if you want.

Arrange
Shows all nine potential desktops and allows you to drag/drop windows between desktops, create/delete desktops, rename desktops, lock windows on all desktops.

Unlock
Shows a list of locked windows. Selecting one unlocks it. This menu item is only enabled when there are locked windows.

Lock
Shows a list of windows on the current desktop. Selecting a window locks it. A locked window appears on all desktops. This menu item is only enabled when there are windows on the desktop which can be locked. You can also pre-specify locked windows using the *"Show on All Virtual Desktops"* edit box on the *Desktops >Setup* dialog.

Remove From Desktop
Shows a list of windows. Selecting one removes it from the current desktop.

Move/Copy from this
Shows a list of windows. Selecting one causes a menu of desktops to be shown; selecting a desktop from this list moves the selected window to that desktop (hold down Ctrl to copy the window). Only enabled if there is a window which can be moved and there is more than one desktop.

Clear this Desktop
Closes all windows on the current desktop. If the windows only appear on this desktop, the corresponding programs are closed.

Clear All Desktops and Close
Moves all windows to the current desktop and then closes all windows.

Move all Windows to Current Desktop
Moves all windows to the current desktop and closes other desktops.

Clear and relaunch from list
Closes all windows on the current desktop and restarts the programs in the command list of the same name. When closing windows, if the windows only appear on this desktop, the corresponding programs are closed.

Close and move windows to
Closes current desktop and moves its windows to selected desktop. Only enabled if there is another desktop besides the current one.

Rename Desktop
Allows you to assign a new name to a desktop while it is active.

See All/Move/Copy to this
Shows the names and window captions of other desktops and allows you to copy/move a window to the current desktop.

The active desktop name is show in round parentheses, e.g. **(mydesk)**; other desktop names are shown in angle brackets, e.g. **<otherdesk>**. Select a window name to move that window to the current desktop or hold down the Ctrl key while selecting a window name to copy it.

See All/Switch To
Shows the names and window captions of other desktops and allows you to switch to another desktop and activate a window on that desktop.

The active desktop name is show in round parentheses, e.g. **(mydesk)**; other desktop names are shown in angle brackets, e.g. **<otherdesk>**. Select a desktop name to switch to that desktop and activate the last window which was active. Select a window name to switch to that desktop and activate that window.

Start Desktop From List

If a desktop of the specified names exists, switches to it; otherwise creates a new desktop and runs command list of same name to populate the desktop.

Built in Commands : *Vdesk

Initializing virtual desktops

using the "Desktops" dialog

Use the Desktop tab on the configuration dialog to initialize virtual desktops. You can specify an initial name, an initial command list to populate the desktop, a command list to be shown as a bar for the desktop, and a wallpaper file for the desktop.

If you specify a name or an initial command list, the desktop will be created and the command list (if specified) will be run to initially populate the desktop. This only happens when PowerPro starts.

If you specify a command list to be shown as a bar, the bar will be shown each time the desktop is activated. Make sure *"Show as bar"* on the command list configuration is not checked, and make sure *"Show on all Vdesks"* on *Command List >Properties* is not checked.

If you specify a wallpaper file, each time you switch to the desktop, PowerPro will set the wallpaper to the specified file. Use a .bmp file for best performance.

Built in Commands : *Vdesk

Virtual desktops demonstration

Start the virtual desktop demonstration configuration: Ctrl-right click on any bar, select "Change configuration" menu item, and then select "demodesk" from the resulting submenu.

The sample shows many different approaches to desktops; you would not likely want to use all of these at the same time on your configuration. In addition, since every computer has exe files at different locations, the example uses only standard utilities like notepad and regedit.

After starting the sample, there should be a bar in the upper left of your desktop. Click on different gray buttons to show different desktops. Note how the active desktop has a pressed button. This is because the command for the button is set to *Vdesk switchto deskname and because the option *"Show button with *Vdesk ... as pressed"* is selected on *Desktops >Setup* configuration. Note how desktop "Manual" is started with the CreateOrSwitchTo command; this command runs the command list Manual to create the desktop when it does not exist and switches to the desktop when it does exist.

Selecting *Desktop edit* shows a bar which only appears on that desktop in the top center of your screen. Note that when you select *Desktop explore*, extra buttons on the subbar "explore" are shown. This also requires the option *"Show subbar of same name as vdesk"* on the *Desktops >Setup* tab.

You can use hot keys ctrl-left arrow and ctrl-right arrow to switch desktops.

You can also switch desktops by left clicking the leftmost button (the one with the desktop name) and selecting a desktop from the menu.

Ctrl-right click on the bar, select *Configure*, and view the configuration of the bar, the label on the first item, and the menu Desks to see how the above features have been implemented.

The initial contents of the desktop are set by the entries on the *Desktop* tab. Note the Explore and Edit command lists used to populate these desktops. Note also the bar "baredit" for the Edit desktop.

You can use the *Vdesk arrange command to see the contents of all desktops. In the sample this can be activated by moving the mouse to the top right screen momentarily, or by right clicking the leftmost button on the bar. You can make this window smaller by using the resizing border at the bottom or by right clicking on the window and selecting the Use Small Window option.

Yet another way to access desktop features is to shift+right click a bar. You can use this menu to switch desktops or to create new desktops. You can also use the menu to lock windows on all desktops. Middle (or shift-left) click the left most button to start calculator. Shift-right click a bar and select Lock >Calculator. Calculator will become part of all desktops: try all of the desktop buttons to see this. You could have also pre-specify that Calculator be locked by entering the caption Calculator in the *"Windows to be shown on all desktops"* edit box on the *Desktops >Setup* configuration dialog.

You can repopulate desktops by running the desktop command list as a script. There is a sample on the Explore button. Left click this button to switch to desktop Explorer. Close the File Manager and Explorer windows. Now right-click the explore button and the explore command list is used to populate the desktop.

Built in Commands : *Wait

*Wait command

Use the Wait command in the More Commands box, or in a script, in order to wait for some condition before executing some of the commands. For example, you may want to wait for a specified amount of time, or until a given window is visible, or until a key is pressed.

PowerPro has many different ways of waiting. None handles all cases. You need to choose the wait most appropriate for your circumstances. Here is a summary of the kinds of waits:

- **wait for** can wait on many circumstances, as detailed below. Can be used anywhere and in any script situation. Multiple outstanding waits can exist, but only the last executed one is monitored; when it completes, the next to last is monitored, and so on.

- **wait until** can wait on many circumstances, as detailed below. Cannot be used in nested scripts (a script which is called from another script). Multiple outstanding waits can exist and each is monitored independently. Cannot be used with Do() function.
- **event plugin** can be used in any circumstances and many independent events can be outstanding. However, the event plugin is complex to use, especially when a wait in the middle of a script is needed.
- file.runwait and file.runcallback from the file plugin can be used in the special case where you want to run a program and wait for it to finish before doing other command

Most of the time, the wait for command should be considered. Use wait until or an event plugin call if you need multiple outstanding waits of greatly different duration or conditions which must act independently.

See the file event.txt in the plug in folder for more information on the event plugin. Following is information on the wait for and wait until commands.

(Programming note: As an alternative to using *wait in a script with a loop, consider using the Monitor command list as set on *Command Lists Setup* or the event plugin.)

*Wait for command

Syntax:

wait.for (n,expression)

wait.for (n)

wait.for (expression)

wait.forinterval (expression)

Here n must be an integer, not an expression.

In the first case, the wait continues for n milliseconds *or* until the expression becomes true. In the second, the wait continues for n milliseconds. In the third, the wait continues until the expression becomes true. In the last, the wait continues for the number of milliseconds given by expression. You must use wait.forinterval to specify an expression if you want to wait for a number of milliseconds, since wait.for(expression) waits until the expression is true.

Note the period between wait and for.

Examples:

wait.for (1500,activewindow("*notepad*"))

wait.for (50)

wait.for (ctrl or not anywindow("=explorer"))

Note:

The older literal syntax form: wait for n,expression

(no period between wait and for) is also acceptable.

*Wait until command

Syntax:

wait.until (n,expression)

wait.until (n)

wait.until (expression)

In the first case, the wait continues for n milliseconds *or* until the expression becomes true. In the second, the wait continues for n milliseconds. In the third, the wait continues until the expression becomes true. Note the period between wait and until.

Examples:

wait.until (1500,activewindow("*notepad*"))

wait.until (50)

wait.until (ctrl or not anywindow("=explorer"))

Note:

The older literal syntax form:

wait until n,expression

(no period between wait and until) is also acceptable.

This form of wait cannot be used with the Do() function.

other wait commands

waiting for m milliseconds

Command: wait

Parameter: sleep m

where m is any number, waits for that number of milliseconds. PowerPro will be unresponsive during the wait.

waiting with a message box and a count down timer

Command: wait

Parameter: message n,expression

or: message expression

displays a message box containing text from evaluation of the expression and, optionally, a countdown timer which starts at n seconds. Expression could be literal text in double quotation marks. If n reaches 0, or the "Start Now" button on the message box is pressed, then the wait ends and the next command is run; if the Cancel button is pressed, the wait ends and all following commands are ignored. The position of the message box is set by the *"Screen position for alarm message windows"* on the *Scheduler Setup* dialog.

You should also look into the event.message service, which also shows a countdown message box, but executes a provided command at termination instead of waiting.

waiting for mouse or keyboard activity

Command: wait

Parameter: activity

Waits until mouse or keyboard activity. Always waits at least 3 seconds to ignore activity associated with launching the command.

waiting for n seconds

Command: wait

Parameter: n

Waits for n seconds; m must be less than or equal to 30. Equivalent to wait.until(n000).

waiting for a program to finish

- The **runwait** service of the **file** plugin allows a program to be run and a script to be paused until the program completes while PowerPro remains responsive. Example:

`exitcode = file.runwait(maxwait, "c:/path/to/progr.exe", "params", "work","howstart")`

runs the specified program (which can be .exe, .bat, or .cmd) with specified parameter and workdirectory and waits up to maxwait milliseconds for it to end. "howstart" can be one of: min, max, hide, normal. The final three parameters are optional.

PowerPro remains responsive until the program has completed. If maxwait is zero, then there is no maximum wait. The result, exitcode, is set to the exit code of the program or to 9999999 if maxwait was exceeded.

- The **runcallback** service is similar but does **not** pause the script

`file.runcallback(maxwait, "PPro Cmd", "c:/path/to/progr.exe", "params", "work","howstart")`

runs the specified program (which can be .exe, .bat, .cmd) with specified parameter and workdirectory. "howstart" can be one of: min, max, hide, normal. The final three parameters are optional.

This call returns immediately, but when the program ends or maxwait milliseconds elapse, the PowerPro command "PPro cmd" is executed (this command could be a script call). In addition, the global variable _exit_ is set to the exit code or to 9999999 if maxwait is exceeded. If maxwait is zero, then there is no maximum wait.

Built in Commands : *Wait

Wait example

an example of using *wait

Here is a sample of a script which uses *Wait to wait on the status of both the modem and a program and uses *Window to terminate a program.

Starting the script uses the Dundial program, described in dundial.txt in the Powerpro directory, to dial a DUN connection. When the connection is completed, both Microsoft Internet Explorer and a communications optimization program called Speedup are started. The script then waits until the user terminates Internet Explorer; then the modem connection and the Speedup program are also terminated.

To configure this script, create a new command list called (say) internet. Then enter the items shown below. Once the menu is created and saved, the script can be run with the command:

Command:	*Script
Parameter:	run internet

Here are the entries for the command list "internet".

Name	Dial connection
Command	c:/program files/PowerPro/dundial.exe
Parameters	DunName UserName Password
Name	Wait for connection
Command	*wait
Parameters	until(modem)
Name	Start SpeedUp program
Command	c:/program files/speedup/speedup.exe
Parameters	
Name	Start Internet Explorer
Command	c:/program files/Internet Explorer/IExplore.exe
Parameters	

Name	Wait for explorer to be terminated
Command	*wait
Parameters	until(not anywindow(?"c:/program files/Internet Explorer/IExplore.exe"))

Name	Hangup connection
Command	c:/program files/PowerPro/dunhang.exe
Parameters	*

Name	End SpeedUp program
Command	*Window
Parameters	close =c:/program files/speedup/speedup.exe

In a file, the script would look like this:

```
"c:/program files/PowerPro/dundial.exe" DunName UserName Password
;; note that is suitable if dundial's 3 arguments are being supplied as literal text
;; otherwise one would wrap it in a do() statement.
wait.until(modem)
"c:/program files/speedup/speedup.exe"
"c:/program files/Internet Explorer/IExplore.exe"
wait.until(not anywindow(?"c:/program files/Internet Explorer/IExplore.exe"))
"c:/program files/PowerPro/dunhang.exe"
window.close (" =c:/program files/speedup/speedup.exe")
```

Built in Commands : *Wallpaper

*Wallpaper command

Use *Wallpaper to change the wallpaper. You can also change the wallpaper from the Media dialog.

In addition to .bmp files, Powerpro allows you to use .jpeg and .jpg files as wallpaper.

These are the *Wallpaper actions:

*Wallpaper Change	changes to file in same folder; random
*Wallpaper Change *	changes to file in same folder; sequential
*Wallpaper ChangeTo filespec	changes paper to filespec and saves new filespec in .pcf
*Wallpaper Show filespec	changes to filespec but does not store new file name
*Wallpaper Style style	style can be center, tile, or stretch

Related function:

str = paper Returns the current wallpaper file name

Built in Commands : *Window

*Window command

manipulating windows of running programs

purpose

Use the *Window command to ask Powerpro to close, minimize, tray minimize, rollup to caption and perform many other actions with the windows on your system. You can specify the windows to be controlled by selecting the active window, the window under the mouse, a window from a menu of active windows that Powerpro shows, a list of window captions, or all windows on your system.

configuration

The command has this format:

Command	*Window
Action	action
Parameter	windowID

- The action specifies what to do.
- The windowID species which windows to perform the action on.

examples

Command	*Window
Parameter	min active

minimizes the active window.

Command	*Window
Parameter	rollup menu

displays a menu of active windows; the selected one is rolled up to the caption.

Command	*Window
Parameter	show menu hidden

displays a menu of active windows including hidden windows; the selected one is shown and activated.

Command	*Window
Parameter	close all

closes all windows on your desktop.

Command	* Window
Parameter	Position 10 30 100 200 autorun

positions lasts window selected on autorun command list.

Command	*Window
Parameter	minmemory "*Netscape,*Internet Explorer"

swaps Netscape or Internet Explorer out to disk (NT only).

new syntax

As with most built in commands, in scripts the new Expression Syntax is recommended.

Instead of the literal syntax: Window Close all

Use the expression syntax: window.close("all")

Built in Commands : *Window

Actions for the *Window command

Here are the possible values for the **action** of the *Window command:

The action should be followed by a target window parameter

close	closes specified window(s)
close2	closes the window using a different technique than close
closeforce	forces the window to close; you may lose unsaved information
min	minimizes the window
max	maximizes the window
normal	displays as sizeable (not minimized or maximized)
move	move the window by moving the mouse; click any mouse button to stop
size	size the window by moving mouse; click any mouse button to stop
hide	makes window invisible
hideshow	makes window invisible if visible, shows it if invisible; take care when applying to multiple windows as there are many windows which should normally remain invisible
ontop	displays always on top (ontop is one word)
nottop	removes always on top setting (nottop is one word)
topnottop	reverses always on top setting (topnottop is one word)
show	activates the window and shows it if hidden
back	sends window to bottom of stack of displayed windows
backshow	sends window to back if it is foremost; activates if it is not
center	centers window within full screen
rollup	rolls up the window to just its caption; shows if it is already rolled-up
maxnormal	maximizes a normal window; makes a maximized window normal
minrestore	restores a minimized window; minimizes it otherwise
traymin	minimizes window to tray
automin	minimizes window to tray <u>if</u> window matches "Auto tray min" list on the Setup tab; otherwise does an ordinary minimize.
minmemory	sets the memory working set (NT only). [see notes below]
Position x y w h	sets a window's position and size [see notes below]
Trans n	makes window transparent (W2K, XP only). [notes below]
Transmouse n	make window transparent; put on top and make all mouse clicks pass through (W2K, XP only) [notes below]
SetPriority priority	sets process priority of selected window; process can be one of: idle (lowest), below, normal, above, high (highest)
PostMessage m w l	Does PostMessage(h, m, w, l) where h is selected window [notes below]

SendMessage m w l	Does SendMessage(h, m, w, l) where h is selected window; variable SendMessage is set to result of the SendMessage call

notes

For the Position command, you must type four numbers before the target window. The four numbers provide the window horizontal and vertical position (positive or negative) and the window width and height. You can capture these numbers from an active window using the find button.

Alternatively, you can replace the four numbers by center (to center), wmax to maximize width, or hmax to maximize height. You can use = for any of the four to keep the current value. You can precede the number by a plus sign to set relative to current position:

*Window Position +-50 = +-100 = active

moves 50 positions to the left and reduces the width of the active window by 100 pixels.

*Window Position = = +50 +50 under

increase width and height of window under mouse by 50 pixels.

See Auto positioning windows for automatically positioning a program's window every time it starts

For the Trans and TransMouse commands, precede the window target by an integer -254 to 255; the larger the number, the more transparent the window. Zero means not transparent. Negative numbers reverse the transparency each time the command is used.

For PostMessage and SendMessage you can use the string wm_command, wm_app, or wm_user to represent the corresponding message id. You can also use wm_user+n, where n is a number. You can enter a hexadecimal number by preceding it with 0x, e.g. 0x1f0a.

For example, to use SendMessage with WinAmp, use *Window SendMessage and

wm_command 40046 0 c=winamp v1.x

to pause winamp

wm_user 1 105 c=winamp v1.x

to set variable SendMessage to length of current track

See http://www.winamp.com/nsdn/winamp2x/dev/sdk/api.jhtml for details.

If you use the MinMemory command, you can optionally follow the WindowId with two decimal integers giving the minimum and maximum working set sizes in bytes. The virtual memory manager attempts to keep at least the minimum working set size resident in the process whenever the process is active and to keep no more than the maximum memory resident in the process whenever the process is active and memory is in short supply. If you omit these values, or if you specify -1 for both, the function temporarily trims the working set of the specified process to zero. This essentially swaps the process out of physical RAM memory.

Built in Commands : *Window

Specifying the windowID for the*Window command

Select one of the following options for the WindowID of the *Window command.

active	Selects the active window.
*	Selects the active window.
autorun	Last window matched by autorun command list. Use this for commands in autorun command lists.
activebar	Window corresponding to last active bar button pushed.
under	Selects the window under the mouse. For applications which use the Multiple Document Interface, the commands close, min, max, rollup will operate on the MDI child only; put Parent after under to avoid this and ensure the command always runs on the parent window.
menu	Displays a menu of active windows; select one for the action. Put hidden after menu to include hidden and tray minimized windows. Put traymin after menu to include tray-minimized windows. Put onlyhidden to include only hidden windows. If the *Window menu command is included in a Powerpro menu, the generated menu will be embedded in the outer menu. To ensure all items appear on screen you could put the *Window command as the sole entry in a submenu. Or to activate the *Window command when the menu item is clicked on, put noembed in lower case at the end of the *Window command You can also put a captionlist after the keyword menu to include only those windows matching that list.
menux	Same as menu, except that if only one window matches the specified captionlist, that then command is executed on that window without showing the menu.
all	Selects all visible windows, including minimized windows.
captionlist	Selects the windows specified in the list. Enter one or more window captions, separated by commas. Enter xxx* for captions starting with xxx, *yyy for captions ending in yyy, and *zzz* for captions containing zzz anywhere. Or you can enter =exename to select all windows shown by the program with file name exename (you must only enter the file name: not the path and not the .exe extension). Example: "*Notepad,*Internet Explorer, =calc" selects notepad windows, Internet Explorer windows, and Calculator windows. Or use any standard caption list. To avoid an error message if no window matches the caption, put an exclamation mark immediately after the action keyword, e.g. *window close! *notepad* You can also use window handles in the window_list.

Built in Commands : *Window

Auto positioning windows

position a program's window each time it starts

To position the window of any program every time the program starts, use a combination of the *Window Position command and the autorun command list. The idea is to create an autorun command list with entries for each window you want to position; and execute the *Window Position command to position the window when it firsts opens.

example

Suppose we wanted to position Notepad at position 10 20 with size 300 400 and to position Exploring of c\setup at position 100 200 with size 300 300.

Create a command list called AutoRun or edit any existing autorun command list you have. Add these two entries to the command list

Name	*- Notepad
Command	*Window
Action	Position
Parameters	10 20 300 400 autorun

Name	Exploring - C:\setup
Command	*Window
Action	Position
Parameters	100 200 300 300 autorun

The Name fields are used to indicate the window to be positioned by identifying a caption that will match the desired window(s). Note how an asterisk is used with Notepad to match any notepad main window.

The parameters give the position and size of the window. The keyword "autorun" indicates the *Window Position command should operate on the last window matched by the autorun command list.

When configuring the position command, you can capture the window position and size of the active window by pressing the find button beside the parameters edit box.

Scripts

PowerPro Scripts

There is a separate tutorial on scripting in the file PPST.chm in the PowerPro folder. There is a summary of scripting for people familiar with programming in the next section.

purpose

PowerPro has a scripting language to run a series of commands. A script can be stored in a text file or in a command list. Scripts have many features including:

expressions, such as variables, calculations, dialog boxes for input, string functions, functions to access system settings like mouse position, etc

programming with variables, assignment statements, for... loops, if...else for flow control,

plugins so that other programmers can create services which can be used as functions or commands in PowerPro, such as file manipulation, registry access, regular expression usage

command list functions which allow command lists to be manipulated programmatically

storing scripts in files

A script file is a plain text file, editable in Notepad or your favorite text editor. If the extension is .powerpro, you can double click on the file to run it. Or you can use .txt as the extension.

Each line in a script file is one complete "statement". It is equivalent to the Left command of one Item in a script saved as a command list, including its parameter fields.

For example if a command list script contains this item:

Command:	*Bar
Action:	Show
Parameter:	mybar1

then the equivalent script file would have this line:

```
Bar Show mybar1
```

or, using the newer syntax for Powerpro commands (recommended):

```
Bar.Show("mybar1")
```

(note the dot between Bar and Show)

Each line in the script file can run an exe file or a built-in PowerPro command. To run an exe file with expressions as its parameters, use the do() command. To run built-in commands like Menu or Clip, use the Command.Service(params) syntax.

There are also several Script commands which can only be used in scripts, such as if(condition) and jump label.

Statements in file scripts can be labeled by preceding them with @, for example

```
do ("c:\windows\notepad.exe")
wait for (visiblewindow(" =notepad"))
@loop win.sendkeys("a{enter}")
if (not ctrl)
jump loop
win.sendkeys("b")
```

This file script starts notepad, waits till its window is visible, and then sends the letter a until the ctrl key is pressed. The letter b is sent and the script file exists.

special characters

You can continue one statement on multiple lines, up to 530 characters, by ending each line except the last with `\+` (that is, backslash, then plus), assuming backslash is your escape character as set on *Setup > Advanced > Characters*. Regardless of the escape character used, you can also continue with `;;+`

Any line starting with a semi-colon is ignored and can be used as a comment. Blanks and tabs at the start of a line are ignored and can be used for indentation to show structure. You can include comments at the end of a line by preceding with two semi-colons or two slashes:

```
alpha = beta -1 ;; this is a comment
alpha = beta -1 // this is a comment
// this is a comment
```

You can create multi-line (block) comments by starting the first comment line with /* and ending the last command line with */. Single line block comments are allowed:

```
/* single line comment*/
/* multi-line
 line
 comment
*/
```

running script files

There are five different ways to run a script file: use a string to specify the script path, use a variable, using the runfile.filename format, using the .filename format, or using the call function. They are shown here:

```
"filepath" (arg1,arg2, …, arg23) //you can use a string containing a file name or full path
var = "c:/path/filepath"
var(arg1,arg2, …, arg23)     //you can use a variable containing a file name or full path
runfile.filename(arg1,arg2, …, arg23)
.filename(arg1, arg2, …, arg23)
call ("filename", arg1, arg2, …, arg23) //the first argument can be partial or full path
```

The filename can only contain alphanumerics and the _ character, and the \ to indicate a subfolder. If the file name has no extension, then .txt or .powerpro is used; PowerPro looks for both.

For the runfile, .filename, and call methods, you can omit the parentheses and list of arguments if there are no arguments. However for the first two approaches, you must use () if there are not arguments. The following show how to do calls with no arguments:

```
"filepath" ()
var = "c:/path/filepath"
var()
runfile.filename
.filename
call ("filename")  //the first argument can be a partial or full path
```

For the var() format, the "..."() format, and the call function, the filename can be a full or partial path, or just a filename with no folder specified. For the others, only a filename or partial path can be used.

If a full path is not specified, PowerPro will search different folders to try to find the file, using the following steps:

If a script file is currently executing, try the folder containing that script file.

Try all of the folders listed in the *Script Path statement.

Try the \scripts\ subfolder of the folder containing the PowerPro configuration .pcf file.

The *Script Path statement provides a list of one or more folders separated by a semi-colon or comma. Each folder can be a full path or a relative path; a relative path is assumed to be under the scripts subfolder. For example

```
*script path c:\myscripts;common
```

would cause PowerPro to search c:\myscripts and c:\program files\powerpro\scripts\common (assuming configuration file was in standard folder).

functions in script files

Script files can contain functions which are like mini-scripts which are executed independently. You can start execution of a function called FName by using any of the above forms with @FName after the filename; that is,

```
var = "c:/path/myscript@FName"
var(arg1,arg2, …, arg23)     // variable contains a file name or full path and @FName
"myscript@FName" (arg1,arg2, …, arg23) //string contains file name or full path and @FName
runfile.myscript@FName(arg1,arg2, …, arg23)
.myscript@FName(arg1, arg2, …, arg23)
call ("myscript@FName", arg1, arg2, …, arg23)   //the first argument can be a partial or full path
```

which would start file myscript.txt at the function FName.

Use the function statement to indicate the start of a new function.

```
Function FName(arg1, arg2, …, arg23)
```

will assign the arguments to local variables arg1 through arg23 and then start execution of the function at the next statement. Execution continues until another Function statement, or a quit statement, or the end of the file.

Note that you can put many functions in a file. You can end each with a quit statement, if you like, but this is optional as PowerPro automatically quits a function in a script when it finds another a Function statement or the end of the file.

There is a simplified approach for calling a function which appears in the same file as the call. You can call a function FName which appears within the same script file with

```
FName(arg1,arg2, …, arg23)     //Call a function within the same script file
FName()     //Call a function within the same script file with no arguments
```

You can accomplish the same thing with

```
.@FName(arg1,arg2, …, arg23)     //Call a function within the same script file
.@FName    //Call a function within the same script file with no arguments
```

If a script file starts with a Function statement, and the script file is called without the @FName, the call executes that first function in the script file.

Note: versions of PowerPro before 4.6.01 did not have a function statement and used the following instead:

```
@FName
Args arg1, arg2, …, arg23
```

arguments

The Function statement is used to assign the arguments from the call to local variables in the script. For example, the following script returns the maximum of three arguments:

```
Function GetMax(a1,a2,a3)
local max=a1
if (a2>max)
max=a2
if (a3>max)
max=a3
quit (max)
```

You can also access the arguments in the script with the function arg(n) where 1<=n<=23. You can use arg(0) to access the number of arguments used to call the script. Here is the same script using the arg function.

```
local max=arg(1)
if (arg(2)>max)
max=arg(2)
if (arg(3)>max)
max=arg(3)
quit (max)
```

Or you can use the args command to assign arguments to up to 23 variables. List up to 23 variable names separated by spaces or commas after the keyword args. The variables are declared as local and assigned the corresponding arg(i) value. Note the first variable is always assigned arg(1). Thus, the statement

```
args alpha, beta, gamma
```

is exactly equivalent to

```
local alpha = arg(1)
local beta = arg(2)
local gamma= arg(3)
```

importing functions from other script files

You can use the imports statement to allow you to use func(xxx) to call functions in other files. After PowerPro processes the statement

```
imports (filepath_expression)  func1, func2, func3, …
```

it assumes functions calls to func1, func2, etc are calls to the file path given by the filepath_expression. This expression can yield an absolute or relative path (in which case, the Script Path applies). For example

```
imports (pprofolder ++"greek/to me/stuff")  alpha, beta, gamma
```

declares that functions alpha, beta, and gamma reside in file greek/to me/stuff under the PowerPro folder.

The imports statements creates local variables with the same name as the imported functions; do not try to use the variables for other purposes.

storing scripts in command lists

You can also store scripts in command lists. Place each line of the script as a separate item in the command list. Only the left command can be used. Avoid using the More Commands field.

Place labels in the item name field. Unlike in file scripts, do not use the @ character at the start of labels.

Programming statements like if, else, endif, for, endfor, can be entered directly into the command edit box, e.g.

```
if (test>value) do
```

Or you can set the *command* to *Script, the *action* to (say) if, and the *parameters* to the remainder of the statement.

running scripts stored in command lists

Use the syntax

```
run.clist(arg1, arg2, …, arg23)
```

to run a command list called clist with the specified arguments. As with scripts in files, use the arg(n) function to access the nth argument. Omit the parentheses and list of arguments if there are no arguments.

Also as with scripts in files, you can use the syntax

```
run.clist@startpoint(args)
```

to start the command list at label startpoint.

predefined command list script names

The following command list names are called automatically as scripts under the documented conditions:

Monitor	Called once per second; if *>Command Lists >Setup >Special Lists >"Run monitor..."* is checked.
Reconfigure	Called each time PowerPro is reconfigured while running; if *Setup >Advanced >Other >"Run Reconfigure with new configuration"* is checked.
PProShutdown	Called when PowerPro shuts down, whether by manual shutdown or by system shutdown; if a list with this name exists.
HookWindowEvents	See Running a script when a window or system event occurs for details.

Scripts

Scripting for Programmers

Objective

This section provides an overview of PowerPro scripting for people who are already familiar with programming in a language like the C family, Java, VB. It first discusses basic structure, then expressions, variables, and vectors, then control constructs, calling PowerPro commands, and finally plugins and handles.

Where there is a choice of what syntax to use to accomplish something in scripting, this section uses the most recent, preferred approach.

Structure of PowerPro Scripts

Scripts are stored in files with extension .txt or .PowerPro. Each line starts a new statement. There is no special character (like semi-colon) at the end of statements. Continue lines by putting ;;+ at the end of the line; statements can be up to 530 characters long. Blanks and tabs at start of lines and blanks within lines are ignored.

Comments start with // or ;; anywhere on the line. You can also create multi-line comments starting with /* and ending with */

Script files can be divided into functions which can have zero to 23 arguments; each function starts with

```
function Func(arg1, arg2, …, arg23)
```

The functions ends with a quit, end of file, or the appearance of another function statement.

Case of letters in function names is ignored. Call functions within the same file with

```
Func(arg1, arg2, …, arg23)
```

Call functions from other files with

```
.filename@Func(arg1, arg2, …, arg23)
```

where filename is in scripts subfolder or another folder set by Script.Path statement. Note dot at start. A script file can also be called directly and started at the first line (which may or may not be a function statement) with:

```
.filename(arg1, arg2, …, arg23)
```

Functions in other files can be imported and then referenced with Func(args) with

```
imports (filepath_expression) func1 func2 func3
```

In all cases, use Func() if there are no arguments.

Functions can return values with

```
quit(expression)
```

PowerPro scripts are interpreted line-by-line as they are read from the file. There is no compilation.

There is a script debugger.

Expressions

PowerPro supports the usual expression syntax for strings, integers, and floating point numbers. Integers are stored internally as strings. Floating point numbers are not and must be displayed with floattostring(x) function. The binary operator ++ is used for string concatenation; all other operators are standard C/Java, except that unary ++ and –- can only be used for autoincrement/decrement with plain variables.

There are many built-in functions to manipulate strings, do input/output, access PowerPro and system state, and so on.

Literal strings can be defined within double quotes in which case the backslash is an escape character and must itself be escaped as in

```
var = "backslash \\ then tab \t then newline \n"
```

Literal strings can also be enclosed by ?x…x where x is any non-alphanumeric character; in this case, there are no escape sequences.

```
var = ?%c:\file path\with\backslashes\name.txt%
```

PowerPro has built-in support for regular expressions. The format

```
var[expression]
```

takes expression as a pattern to be matched against var and the result is returned. The form

```
var[expression] = expression2
```

matches pattern against var, replacing matched substrings by expression2.

Variables and Vectors

Variables names can be up to 64 characters, and include any alphanumeric or _. Case is ignored.

Variables can store zero-byte-terminated strings of any length, floating point numbers, integers (stored as strings), and handles (see section on Plugins below). You do not declare the type of a variable; assign any value to any variable.

You do declare a variable's scope, which can be global, local (released at script quit) or static (local, but maintains value when script quits) with statements like:

```
global g_x,, glob2
local myvar = expression   // only one initializer per statement.
```

Declarations are executed just like any other statements. Declarations can appear anywhere and do not take effect until they are executed.

Global variables can be used in any script or outside of a script (e.g. in button commands). Each function in a file has its own, separate local variables. All functions in a file share static variables declared in that file.

Vectors are declared with statements like

```
static vec1[55], vec2[length(string1)]
```

Vector elements are referenced and assigned with vec1[index_expression]. PowerPro supports one and two dimensions for vectors. Vector names are in fact variables containing handles; these vector names can be used to access services in the vec plugin.

PowerPro also supports maps (hash tables) which are created with map.create and then have elements referenced and assigned with mapname[key_expression].

Control Statements

Powerpro supports if, if-do-endif, if-do-else-endif, if-do-elseif-else-endif, for-endfor, break, continue, quit. Note that keywords, not braces, are used to indicate the scope of statements controlled by if and for. Nesting is allowed. Here are examples showing the structure:

```
if (expression)
  single statement
if (expression) do
  one or more statements
endif
if (expression) do
  one or more statements
else
  one or more statements
endif
for (local j=0; j<aVec.length; j++)
  one or more statements
```

```
break  // ends for loop
continue // go to increment statement in for loop
endfor
quit (expression)  // quits function and returns expression; expression optional
```

Executing Files and PowerPro Built-in Commands

To ask Windows to execute a file with blank-separated command-line parameters, use:

```
Do(filepath_expression, parameters_expression) //all parameters in one string expression
```

You can omit the parameters_expression to run a file directly with no command line parameters.

To run PowerPro built-in commands, use the following syntax. PowerPro commands usually have a command name (examples are Window and Vdesk) and an action (like close or switchto). The way to run a built-in commands is shown by the following general syntax, followed by examples for Window and Vdesk:

```
command.action (param1, param2)  // one parameter for each blank-separated command line arg
window.close("*notepad*")  // close window with notepad in caption
vdesk.switch("mydesk")   //switch to named desktop
win.debug(expr1, expr2) // up to six expressions to display in debug window
win.keys("keys to send")
```

The last two example show specials cases for debug output and for sending keys.

Plugins and Handles

Plugins provide a way to extend PowerPro capabilities. There are plugins for manipulating windows, notes, the clipboard, Unicode, dates, ini files, dialogs, the registry, regular expressions, COM objects and even for calling Windows API and other dlls. A plugin called (say) myplug exposes services which are called as part of expressions or as standalone function calls like this:

```
myplug.service(arg1_expr, arg2_expr,…, arg23_expr)
```

Use () if no arguments.

Many plugins return handles which can then be used in expressions to call other services in that plugin and apply the service to the value represented by the handle. For example, the following moves a window with handle in hwnd:

```
local hwnd = win.handle("=exename")  // get handle to window from program exename
if (hwnd)    //move it 5 pixels to the right and 10 down
  hwnd.move(hwnd.left+5, hwnd.top+10)  // uses win move, left, top services on handle in hwnd
```

You can also use the dot operator to apply functions to strings and floats, such as in:

```
 str.replacechars("alpha", "aleph")  //returns str with alpha replaced by aleph
aFloat.format(5)    //returns formatted aFloat with 5 digits after the decimal
```

Command Lists and Scripts

You can manipulate command lists and show bars and menus from them, by using the cl functions.

Scripts

Script commands

Some of the Script commands can be used only inside scripts. Others can be used anywhere that accepts a PowerPro command.

Note: It is not necessary to type the word *Script in front of any of these commands.

Script commands used only inside scripts

These may be used in a File Script, or in a Command List script (or in the More Commands box if *Exec MoreCommandsAsScript was used as the item's primary command).

Quit	Ends execution of the script.
Quit all	Ends any calling scripts too.
Quit(exp)	Sets the Return Value from a script which was called using: var=run.script(args) or var=runfile.scriptfile(args) or var=.scriptfile(args)

If(expr)	Tests an expression to control whether the next single statement is executed
If(expr) do	Tests an expression to control whether the next block of statements (up to an Else or ElseIf or Endif) are executed.
Else	Used with *Script If()do
ElseIf	Used with *Script If()do
EndIf	Used with *Script If()do

For (var=initexpr; testexpr; var=increxpr)	Starts a for loop
EndFor	Ends a for loop
Break	Breaks out of a for loop
Continue	Skips rest of for loop until endfor, then continues loop

Jump label	Jumps to a label in a script. *In script files:* Jump label Jumps to a labelled line which starts with @label The rest of the line can be blank or it can include a command. *In command lists run as a script:* *Script Jump itemName Jumps to an Item with Name itemName Unlike in script files, the Name of the Item to be jumped to should not start with a @

Function	fn(args) – Declares start of function named fn. Only for scripts in files.
args v1 vn	Used inside a script. Declares nth variable local and assign it nth argument arg(n).

*Script commands used inside or outside of scripts

Debug text	Writes text to the debug window.

Alternative: win.debug(sz1, sz2, ... sz6)
up to 6 arguments can be specified; they are joined and shown in the debug window.

Close commandlist	Ends all programs listed in commandlist
CloseForce commandlist	Forces them closed, possibly losing data.

flags

Flags can only have a value of 0 or 1

Flag set n1 n2	Sets flags n1 to n2 inclusive Example: Flag set 4 7 sets flags 4, 5, 6, 7
Flag clear n1 n2	Clears flags n1 to n2 inclusive
Flag reverse n1 n2	Toggles the values

There are 32 flags numbered 0 to 31. So Flag Clear 0 31 clears all flags.

Test a flag with pproflag(n) , for example:

If (pproflag(3)) do or If (not pproflag(27)) do

PowerPro's "script path"

With any of the script running functions, if you do not specify a full path to the script file, PowerPro will search different folders to try to find it, using these steps:

- If a script file is currently executing, try the folder containing that script file.
- Try all of the folders listed in the *Script Path statement. [see below]
- Try the scripts subfolder of the folder containing the current configuration .pcf file.

The Script Path provides a list of one or more folders separated by a semi-colon or comma. Each folder can be a full path to any folder or a relative path; a relative path is assumed to be under the scripts subfolder.

*Script Path path1;path2;... [one or more paths] Sets the search path for scripts.

myvar = scriptpath Returns the current setting.

Example: *script path c:\myscripts;morescripts
would cause PowerPro to search c:\myscripts and c:\program files\powerpro\scripts\morescripts [assuming that the configuration file was in the standard folder].

Scripts

Running a script when a window or system event occurs

hooking events – for advanced users

window events

If you create a special command list called **HookWindowEvents**, PowerPro will automatically run the command list whenever one of the following events occurs:

0 = A moving or sizing operation is completed using the GUI

1 = A window is maximized or restored to normal using the GUI [check with win.maxxed()]

2 = A window is minimized using the GUI.

3 = A new window is created.

4 = A window is destroyed.

5 = A window is activated (i.e. made foreground, also called focussed)

The command list is called with four arguments:

arg(1)	The integer event number from the above list (e.g. **3** for new window)
arg(2)	The window handle
arg(3)	The window class
arg(4)	The window caption (arbitrary when window first created-- event **3**).

To use a file script called (say) **filehook.powerpro** instead of a command list, set the first line of the command list to `.FileHook(arg(1), arg(2), arg(3), arg(4))`

system events

Similarly, if you create a command list called **HookSystemEvents**, then it will be called with a single argument indicating one of the following system events whenever such an event occurs:

arg(1)==1	Power suspended.
arg(1)==2	Power resumed after suspension.
arg(1)==10	Display resolution changed

Scripts

Programming scripts with if, jump, variables, flags

You can program scripts with `for` loops, `if` statements, and variables. These scripts can reside in command lists or files.

You may find it helpful to print this help section out for study.

variables

PowerPro lets you create variables to hold text or numbers and use them in expressions. Variable names must start with a letter. They can consist of up to 63 letters and digits and underscores; The case of letters is ignored. You cannot use any of the following as variable names:

mci, not, random, vdeskempty, mounted, length, anywindow, visiblewindow, activewindow, validpath, input, inputcancel, timer, filemenu, arg, timerrunning, run, eval, pproflag.

Variables store text strings, but PowerPro will automatically interpret the strings as numbers when appropriate (.e.g. `"-5" + "3"` is "-2").

You assign a value to a variable using `var = expression`

For example

```
var1 = "abc"
w2 = length(var1) + 5
LCheck = var1 le "def" and w2 >3
```

assigns the variable var1 the string "abc", the variable w2 the string "8", and the variable LCheck the string "1" (representing true).

If you do not assign a value to a variable before you use it in an expression, the variable is initialized to the empty string ("") which is treated as zero in arithmetic or logical expressions. Or, you can use a setting on the *Setup >Advanced >Other* dialog to have an error message when you use undeclared variables to help you catch spelling mistakes.

You can use variables outside of expressions in commands and script file executions.

You can prompt for a variable using the input or inputdefault functions.

```
v = input ("Enter the value")
```

will prompt for input to set the variable v using a dialog box titled "Enter the value". You can also use inputdialog to prompt for several variables in the same dialog.

You can use the operators += , -= , *=, /=, %=, <<=, >>= ++= to add, subtract, multiply, divide, mod, shift, or join a value to a variable (or vector element). For example

```
w2 += length(str)*2
```

is exactly equivalent to

```
w2 = w2 + (length(str)*2)
```

Note that parentheses are added around the expression.

You can use the operator .= to apply a function directly to a variable. For example

```
w2 .= replacechars("abc", "***")
```

is exactly equivalent to

```
w2 = w2.replacechars("abc", "***")
```

Note that no extra parenthesis are inserted for .=.

You can also set a variable to a string with

```
setstring var anycharacters
```

which sets the variable var to the following characters, except the first space character after the var. In setstring there is no special processing for escape characters.

global, local and static variables

This section is for advanced script programmers.

By default, when you create a variable in PowerPro, it is a global variable. This means that the variable name refers to the same memory location no matter where the variable name appears. So if you use the variable name myVar in two different scripts, the same memory locations will be accessed. You may not want this. Instead, you may want to be able to use a variable in a script without worrying about whether the same variable name is used in some other script. PowerPro has two types of variables which allow you do this: static variables and local variables.

Both static variables and local variables are known only in the script file/command list that creates them. That is, you can create a local or script variable called (say) locVar in script alpha and a local/static variable call locVar in script beta. The two variables will exist independently and will not refer to the same memory locations. The differences between static and local variables are discussed in the following:

static variables

You create static variables with the *script static statement, which takes the form:

```
static var1 , another third
```
[use either a comma or a space between each variable]

Variables which are to be static are listed after the keyword static and separated by blanks or commas. The static statement must be executed before using the variables it lists. Static variables retain their value after the script exits. So if this script is named myScript,

```
static test
test="mystring"
quit
function sub
static test
win.debug(test)
```

then a call .myScript followed by a call .myScript@sub will output "mystring" to the debug window.

You can declare a variable to be static and assign it any expression with a statement like:

```
static var = 3*other
```

local variables

You create local variables with the *script local statement, which takes the form:

```
local var1 another,third
```

Variables which are to be local are listed after the keyword local and separated by blanks or commas. The local statement must be executed before using the variables it lists. Local variables do not retain their value once a script quits. Instead, the memory used by the variable is freed when the script exits; if the variable holds a vector or map, PowerPro will free all of the vector/map memory as well. Each time the script is called, a new version of the local variable is created. For example, if the following is in a script called myScript:

```
local test
test = "outer"
sub()
win.debug(test)
quit
function sub
local test
test="inner"
win.debug(test)
quit
```

Then the call .myScript will output "inner" then "outer" to the debug window.

You can declare a variable to be local and assign it any expression with a statement like:

```
local var = 3*other
```

If a script calls itself, eg with

```
.@callme(arg1)
```

then any local variables declared during the exection of callme will be in different memory locations than local variables declared before the call to callme, even if the names are the same. However static variables with the same name as those declared before accessing callme will use the same memory locations.

global variables

Finally, you can use the *Script Global statement to declare a global variable. Its format is similar to the local and static statements, e.g.

```
Global var, g2 g3
```

The global statement is only needed if you check *"Variables must be declared"* on *Setup >Advanced >Other*; the idea is to catch misspelled variable names by forcing all variable names to be declared in Global, Local, or Static statements.

You can declare a variable to be global and assign it any expression with a statement like:

```
global var = 3*other
```

conditional statements
if / if()do ... else ... elseif ... endif

Use conditional statements if and if()do to control which part of a script gets executed.

For *If (expression) [without do] the next statement following the if is skipped if the expression evaluates to 0 or the empty string ""

To control more than one statement, use

```
If (expression) do
... statements ...
elseif (expression)
... statements ...
elseif (expression)
... statements ...
else
... statements ...
endif
```

Note keyword do which signals that the if statement controls a block of statements.

The expression must be in parentheses. The one or more statements following the If()do are skipped if the value of the (expression) is 0 or "" until an elseif with an expression which is not 0 or "" or an else are found and the statements following the elseif or else are executed instead. Otherwise, the statements following the if()do are executed then the statements between the else or elseif and the endif are skipped. You can use any number of elseif statements, including none. You can omit the else and its statements. If both else and elseif appear, the else follows all elseifs. You can nest other if()do-else-endifs within the statements follow the if()do or the else. For example

```
if (ctrl or v>3) do
    *Message ctrl is down or v is greater than 3
    x=12
else
    Message ctrl is not down or v is le 3
endif
```

Shows the first message when the expression is true (evaluates to non-zero) and assigns x the value 12. Otherwise the second message is shown.

```
if(a eq 1) do
    win.debug(" a is 1")
elseif (a eq 2)
    win.debug(" a is 2")
elseif (a eq 3)
```

```
    win.debug(" a is 3")
else
    win.debug(" a gt 3")
endif
```

Selects one of the win.debug statements depending on the value of a

using if() in the more commands box

The above explanation applies to if() used in script files, or in command lists without any More Commands. You can also used a restricted form of if-else with More Commands in command entry controls.

If the if() is followed by commands in More Commands, then those commands are executed if the expression is not 0 or "". If the expression does evaluate to 0 or "", then the contents of More Commands are checked for an else statement. If there is an else statement, then all statements up to the else are skipped if the expression is 0 or ""; if there is no else, then all statements in More Commands are skipped.

Note that the keyword Do is optional: an if with More Commands always applies to all following More Commands. You cannot use endif or elseif with More Commands.

for ... endfor and break/continue statements

Use the For statement to create a loop. The statements

```
for (var=initexpr; testexpr; var=increxpr)
... statements inside the loop ...
endfor
```

are exactly equivalent to

```
var=initexpr
@loop
if (not testexpr)
    jump exitloop
... statements inside the loop ...
var = increxpr
jump loop
@exitloop
```

In other words, the for executes the initialization var=initexpr, then tests the textexpr. If not 0 or "", the statements and the var=increxpr are executed, then the loop is repeated until the testexpr is 0 or "".

You can prematurely end a for loop with the break statement: executing a break skips past the endfor. You can use continue to skip the rest of the iteration of a loop and continue with the next iteration; this statement effectively jumps to the endfor of the loop.

You can nest For statements to a maximum depth of 3.

The statements var=initexpr and var=increxpr can actually be any statement (except for or if).

You can omit sections of the for statement:

```
for (; testexpr; var=increxpr)
for (var=initexpr; testexpr;)
for (var=initexpr; ;var=increxpr)
for (;textexpr;)
for (textexpr)
```

If you omit the test expression, then it is taken to be true and you will get a loop that can only be terminated by a break or quit. Make sure you use ; ; with a blank in this since ;; with no blank means start of comment.

The form for(testexpr) is equivalent to a "while" statement as used in other languages.

Do not use jump to exit a for loop. You will get error messages or unexpected results if you do. Use break.

Be careful when nesting for and if()do statements. If an if()do is within a for loop, the corresponding endif must be as well. Also, if a for is within an if()do, the corresponding endfor must be as well. PowerPro does not always check for this, but you will get unpredictable results if you do not nest correctly.

You cannot use for() in More Commands; it only works in scripts.

You cannot use a wait.until statement in a for loop.

quitting scripts

Powerpro normally executes all commands until the end of the script, but you can stop execution by the quit command. Follow quit by the keyword all to quit any calling scripts too.

If your script is called with the syntax run.script(args) or runfile.script(args), you can return a result from the script with the syntax

```
quit (expression)
```

For example

```
max=arg(1)
if (arg(2)>max)
    max=arg(2)
if (arg(3)>max)
    max=arg(3)
quit (max)
```

returns the maximum of three arguments.

jump

To jump or loop, use

```
Jump xxx
```

to go to label xxx of the currently executing script for the next command.

For command lists, the label is put in the menu item's *Name* field as xxx.
For files run as scripts, the label is put as @xxx at the start of a line. The rest of the line can be blank or it can include a command.

flags

To help with Script programming, Powerpro has a set of 32 flags which you can manipulate and test. To set flags n1 through n2:

Command	*Script
Action	flag
Parameters	set n1 n2

where n1 and n2 are any numbers between 0 and 31. You can omit n2 if you only want to access one flag. To clear flags, use flag clear n1 n2; to toggle (reverse) the setting, use flag reverse n1 n2. Use 0 31 for n1 n2 to access all flags.

You can test the flag with the if command;

Use: `if (pproflag(4))`

to check to see if flag number 4 is set;

Use: `if (not pproflag(4))`

to check to see if it is clear.

You can reference flags in any expression as pproflag(n), where n is a number between 0 and 31; if the flag is set, then the value is 1, else it is 0.

You can set flags at start up with the command line.

Caution: flag 0 is set or cleared by several commands and functions to indicate a result. Do not use flag 0 for your own data.

Scripts

Vectors

Overview

Powerpro supports single dimension vectors through the use of braces [] and the vec plugin.

A vector is a set of strings which are accessed and assigned by an index number. The index numbers must be between 0 and the vector capacity. For example, if v is a vector, v[0] is the first string, v[2] is the third string, and v[5+2*3] is the 12th string. Note that you can use any expression that yields a non-negative number to generate the index of a vector.

You assign elements to a vector with statement like

```
v[index] = expr
```

All vectors have a maximum number of elements, called the capacity. You can specify that PowerPro automatically grow the vector capcity when you assign an element one beyond its current capacity.

Creating Vectors

```
v= vec.create(capacity, growth, minsize)
```

Creates a vector with specified capacity.

You can also use

```
local v[capacity; growth; minsize]
```

(You could use static or global instead of local. Note use of semi-colons to separate growth and minsize. A comma would signal a 2D vector).

If growth is >0, the vector will be grown automatically if you set or insert element with index equal to capacity. (Attempts to set or get an index greater than capacity are in error). If non-zero, growth is usually about the same order of magnitude as capacity, rather than very small, to avoid allocating memory too often.

If you specify minsize, all elements will be allocated at least this much memory to their string values, and this memory will be re-used if any new string is written to the element which requires no more memory.

Both growth and minsize can be omitted, in which case they are assumed to be zero Zero for growth means the capacity is fixed. Zero for minsize means vector elements are allocated new memory when a string bigger than the element's current string is assigned.

PowerPro permits the elements of a vector to be themselves vectors (or any other object, such as a map); for example

v[0]= vec.create(10)

allows you to reference the jth element of the vector v[0] with v[0][j]. Different vectors of different lengths could be created and assigned to v, if desired. Note that this type of vector of vectors is different from a 2D array, described next.

v2= vec.create2D(rows, columns, minsize)

or

local v2[rows, columns; minsize] //could be global or static too

Creates a 2-Dimensional array with the specified number of rows and columns. Elements if the array are stored contiguously by row. You reference elements with v2[i,j] to access ith row, jth column. You can also access the elements with a single subscript. Because arrays are stored by row, v2[i,j] can also be accessed by v2[i*columns+j].

You cannot grow or insert elements into 2D arrays.

If you specify minsize, all elements will be allocated at least this much memory to their string values, and this memory will be re-used if any new string is written to the element which requires no more memory.

v= vec.createFromLines(string, growth, minsize)

Creates a vector with one element for each line in the string. Lines in string are separate by \n, \r, or the pair \r\n. The \r or \n at the end of each line is not stored. For example,

v=vec.createFromLines("line1\n2222\r\n3\n\n")

creates a four-element vector with v[0] set line "line1", v[1] set to "2222", v[2] set to "3", and v[4] set to "".   As another example,

v=vec.createFromLines(clip.get)

creates a vector with one element for each line on the clipboard. Growth and minsize parameters are both optional and serve the same function as in the Create service.

v= vec.createFromWords(string, dchars, growth, minsize)

Creates a vector with one element for each word in the string. Words are separate by one or more consecutive occurences of any character from dchars (blank and tab if dchars omitted). The delimiting characters at the end of each word are not stored. For example,

v=vec.createFromWords("line1 2222\t3")

creates a three-element vector with v[0] set line "line1", v[1] set to "2222", and v[2] set to "3",.   As another example,

v=vec.createFromWords(win.handleList("*"))

creates a vector with one element per handle of visible windows. Growth and minsize parameters are both optional and serve the same function as in the Create service.

Other Vector Operations

v=v.destroy

Frees all memory assigned to v and its elements and frees the handle for use for another vector. Returns 0 which you can then assign to v to indicate that v is no longer a valid vector.

v.exists

Returns 1 if v is a valid vector; 0 otherwise.

v[i]=expr

Assigns the expression expr to element i of vec v. Any expression can be used for the index i. Indexes start at zero. If i>=vec.length, but less than v.capacity, then the vector length is increased to i+1. If i==v.capacity, and growth is bigger than 0, then PowerPro will grow the vector by growth elements to accomodate the new element. Indexes i<0 or i>vec.capacity are in error and will elicit an error message from vec. You can also use v.set(i,expr).

You can also use +=, -=, and so on, as in the following to squeeze runs of blanks in v[i]:

v[i].squeeze(" ")

To retrieve values:

s=v[i]

Retrieves element i of vec v and assigns it to s. Indexes start at zero. Access to unassigned elements < vec.capacity return "". Indexes i<0 or i>=vec.capacity are in error and will elicit an error message from vec. You can also use v.get(i).

v=vec.localcopy(arg(1))

This service must be used if you want to create a local variable with reference to a vector which is passed as an argument. If you simply say

local v

v=arg(1) **;;WRONG**

then the local vector v is destroyed when you routine exits, which will also destroy whatever vector was passed as arg(1).

However, if you use the Function statement or the Args statement to assign arguments to local variables, then the localcopy is done automatically.

v.insert(i,s)

Moves all elements starting at i up one, and then assigns the string s to element i of vec v. Indexes start at zero. If the vector is already at capacity and the growth is > 0, then vec will grow the vector by growth elements to accomodate the new element. Indexes i<0 or i>vec.capacity are in error and will elicit an error message from vec. Vec.length increases by 1.

v.insertnogrow(i, s)

Moves all elements starting at i up one, and then assigns the string s to element i of vec v. Indexes start at zero. If the vec.legnth is less than the capacity, it increases by one; otherwise, the last element in the vector is deleted.

v.delete(i)

Moves all elements starting at i down one, deleting the element at i. Vec.length decreases by 1.

v.deleteall

Deletes elements from vector v; vec.length set to 0.

v.grow(growth)

Adds growth elements to the capacity of v.

n=v.capacity

Sets n to current capacity of vector v.

len=v.length

Sets len to current length of v. The length is one greater than the index of the highest set element.

result = vec.maxvecs(n)

raises maximum number of vectors to n. Can be called at any time; existing vectors are preserved.

str = v.makelines

The elements of the vector are arranged in a single string as separate lines with \r\n between each line.

str = v.makewords(dstring)

The elements of the vector are arranged in a single string as separate words separated the string specified by dstring (default blank if omitted).

v.showmenu(maxtext,pos,autoclose,color, xoff, yoff) ;; or can omit any set of arguments at end

Creates and shows a menu built from entries of v. The index of the selected entry is returned; -1 is returned if the menu is dismissed without picking an entry. You can use *sep as an vector entry to show a menu separator, *colsep to start a new column, and *colsepline to start a new column with a separating line.

If present, the maxtext argument gives the maximum length of the labels in the menu. If pos argument is present, the menu is centered on the screen if pos is 1 and is centered under the mouse if pos is 2, is placed at the text cursor if pos is 3, and is placed at (0,0) if pos is 4; otherwise, it is shown at the mouse. If you include an argument autoclose set to 1, then menu is closed after a few seconds if the mouse is moved off of it. If you include an argument color, it gives the menu beckground color: a single integer (red in lowest byte) or a string of three blank separated numbers may be used to give the color (RGB). The xoff and yoff arguments give a pixel offset from the position given by pos. Use "" for arguments you do not want to specify when you want to specify a later one, eg

sel=v.showmenu("","","",color)

To see which mouse button was used to select for v.showmenu, use

win.lastmouse(1)

v.showmenu(...)

mouse=win.lastmouse(0)

Sorting and Searching Vectors

v.swap(i,j)

Exchanges elements i and j in the vector. This swap will make more effecient use of memory than if the swap is programmed directly (ie use swap instead of t=v[i]   v[i]=v[j] v[j]=t).

v.sort(flagdesc,flagcase)

Sorts v in ascending order. If flagdesc is "1", sort is descending. If flagcase is "1", case of letters is used; else it is ignored. Strings of 9 digits or less are treated as numbers for comparisons, so that "4" is less than "20".

v.sortstring(flagdesc,flagcase)

Same as vec.sort, except contents of v are always treated as strings, so that "4" is greater than "20".

v.doublesort(w,flagdesc,flagcase)

Sorts vector v in ascending order and keeps elements of vector w in corresponding order. For example, if v had four elements 3,1,4,2 and w had four elements "q", "a", "t", "X", then v.doublesort (w) would set v to 1.2.3.4 and w to "a", "X", "q", "t". If flagdesc is "1", sort is descending. If flagcase is "1", case of letters is used; else it is ignored. Strings of 9 digits or less are treated as numbers for comparisons, so that "4" is less than "20".

v.doublesortstring(w,flagdesc,flagcase)

Same as v.doublesort, except contents of v are always treated as strings, so that "4" is greater than "20".

v.binsearch(value)

The vector v must be sorted in ascending order. The service searches for value in the vector v and returns its index. If value is not found, -1 is returned.

v.binsert(value)

The vector v must be sorted in ascending order. The service inserts the value into the vector unless the value is already there. If the value is already there, it is not inserted. In both cases, the service results the index of the value in the vector.

Example

The following example reads in file c:\sortme.txt, sorts it, and writes it to c:\sorted.txt.

```
local han = file.open("c:\sortme.txt","r")
local thevec = vec.create(200,100)
for (i=0;1;i=i+1)
  line = han.readline
  if (han.eof)
    break
  thevec [i]=line
endfor
han.close
thevec.sort(v)
```

```
han=file.open("c:\sorted.txt", "w")
for (i=0; i<thevec.length; i=i+1)
  han.writeline(thevec [i])
endfor
han.close
```

Scripts

Maps

Overview

Maps are collections of strings. Each string in the map is accessed by another string, called its "key". You access the string with key "k" in the map called m with m["k"]. You add a new element to the map or change an existing element with

```
m["k"] = expr
```

You can use any expression to specify the key.

Maps use a hashing algorithm which provides very fast look-up of the string for the key.

One standard use for map is a dictionary: the key is the word to be defined and the string is the definition.

Maps are implemented by the map plugin.

Creating Maps

In the following descriptions, m is a map, k is any text string or expression, and val is any text string or expression.

```
m= map.create(size, useCase)
```

Creates a new map. The size should indicate the approximate number of elements in the map. Maps can hold any number of values, but quickest access takes place if `size` is greater than the forecasted number of map strings. `UseCase` is optional; if specified and set to 1, then the case of letters in keys will be used. For example, m["abc"] will be a different value from m["ABC"]. If useCase is omitted or not 1, then case of letters in keys is ignored so that m["ABC"] and m["abc"] would refer to same value.

Other Map Functions

```
m=m.destroy
```

Frees all memory assigned to m and its elements and frees the handle for use for another map. Returns 0 which you can then assign to m to indicate that m is no longer a valid map.

```
m.exists
```

Returns 1 if m is a valid map; 0 otherwise.

```
m[k]= val
```

Assigns the string expression val to key k of map m; if key k was not previously part of map, map.length increases by 1. You can also use m.set(key,val).

val=m[k]

Retrieves value for key k of map m. If there is no key k for the map, "" is returned. You can also use m.get(key).

LOCALCOPY

local m

m=map.localcopy(arg(1))

This service must be used if you want to create a local variable with a reference to a map which is passed as an argument. If you simply say

; WRONG

local m

m=arg(1)

then the local map m is destroyed when you routine exits, which will also destroy whatever map was passed as arg(1).

However, if you use the Function statement or the Args statement to assign arguments to local variables, then the localcopy is done automatically.

m.delete(k)

Deletes key k from map; map.length decreases by 1.

m.deleteall

Deletes all keys and values from map m; map.length set to 0.

m.length

Sets len to number of key-value pairs in map.

len=m.capacity

Sets len to number total number of entries in the lookup table, which is a power of 2 greater than the size specified by map.create.

len=m.slotused

Sets len to number number of slots used in the lookup table. Each key is hashed to an index in the lookup table. This service returns the number of indicies that have been used. It is possible for two keys to hash to the same slot index, in which case map creates a linked list of keys and values from this lookup slot. If a more than 90% of slots are used then, the initial size is too small. If the number of slots is much less than the length, then the map plugin is not doing a good job of hashing for your set of keys, meaning search time is being spent scanning linked lists.

m.restart

Used to list the elements of the map. Resets so that next usage of map.getnext will retrieve first key.

k=m.getnext

Allows you to retrieve all the keys in a map in no particular order. Use with m.restart to initialize and m.eof to check for no more keys. For example, the follow will dump all key-value pairs in a map m:

m.eof

Returns 1 if the m.length is 0 or if preceding call to map.getnext failed because there were no more keys to retrieve.

```
m.restart
for (1)
  key = m.getnext
  if (m.eof)
    break
  win.debug(key, m[key])
endfor
```

The order of retrieval is arbitrary and may change if you add or remove elements. Do not use change a map element while you are enumerating the map with getnext as this will restart the retrieval of elements to the beginning.

Example

; This example shows a simple dictionary of animals and family.

```
local m,key
m = map.create(100)
m["dog"] = "mammal"
m["cat"] = "mammal"
m["black widow spider"] = "arachnid"
m["beetle"] = "insect"

m.restart
for (1)
  key = m.getnext
  if (m.eof)
    break
  win.debug(key, ":", m[key])
endfor
```

The output of this example is:

```
dog : mammal
beetle : insect
```

cat : mammal

black widow spider : arachnid

Note that order of map.getnext is arbitrary and does not reflect order of creation of keys.

Scripts

Regular Expressions

Overview

Powerpro provides support for string matching and replacement using regular expressions based on the Regex Plugin. For detailed information on this plugin, see the regex.htm file in your plugin folder. This plugin in turn uses the PCRE library which provides Perl-Compatible Regular Expressions.

If you do not know how to use Regular Expressions, there are many tutorials available on the web.

Regular expressions use the backslash (\) to help indicate special processing. This will conflict with PowerPro's use of the backslash for the string escape character in the standard configuration. For this reason, you normally use ?" " type strings to specify regular expressions as literal strings.

Matching

The form

```
var[?"RegExp"]
```

returns a set of lines of text, with one line for each substring of var which matches the regular expression RegExp. If there are no matches, returns "" (which will be false in if statement). For example,

```
local alpha = "abracadabra"
lines = alpha[?"a."]
```

sets lines to "ab\nac\nad\nab". You can change separator with Exec SetRegex("sep=c").

If you want to match just the first occurrence, use the special comment (?#1) at the start of the regular expression, so that alpha[?"(?#1)a.] returns "ab"). If you have used Exec SetRegex to default to single match and you want this usage to match all, put the special comment (?#g) at the start of the regular expression.

Matching can be applied to any expression; for example

```
("abra"++remove("cadabra22", -2))[?"a."]
```

yields same results as above match against variable alpha.

Matching uses the regex plugin services matchg and match.

Replacing

The statement

```
var[?"RegExp"] = ?"str"
```

replaces all matches for RegExp in var with "str", which can be any expression, including the empty string to delete all occurrences of the matched string. The replacement string can contain regular expression back-references, such as \1 for the first matched substring, in which case the ?" " form should be used for literal strings to avoid conflicts with \ as the PowerPro string escape character. For example:

```
local alpha = "abracadabra"
alpha[?"a."] = "123"
```

sets alpha to "123r123123123ra". To replace just the first occurrence, use the special comment (?#1) at the start of the regular expression. If you have used Exec SetRegex to default to single match and you want this usage to match all, put the special comment (?#g) at the start of the regular expression.

You can also use the replace and replaceg functions for matching; here global is specified explicitly:

```
replace(str, pattern, repl)   // returns str with first match replaced by repl
replaceg(str, pattern, repl)   //returns str with all matches replaced by repl
str = str.replaceg(pattern, repl)   //dot syntax used to replace then assign back to str
str .= replaceg(pattern, repl)   //same as preceding by using .= operator
```

Replacing uses the regex services replaceg and replace.

Global Variable _RegExp_

Each time you use a regular expression, the global variable _RegExp_ gets set to an integer reflecting the result of the operations. The possible values are:

0 – full match: the pattern matched the complete string

1 – partial match: the pattern matched a part of the string

2 – no match

8 – error, invalid pattern (you will get a PowerPro error message as well)

9 – error, pattern or string is empty string (you will get a PowerPro error message as well)

Exec SetRegex

You can set the default for global versus single match/replace with

```
exec.setRegex("string")
```

If "string" contains "onematch", the default match is to the first occurrence only. If string contains "global", the default becomes matching or replacing all occurrences. If string contains "onematchonce", then the onematch flag is set for the next match/replace only.

To change the separator for multiple matches, use

```
exec.setRegex("sep=c")
```

where c is any sequence of one, two, or three characters, including special characters like \x01.

Scripts

Debugging Support for Scripts

Activating Debugging

PowerPro has features to help with script debugging. Use the Exec.ScriptDebug command to access them

```
Exec.ScriptDebug(1)     ;; turn on debugging
Exec.ScriptDebug(0)     ;; turn off debugging
Exec.ScriptDebug(-1)    ;; reverse debugging state
```

When debugging is turned on, any script which runs will use the debugging support. So when debugging, you normally stop all scripts except those you want to work with.

Using Debugging

Activating debugging stops execution of the current script (or if no script is executing, stops execution at the first line of the next script which is run). A dialog is displayed showing the script line which is about to be executed, but which has not been executed yet.

The dialog is resizable to show up to ten edit boxes for entering expressions. A single variable name could be used as an expression. These expressions will be evaluated by pressing the "Evaluate" button. If the "Evaluate on Displayl" checkbox is checked, then the expressions are automatically evaluated each time the dialog is displayed.

If the expression contains a variable which is not defined for this script, the expression result will be set to "Undefined variable: varname". No PowerPro error will appear.

The dialog also containts four buttons which control the restart of execution of the script:

Step – Execute current, displayed script line, then redisplay dialog

Run to break – Execute script lines until breakpoint is found; a breakpoint is either a comment line start with ;b! or a line specified in the Break At edit box.

Run to end – Executes rest of script. If current script was called by another script, resumes debugging upon return to the calling script.

Quit All – Ends execution of all running scripts.

Once shown, the dialog stays visible until closed. However, the dialog buttons are disabled if no script is running, although you can still enter information in the edit boxes even in this case.

You can also use

```
Exec.ScriptDebug("suspend")
```

to suspend script debugging until return from current script.

Breakpoints and Expressions

The ;b! comment offers a type of breakpoint. PowerPro looks for and processes breakpoints only when the **Run to break** button is pressed.

You can create conditional breakpoints with

```
;b! if (expr)
```

The breakpoint is taken only if the expression if true.

You can pre-set the expressions displayed in the debug dialog with

```
;b! digit(expr) digit(expr)
```

The single digit specifies which of the ten expression boxes to set (0 through 9). The expression text, which must be in parentheses, sets the contents of that box. You can combine conditional breakpoints and expressions:

```
;b! if (expr) digit(expr) digit(expr)
```

In this case, the expressions will be set (and the breakpoint taken) only if the expression after the if is true. For example

```
;b! if (counter>10 ) 0(alpha) 1(vec[counter])
```

As noted above, PowerPro only processes breakpoint statements when the **Run to break** button has been pressed. If you want to set the expression boxes in all cases, use

```
;e! if (expr) digit(expr) digit(expr)
```

The ;e! is always processed. If the if-expression is true, the expression boxes set the expression text the next time the debug dialog appears. The if-expression can be omitted in which case the expression boxes are always processed.

If you prefer, you can use

Exec.ScriptDebugField(digit, "expr")

to set the expression box given by the digit to the expression (which must be in quotes if not stored in variable).

You can also use the Break At edit box to enter breakpoints. Enter one or more blank-separated line numbers. PowerPro will break at these whenever the **Run to break** button is pressed. If you are debugging more than once script file at a time, you can enter name@n to indicate a breakpoint at line n in the script called name. The Break At edit box can be accessed programmatically: it is stored and loaded from the global variable _BreakPoints_.

OnScriptDebug Command List

If a command list called OnScriptDebug exists, it is run each time the debug dialog box appears, just before showing this dialog. The command list is called with five arguments

```
args fullpath name line breaktext
```

fullpath – Full path to currently executing script's file

name – Name of currently executing script.

line – Line number of currently executing script.

breaktext – If debug dialog being shown by a breakpoint, contents of that breakpoint line.

Note that you cannot use internal functions like ScriptLine for these values, as these functions would return the value for the OnScriptDebug command list.

*Info labels

Using *Info labels

dynamic text for buttons, menus, tray icons and tool tips

You can display dynamic text which monitors time, date, system resources, and other information. This text can be displayed on button labels, as menu item labels, in the system tray replacing the clock, or as tool tips on buttons, menus or tray icons.

Note: *Info is **not** a built in Command. It is only used in the Name field of an item, or in the Tooltip field, not in the Command field.

configuration

Set the start of the item label or tool tip to *Info and then use the Info button in the top right of the item configuration dialog to select a dynamic resource keyword to add to the end of the *Info display. Keywords are replaced by the corresponding system value.

There are lists of the three types of *Info keywords: time/date, resources, and other like clipboard contents, virtual desktop name, variable contents, free disk space, timer value.

You can also put unchanging text on a *Info display by putting it in quotes (e.g. "any text"). Only alphanumerics need be put in quotes; special characters like % or / do not need to be put in quotes.

The case of keywords is important. Most keywords are in lower case, except for a few time/date keywords (e.g. MMMM, HH).

For bar labels or tray clock text, use the width field on the command list item to make sure there is enough room to display the text as it is updated. Use a positive large width or a negative width (which sets the buttons width to the initial *Info size plus the absolute value of the width).

For continuously displaying labels on bars, *Info displays are updated once per second.

examples

```
*Info gdi/user dunrate
```

shows gdi and user resources separated by slash then current download rate on DUN connection.

```
*Info yyyy MMM dd HH:mm:ss swap ppmem%
```

shows year, short month name, day number (with leading zero), minutes, seconds, swap file size, percentage of free memory. Could look like 1999 Sep 04 13:18:22 60 15%. Note special characters : and %.

```
*Info "c:" disk c "d:" disk d
```

Shows free space on disks c and d.

Date and time *Info labels

dynamic text for buttons, menus, tray icons and tool tips

Use *Info labels in the Name field of the command editing dialog to specify date or time display. Use these keywords and sequences of letters. The case of the letters is important.

Note: these special keywords can only be used in *Info labels. They are **not** built in functions and cannot be used as parameters of commands etc.

offset n	Add n minutes to current time before processing following date/time items; the number n can be positive or negative.
shortdate	Short date format (as set on Control Panel, Regional)
longdate	Long date format (as set on Control Panel, Regional)
d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month as digits with leading zero for single-digit days.
ddd	Day of week as a three-letter abbreviation.
dddd	Day of week as its full name.
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation.
MMMM	Month as its full name.
yy	Year as last two digits, but with leading zero for years less than 10.
yyyy	Year represented by full four digits.
w	week number with no leading zero.
ww	week number with leading zero if less than 10.
daynum	day number of year
time	time format (as set on Control Panel, Regional)
h	Hours with no leading zero for single-digit hours; 12-hour clock
hh	Hours with leading zero for single-digit hours; 12-hour clock
H	Hours with no leading zero for single-digit hours; 24-hour clock
HH	Hours with leading zero for single-digit hours; 24-hour clock
m	Minutes with no leading zero for single-digit minutes
mm	Minutes with leading zero for single-digit minutes
s	Seconds with no leading zero for single-digit seconds
ss	Seconds with leading zero for single-digit seconds
t	One character time marker string, such as A or P
tt	Multicharacter time marker string, such as AM or PM

Put characters in double quotes to avoid being scanned for date/time codes.

example

```
*Info "London" offset +360 ddd yy/MMM/dd h:mm:ss t
```

could show: London Sun 98/Sep/19 7:53:01 P

*Info labels

System resource *Info labels

dynamic text for buttons, menus, tray icons and tool tips

Use *Info labels in the Name field of the command editing dialog to specify resources to display. Use these keywords and sequences of letters. The case of the letters is important.

Note: these special keywords can only be used in *Info labels. They are **not** built in functions and cannot be used as parameters of commands.

gdi (95/98)	Displays the percentage of free GDI resources.
user (95/98)	Displays the percentage of free USER resources.
pkmem	Displays free physical memory in Kilobytes.
pmem	Displays free physical memory in Megabytes.
ppmem	Displays percent free physical memory in Megabytes.
vkmem	Displays free page file plus physical memory in Kilobytes.
vmem	Displays free page file plus physical memory in Megabytes.
cpu (95/98)	Percent CPU in use (approximate).
swap (95/98)	Swap file size in megabytes.
swapinuse (95/98)	Swap file size in use in megabytes.
pswapinuse (95/98)	Percentage of swap file in use.
dunin (95/98)	Kilobytes received since DUN modem connected.
dunout (95/98)	Kilobytes sent since DUN modem connected.
dunrate (95/98)	Running average of kilobytes received per second.
battery	Percent of battery power remaining; 255% means no information available
allbattery	The allbattery display consists of these three fields:

- percent of battery power remaining (255% means no information available)
- character + if battery charging, - if discharging, ? if unknown charging status
- AC if ac connected, DC if battery power being used; ?? if unknown.

notes

Any other text in the *info item label is displayed without change.

Resources displays are updated once per second.

If you are running win95 and the dun modem displays are not working, try upgrading to at least DUN driver 1.3 driver from Microsoft www.microsoft.com (Dun 1.3 comes with Win98).

examples

*Info user/gdi pkmemK

shows user and gui resources separated by a slash, them physical memory followed by "k"; sample output would be 78/80 123K

*Info gdi/user "Virtual:" vmem

displays the GDI and User resources separated by a slash, then the word "Virtual", then the free virtual memory in megabytes.

*Info labels

Other *Info labels

dynamic text for buttons, menus, tray icons and tool tips

Use *Info labels in the Name field of the command editing dialog to specify other items to display. Use these keywords and sequences of letters. The case of the letters is important.

Note: these special keywords can only be used in *Info labels. They are not Functions and cannot be used as parameters of commands etc.

clip n	Shows text on clipboard; use n to limit to first n characters.
subbarname	Current subbar name.
deskname	Current desktop name.
desknum	Current desktop number 1-9.
disk x	Free space for disk x; x is any letter.
timer x	Value of timer x in hours and minutes; x is any letter.
timerdays x	Value of timer x in days, hours, and minutes.
timersec x	Value of timer x in hours, minutes, seconds.
uptime	shows time since Windows started in hours:minutes
uptimesec	shows time since Windows started in days or hours:minutes:seconds
uptimedhm	shows time since Windows started in days:hours:minutes.
uptimedhms	shows time since Windows started in days:hours:minutes:seconds.
uptimedays	shows time since Windows started in days.
cdcurtrack	current cd track number
cdlasttrack	last cd track number
keylog	X if logging keys, empty otherwise.
keylogfile	name of currently open key logging file, empty otherwise.
expr(expression)	Evaluates the expression, e.g. expr (dunrate/10), and displays the result.

Although expr(...) can be used to mimic most of the other *info labels, you will save CPU time on your system if you use the *info label when it exists.

defaultprinter	name of default printer.

You can set the default printer via *Menu Folder Printers. Right click on the resulting menu.

Putting Controls in Buttons

Overview

You can insert Windows controls – edit, combobox, button, checkbox (done using button), progress bar, and slider into PowerPro buttons by starting the button label with *control and following it with keywords as described below. You can also host windows COM Controls in buttons.

The control is always sized to fit the button dimensions.

The background and text colors for the control are taken from the settings for the button, except for progress bars, which only use the colors if Windows Classic theme is set.

General Information

Set the label of the control as follows:

*control type keywords

where type is one of *edit*, *combobox*, *button, progress,* or *slider* and keywords are described in the following paragraphs. Note that you use *button* for checkboxes: one of the keywords autocheckbox or auto3state makes the button a checkbox.

For all keywords except sliders and progress, you can put text in quotes "initial text" or in a ?-literal, like ?=literal=, and that text will be the initial value of the control.

For sliders and progress bar, the keywords can be preceded by up to four numbers or expressions giving the starting setting, lower bound, upper bound, and tic frequency (this for sliders only).

You can also change and access the control with cl functions cl.getctrlhandle, cl.getctrlvalue, cl.setctrlvalue, cl. clearctrllist, cl. addctrllistitem, cl.setctrlURL.

When you press enter when a control has the window focus, the left command of the PowerPro button is executed (use shift-enter for multi-line Edit). The command is also executed when you close up a combo box, click a button, or change the state of a checkbox. You can use tab and shift-tab to navigate controls with the keyboard. You can use *Bar ToButton to move the focus to a bar.

Following are the permitted keywords for each Control

Edit

The following keywords can be used:

autohscroll – automatically scroll horizontally when end of edit box reached

autovscroll– automatically scroll horizontally when end of multi-line edit box reached

center – center text

left – left- justify text

lowercase –- lower case of text

multiline –- multi-line text box

nohidesel –- don't hide selection when edit loses focus

number – -only allow numerics

password –- show entered characters as stars

readonly –- do not allow text entry

right –- right justify text

uppercase –- upper case text

clientedge –- do not show boundary around control

Button

The following keywords can be used:

auto3state – the button is shown as an 3-state check box; returned values for cl.getctrlvalue are 0, 1, 2.

autocheckbox– the button is shown as an 2-state check box; returned values for cl.getctrlvalue are 0, 1.

pushlike – combine with autocheckbox to get a button that toggles pressed/up and returns values as a checkbox does

bottom – text on bottom

center – text centered

multiline – multi-line text label

rightbutton – text right justified

top – text at top

vcenter – vertically center text

clientedge –- do show boundary around control

In addition, for checkboxes, you can put a number 0, 1, or 2 or an expression in parentheses anywhere among the keywords to set the initial state of the checkbox (0 is unchecked, 1 is checked, 2 is grayed). Example to create a checkbox set initial to the Muted state with label "Mute":

*control button autocheckbox (muted) "Mute"

Comboxbox

The combobox requires that you select one of dropdownlist or dropdown keywords. You may also have to set the button height to see the whole drop list; start with zero and experiment with other values if that does not work.

You can set the initial text in the combo drop down by including it within quotes. Separate each entry with a vertical bar |. For dropdown, the first entry goes into the edit box. For example, "alpha|beta|gamma|delta" puts alpha in edit box, and has three dropdown items: beta, gamma, delta.

The following keywords can be used:

autohscroll – auto scroll text in edit

dropdown – show the drop down; do not forget this if you want to see the icon and the drop down list of items

dropdownlist – no edit box

lowercase – lower case input text

nointegralheight – show partial items in drop down

sort – sort items in dropdown alphabetically

uppercase – upper case input text

clientedge –- do not show boundary around control

scroll –- include vertical scroll on dropdown

border – draw border around control

For example, the following creates a combo box with drop down and initial value "choose"; note that the initial value will not appear in the combo box drop down list.

*control combobox "choose|alpha|beta|gamma" dropdown

Slider

Before the keywords of the slider control, you can put one to four numbers or expressions in parentheses which give the initial setting, the lower bound, the upper bound, and the tic frequency. If the last number is omitted, no ticks are placed except for those at the end.

The following keywords can be used after the numbers:

vertical – vertical slider

left – ticks on left

right – ticks on right

bottom – ticks on bottom

top – ticks on top

both – ticks on both sides

noticks – no ticks at all; also removes ticks at end points

For example, the following creates a slider with values 0 to 255, initial value set to volume, ticks every 20.

*control slider (volume) 0 255 20

Progress Bar

Before the keywords of the slider control, you can put one to three numbers or expressions in parentheses which give the initial setting, the lower bound, and the upper bound.

The following keywords can be used after the numbers:

vertical – vertical progress bar

For example, the following creates a progress with values 0 to 1000, initial value set to 100.

*control progress 100 0 1000.

You use the cl.setctrlvalue function to change the progress bar setting.

Auto Completion

You can use the following key words with edit controls or combo boxes (dropdown only) to add autocompletion to the file system or URLs.

autofile – autcomplete from file system

autourl – autocomplete from MRU or History URLsl

autourlhistory – autcomplete from hstiry URLs only

autourlmru – autcomplete from MRU URLs only

autoappend_off – force auto append off, regardless of registry setting

autoappend_on – force auto append on, regardless of registry setting

autosuggest_off – force auto suggest off, regardless of registry setting

autosuggest_on – force auto suggest off, regardless of registry setting

Windows COM Controls

If the control type is "ie", then the following text is taken as a URL and a browser control is placed in the button and navigates to that URL. If the control type starts with "{", an attempt is made to load a COM control with that GUID. If the control type contains a dot, an attempt is made to load a COM control with that progid.

Expressions and Functions

Expressions

purpose

Expressions can be used to represent a number or a string.

Every time a statement is executed, any expressions it contains are evaluated to obtain a number or a string, to be used in its place on that occasion.

usage

Expressions can be used:

to assign a value to a variable, using: var = expression

as the condition in if(expression) statements, and as the context (i.e. condition) of contextual hotkeys, context menus and bars, etc.

in the parameters of: plugin calls, and built in Functions.

in the parameters of built in commands but see Combining expressions, variables, and PowerPro commands.

to display dynamically changing information in the labels of bar buttons, menu items and tray icons, and in tooltips, using *Info expr(expression)

Apart from the following exceptions, expressions never appear as the first element of a statement, because each statement must start with a literal command -- either a PowerPro built in command, such as Window Close *Notepad , or a Windows command such as Notepad.exe.

exceptions: functions used as commands

Most built in Functions can only be used as a value in an expression. Exceptionally, PowerPro will allow the following Functions to be used alone as commands:

Functions starting with do, eval, messagebox, mci, inputdialog, assign

plugin calls

Plugin calls can be used as a stand alone command.

structure of expressions

An expression is a series of **values** and **operators**.

The simplest form of expression consists of just one value. Complex expressions consist of two or more values joined by operators.

Complex expressions are evaluated in this order: first each value is evaluated; then the operators are applied to those resolved values, from left to right taking into account any parentheses as well as the relative priority of operators (e.g. multiplication before addition so that 1+2*3 is 7).

Example: variable = string1 ++ string2

Everything after the = is one expression, in this case a complex expression.
That expression has two values string1 and string2 and one operator, ++
In this case, both of the values are user defined variables. Each will be resolved into its current value before being joined by the ++ operator.

Expressions can be nested by using an expression as one of the values in another expression.

Example: variable = string1 ++ length(string2 ++ string3)

values in expressions

A **value** in an expression can be one of the following

- a string literal enclosed in double quotation marks, like "abcd".
- a number, like 15 or –22.0; or 3.2e-6
- a variable, like a or x4;
- a reference to a vector or map, like v[j] or m["key"];
- a regular expression match, like var["matchme"];
- a plugin call which returns a value, like date.add(date, 5)
- a call to run a script which returns a value, like run.cmdlist(args) or runfile.scriptfile(args) or .scriptfile(args)
- a keyword which returns a value, like date [That is a built in Function which has no parameters]
- a Function call which returns a value, like length(string) or window("caption", "=notepad") or str.squeeze(" ") [That is a built in Function with parameters]
- an expression can act as a value inside another expression

functions as values

The built in Functions are listed in Functions by category and also in Functions alphabetic list.

numbers as values

You can enter numbers with or without quotes:

alpha = "5" + 7

will assign 12 to alpha.

However, floating point numbers must always appear without quotes.

strings or numbers

PowerPro can work with either strings or numbers in expressions and will try to use whatever is appropriate to the operator. It will use floating point if any value in an operation is floating point. For example:

"-12" + 9	yields the number 3
"-12" join "ab"	yields the string "-12ab"
5.1*math.sqrt(4.0)	yields the floating point number 10.2

Internally, PowerPro stores all values except floating point numbers as strings. Floating point numbers can be turned back to strings with the floattostring (or ftos) function.

operators in expressions

Priority level zero, unary not and dot

. Dot is used with plugins

not or ! Result is 1 if following value is 0, otherwise result is 0

~ Result is bitwise not of following value

Priority level one, products

* Multiply two numbers. 5*2 is 10.

/ Divide; result is truncated to nearest integer. 12/5 is 2.

% Remainder after division; 15 % 4 is 3.

Priority level two, sums

+	Addition. 2 + 2 is 4.
-	Subtraction. 15 - 8 is 7.

- Priority level three, bitwise shift

<<	Shift bits of number left.
>>	Shift bits of number right.

Priority level four, join

++	Join two strings. "abc" ++ "defg" is "abcdefg"

Priority level five, comparison operators

(Case of letters in strings is ignored in all comparisons, e.g. "abc" is equal to "AbC")

==	eq	Result is 1 if values are equal; 0 otherwise. If both sides of == are numbers, compares numerically, e.g. "099" == "99" is 1 for True
===		Result is 1 if values are equal; 0 otherwise. === always compares as string, e.g. "99"=="099" is 1 but "99"==="099" is false.
!=	ne	Result is 1 if values are not equal; 0 otherwise.
!==		Result is 1 if values are not equal; 0 otherwise. Always compares as strings, e.g. "99" !== "099" is 1.
<	lt	Result is 1 if first value is less than second value; 0 otherwise.
<=	le	Result is 1 if value is less than or second equal to value 2; 0 otherwise.
>	gt	Result is 1 if first value is greater than second value ; 0 otherwise.
>=	ge	Result is 1 if first value is greater than or equal to second value; 0 otherwise.

Priority level six, and

&&	and	Result is 1 if both values are non zero; 0 otherwise
&	and	Result is bitwise and of both values

Priority level seven, or

||	or	Result is 1 if either value is non zero; 0 otherwise
|	or	Result is bitwise or of both values
^^	xor	Result is 1 if one value is one and the other zero; 0 otherwise
^	xor	Result is bitwise xor of both values

The grouping in the above table shows the priority of the operators. Operators nearer the top of the table are executed before those with a lower priority. For example, * is executed before + and -. In turn + and – are executed before the relational operators like ==. The operator or has the lowest priority.

Note that you have a choice of symbol or keyword for the relational and logical operators (e.g. == *or* eq, & *or* and). This is helpful if you have set one of the special characters to have another meaning to PowerPro, e.g. if you have set & to be the "expression follows" character.

Operators which consist solely of alphabetics, like and or not, must be preceded and followed by a blank when used.

Note that && and || are fully evaluated unless the shortcut evaluation option on Setup|Advanced|Other is checked; by default, it is checked.

string delimiter

By default, literal strings in expressions are delimited by the double quote character.

Alternatively, you can specify a different delimiter, for use within a single string. Use a question mark followed by any non-blank character; that character will then be the string delimiter, throughout that string. There are no escape sequences recognized in such strings.

Example: `var = ?+\"+`

assigns var the string comprising a backslash followed by a double quote.
?+ acts as the opening delimiter and sets the next + to act as the closing delimiter.

Any character that is not in the string can be used as a delimiter. Within the string, ? does not have that special meaning. This format is useful for strings that have nested quotes or backslashes that would be awkward to precede with the \ escape, which would be needed for a string delimited by double quotes.

escape character

You use the character \ as an escape character in strings in expressions. First, gray check *Use quote ' for escape in expression strings (gray backslash)* on *Setup >Advanced >Characters*. This will be done automatically if you select the standard configuration. Then you can use the following sequences in PowerPro strings.

\n	newline
\r	carriage return
\t	tab
\"	double quote
\\	backslash
\(expr)	replaced by the value of the expression; thus "abc\(expr)xyz" is a shorter alternative to the equivalent "abc"++(expr)++"xyz". The expr cannot contain a ".
\&	expression follows character (replace & by whatever character you use)
\xhh	\x followed by 2 hexadecimal digits is replaced by the ASCII character given by those digits; e.g. \x44 gives D. Exactly 2 digits must follow \x.
\dnnn	\d followed by 3 decimal is replaced by the ASCII character given by those digits; e.g. \d069 gives E. Exactly 3 digits must follow \d.

In *Setup Advanced Characters*, you can set your own escape character, instead of the standard \

examples of using expressions

```
x = 5 + 2 *length "abc"
```

is an abbreviated form of the older style command:

```
*Script assign x 5 + 2 * length "abc"
```

which assigns 11 [i.e. 5 + 2*3] to variable x.

The value length "abc" is evaluated first, then the *, then the +

```
str = select ("abcdef" , 3) ++ "123"
```

assigns "abc123" to variable str.

```
if (alpha >= 5 or select(beta, 2) == "ab")) do
```

executes following statements if the expression inside if(expr) is true (value is not zero).

```
do("c:/" ++ foldername ++ "/" ++ filename)
```

a do(expression) statement allows an expression to be executed as a command.

Expressions and Functions

Functions by category

a list of PowerPro's built in functions, sorted by category

Operators used in expressions are listed in Expressions.

note: return values

Functions can return a string or a number or a binary value.
This is indicated in the list by using one of these return variables

s = when a string is expected as the returned value.

n = when a number [usually a +ve or -ve integer] is expected.

b = when only either 1 or 0 is expected [b for Binary, or Bit, or Boolean]

In actual use, you would either substitute your own return variable

example: title1 = **caption**

or use the function as the condition of an if() statement

example: if(**anywindow**(*notepad))do

or use the function in a parameter

example: clip.textpaste(**shortdate**)

note: parameters of functions

All functions which take parameters are shown in the list with following parentheses: function(parameters). Functions shown here without parentheses do not take parameters.

list of functions

where necessary, there is a link to a detailed description of the function's syntax and usage

time

s = xtime	Time in control panel regional time format
n = time	Time as 6 digit string hhmmss, 24 hour clock, eg. 140515.
n = timesec	number of seconds since midnight January 1, 1970
s = timezone	Name of current timezone.
n = uptime	Number of seconds Windows has been running
s = formattime(format,time)	Formats a time; see formatdate() and formattime() for details.

date

s = date	Todays date as 8 digit string yyyymmdd (e.g. 2000 12 31).
n = dayofweek	Day of week as single digit (Sunday is 0, Monday 1, etc.)
n = dayofyear	Day of year (eg 35 is Feb 4).
n = unixtime	number of seconds since midnight January 1, 1970 (same as timesec)
s = shortdate	Todays date in control panel regional short date format
s = longdate	Todays date in control panel regional long date format
s = formatdate(format,date)	Formats a date; see formatdate() and formattime() for details.

Note: You can also use the date plugin to manipulate dates.

timers, scheduler, alarms

b = timerrunning(t)	Returns 1 if timer t is running; 0 otherwise. See *timer.
n = timer(t)	e.g. timer("c") or timer(3) return the current value of timer c
n = lastidletime	Length of last idle period in which an alarm occurred

mouse

n = xmouse	Horizontal mouse position in pixels (0 for left edge of screen)
n = ymouse	Vertical mouse position in pixels (0 for top edge of screen)
n = xcursor	Horizontal text cursor position in pixels (0 for left edge of screen)
n = ycursor	Vertical text cursor position in pixels (0 for top edge of screen)
b = mouseleft	Value 1 if left mouse button is down, 0 otherwise.
b = mousemiddle	Value 1 if middle mouse button is down, 0 otherwise.
b = mouseright	Value 1 if right mouse button is down, 0 otherwise.
b = mouse4	Value 1 if mouse 4 button is down, 0 otherwise.
b = mouse5	Value 1 if mouse 5 button is down, 0 otherwise.

keyboard

b = shift	Value 1 if shift key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
b = ctrl	Value 1 if ctrl key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
b = alt	Value 1 if alt key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
b = win	Value 1 if windows key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
s = keylog	Returns "X" if logging; zero otherwise
s = keylogfile	Name of currently open key log file. Returns "" otherwise
n = lastmousekeytime	Time since last mouse or keyboard action (in milliseconds since Windows start)

clipboard

s = clip	1st line of clipboard. Use clip plugin to access whole clipboard.
b = cliptrackon	Returns 1 if clip tracking is on; 0 otherwise
s = lastclipname	File name of last captured clipboard item
s = lastclippath	Full path to last captured clipboard item

file, disk

Note: you can also use the file plugin to manipulate files

b = validpath(string)	Value is 1 if following string is a valid path to a file; string may contain wildcards: * and ?
n = recycleItems	total number of items in recycle bin on all drives
n = recycleSize	total size of recycle bin on all drives in KB
s = readline("filepath",n)	reads the nth line from text file filepath.Line numbers start at 1. e.g. var=readline ("c:/path/path.txt",3) assigns third line of file to var. Flag 0 is cleared by readline if a line is read and is set if end of file is reached before desired line number.
b = mounted(L)	Result is 1 if following string's first letter is a removable disk
n = diskspace("kfree","c:")	Returns free kilobytes on drive C: Use any drive letter. Instead of "kfree", you can use "kuser" (space available to current user in kilobytes), "ksize" (size of drive in kilobytes), and "mfree". "muser", "msize" for the same quantities in megabytes.
s = pickfile(path,title)	Returns a full file path from a dialog of names only.

powerpro

s = disk	Letter of disk where PowerPro was run from [e.g. c]
s = pprofolder	Folder of PowerPro configuration [including terminating \]
s = pcfname	Filename and extension of current configuration file.
n = pproversion	Version of powerpro as four digit integer, e.g. 3310.

s = Subbarname	Name of current subbar
n = StandardConfiguration	Returns 1 if *"Use standard configuration"* is checked
b = Context	Set to 0 by PowerPro to indicate the first file ...
b = ContextLast	Set to 1 by PowerPro to indicate the last file is being processed, of a group of selected files to which an Explorer context menu item has been applied.
s = SendMessage	Set by some Window SendMessage / PostMessage commands
s = _file_	Stands for the selected file, in the commands for a context menu, folder buttons, menu folder, and for files dropped on buttons.
n = lastbuttontype	set from last bar button pressed: 0 if normal, 1 active, 2 tray, 3 folder.
s = onerrorsetting	set from last bar button pressed: 0 if normal, 1 active, 2 tray, 3 folder.

s = alldesknames	All desktop names, as multi-line string with one name per line.
b = deskempty	1 if current vdesk empty; 0 otherwise
s = deskname	Name of current virtual desktop
n = desknum	Number 1-9 of current virtual desktop.
b = vdeskempty(vdeskname)	Returns 1 if vdesk named by followed string is empty; returns 0 if it is not; returns 2 if it does not exist. Vdesk name can be a single digit n for the nth desktop, desktop name, "" for current desktop, 0 for list of locked windows. Example: vdeskempty 1 examines first desktop. Example: vdeskempty x2 examines desktop whose name in variable x2.
b = vdeskhaswindow("vdesk","captionlist")	Returns 1 if vdesk contains a window matching captionlist. vdesk can be a single digit n for the nth desktop, a desktop name, "" for current desktop, 0 for list of locked window.

system

n = uptime	Number of seconds Windows has been running
n = user	User resources [95/98 only]
n = pmem	Percent free memory
n = gdi	GDI resources [95/98 only]
n = cpu	Cpu usage [98 only]
s = defaultprinter	Name of default printer
s = acdc	"A" if running on ac power; "D" otherwise
n = batterypercent	Percent battery power left
n = threadcount	Count of active threads.
s = username	The name of the currently signed-on user
n64 = perfcount	Returns the Windows performance counter.
n64 = perffreq	The frequency, in counts per second, that perfcount is incremented. These are both 64 bit integers, so you have to use the int64 plugin to work with them. You can use these functions to do very accurate timings.
s = windowsversion	four blank-separated numbers: major version, minor version, build, platform id

modem, internet

s = modem	Returns 1 if dial up connected; 0 otherwise
n = dunidle	[95/98] seconds since last character received over dial-up

n = dunrate	[95/98] Dial up rate in characters per second
b = dialupname	Name of current dialup connection. Returns 0 if not connected
s = browserDomain	Domain in current browser window
s = browserSubdomain	Domain and subdomain in current browser window
s = browserURL("brser")	URL in current browser window where browser name given by "brser"; if omitted, uses value from Exec.Setbrowser. Valid values include iexplore, netscape, firefox, maxthon, mozilla.

gui, desktop

n = xscreen	Screen width in pixels. Current horizontal resolution.
n = yscreen	Screen height in pixels. Current vertical resolution.
s = saver	Screen saver file name
b = saveractive	Value 1 if saver active; 0 otherwise. Does not see "Blank" saver.
b = saverenabled	Value 1 if screen saver enabled; 0 otherwise.
n = savertime	Time in seconds to screen saver start.
s = paper	Current wallpaper file name

sound

n = volume	Current master volume (0 - 255); can be set by *Exec VolumeAll
b = muted	Set 1 if muted sound; set 0 otherwise.
n = cdcurtrack	Current CD track
n = cdlasttrack	Last CD track
[r=] mci(string)	Execute mci command. See MCI function.

strings

These are brief references. See String functions and operators for details

n = abs(val1)	Absolute value of number.
s = case(keyword,string)	Changes string case.
n = crc32(string)	CRC32 of string; use === to compare crc's to force string comparison
s = eval(string)	Evaluates string as an expression and returns results. Eval allows you to build expressions as strings and then evaluate them
s = esc(?+string+,?+c+)	Applies escape character processing based on c to string
s = env(string)	Value of environment variable named by string.
s = fill(string1,string2)	Combines two strings.
s = floattostring(x)	Returns string representation of floating point number. Optionally, include second argument "%n.mf" for field width n and m digits after decimal. Can use short form name ftos.
n = index(string1,string2)	Index of second string within first.
s = join(string1,string2)	Joins two strings. join ("abc", "defg") is "abcdefg" Same as "abc" ++ "defg"
n = length(string)	Compute length of string; length("abc") is 3
s = line(string,n)	Scans the string and returns the nth line. Lines start at 1.
n = line(string,0)	Returns the number of lines in the string.
s = min(val1,val2)	Minimum of two values eg: min("abcd", vstring)
s = max(val1,val2)	Maximum of two values eg: max(var1, 7)
s = match(str, pat)	Regular expression match of pat to str; returns first match.
s = matchg(str, pat)	Regular expression match of pat to str; returns all matches.
s = nextword(string, "var", delims)	Returns the next word as delimited by characters in delims (defaults to blank and tab if omitted). The contents of the variable named var are replaced by all the characters after

	this first word. For example, nextword("abc*de*efgh","rest","*") returns "abc" and sets the variable rest to "* de*efgh").
s = remove(string,number)	Removes n characters from string. remove ("abcd", 2) is "cd". Use a negative number of remove characters from the end of a string. remove("abcd", -3) is "a"
s = repeat(string,n)	Returns string formed by repeating first argument n times
s = replacechars( )	Replaces character[s] by specified replacement char
s = replace(str, pat, rep)	Returns str with first match of pat replaced by rep.
s = replaceg(str, pat, rep)	Returns str with all matches of pat replaced by rep.
s = rotate (string,n,pre,min)	Rotates string n chars if it is longer than min.
n = strcoll(string1,string2)	Compares two strings using local collating sequence; returns -1 if first less than second, 0 if first equal second, 1 otherwise
n = stricoll(string1,string2)	Compares two strings using local collating sequence, ignoring case of letters; returns -1 if first less than second, 0 if first equal second, 1 otherwise
s = select(string,number)	Selects n characters from string.
s = squeeze(s1,s2)	Replace runs of characters in s2 from s1
x = stringtofloat(string)	Convert string to floating point.
s = translate(s1,to,from)	Replaces s1 chars which are in to by corresponding from characters
s = trim(s1,out,n)	Removes charactions in out from s1.
n = revindex(string1,string2)	Index of last occurrence of string2 within first See also index()
s = word(string, n, delims)	Scans the string and returns the word given by n; words are separated by any character in delims; if delims omitted, uses blank or tab as separator
n = word(string, 0, delims)	Returns the number of words in the string; words are separated by any character in delims; if delims omitted, uses blank or tab as separator

see also the PickString(lines,title) input dialog described in Input dialogs.

scripts, variables, flags

s = scriptfolder	Folder of currently executing script.
s = scriptname	Name of currently executing script.
s = scriptpath	Script path as set by most recent *Script Path.
s = arg(n)	Returns nth argument in the call to the script. Only used inside a script called with arguments; using runfile.script(arg1,arg2,…) or run.script(arg1,arg2,…). arg(1) is first argument. arg(0) is the number of arguments.
[s=] assign(var1,var2)	Assigns second string to variable name given by first string. eg assign("var1", "xx") assigns "xx" to var1. Then assign(var1, "000") would assign "000" to variable xx. Result of operation is the assigned value (ie second value).
s = allglobals	Returns a string with the names of all global variables, one name per line. e.g. alpha\rbeta\rgamma\r would be the result if there were three global variables with the shown names.
s = allstatics	Returns a string with the names of all static variables for the script in which the function is used, one name per line. e.g. alpha\rbeta\rgamma\r would be the result if there were three static variables with the shown names.
s = cb("entry",args)	Returns a string that can be used as a callback in dialogs or events using the call function. The returned string is

"call("++?`entry, args`++")", which allows a full or partial path as the called point and also allows @ entry points such as cb("@xxx"), where xxx is assumed to refer to current script. Use up to 15 arguments. Each arg is separately quoted by ? in the result of cb. Note that the arguments are evaluated and the values of the arguments are passed as the quoted strings. This is needed since the callback will normally occur outside the scope of the call to cb.

s = call("filepath@entry",args)	Runs script file given by filepath and (optional) entry with following lists of args. The filepath and entry may be given by any expression. A fully-qualified file path may be given, or a relative one, in which case the usual algorithm for finding the script file is used. The call function returns the result from the called script.
s = if(condition,result)	The result is the second string if the first string is true. Else result is "". Unlike the If(condition)do statement, which is used at the start of a line, this function is used in parameters.
s = ifelse(expr,string1,string2)	With three operands: returns s1 if expr is true (not 0 or "") and s2 it if is false.
s = ifelse(string1,string2)	With two operands: the second string should contain a comma; result is everything up to this comma if first string is not zero or empty and everything after the comma otherwise. e.g. ifelse (10<20,"alpha,omega") is "alpha".
b = not(expr)	Result is 1 if following expression is zero or empty string; else 0
b = pproflag(n)	Returns value of nth flag

do(filepath, params[, workfolder[, howstart]])

executes an external command

do(ppcmd.action(params))

executes a built in command using Expression Syntax

do("ppcmd", "action and params", "keywords")

executes a built in command using Literal Syntax

do() statements enable expressions to be used for a command and its parameters. see Combining expressions, variables, and PowerPro commands about using do()

user inputs, onscreen messages

These are brief references. Full details of input functions are in Input dialogs.

Note: If Cancel is pressed in an input dialog, flag 0 is cleared. Otherwise flag 0 is set.

setnextdialogpos(left,top,width,height) Sets position, size of next input dialog.

s = pickfile(path,title)	Returns a full file path from a dialog of names only.
s = pickstring(lines,title)	Select a string from a dialog list.
s = _PickedLine_	The index of the last string chosen from a PickFile or PickString dialog. Also set after Menu command to 1 if item picked; 0 if menu dismissed without picking an item.
s = filemenu(filespec,"pos")	Displays a file as a menu; the selected item is the result. Second parameter is optional. If present, it gives positioning keywords like centerscreen as used for menus.
s = inputtext	Prompts for input text. Same as input(title) but you cannot set title
s = input(title)	Prompts for input using title for the input dialog's title. To limit input length, start title with =n, e.g. input "=15Enter up to 15 chars". Result if whatever input is entered. If Cancel pressed, result is ""

s = inputcancel(title)	Same as input, except that if Cancelled the running script quits
n = inputcolor(def)	Displays color dialog and returns results as single number; def if present is default color for opened dialog
n = inputdate	Displays calendar calculator dialog and returns selected date as 8 digits yyymmdd
n = inputdatetime	Displays calendar calculator dialog and returns date and time as 14 digits yyymmddhhmmss
s = inputdefault("default","title")	Prompts for input string; preset with default value
[s=] inputdialog("var1=title1, var2=title2, [...var6=title6], "dialogtitle")	Details
s = inputfolder(path)	Displays folder browse dialog and returns results; path is optional starting point.
s = inputpath(path)	Displays file browse dialog and returns results; path is optional starting point.
s = inputpathmulti(path)	Displays file browse dialog and returns results, allowing multiple files to be selected; path is optional starting point. Clears flag 0 if cancel pressed; sets it otherwise. The returned result consists of a string with many lines of text. The first line is the path to the folder containing the selected files. The remaining lines are each a file name.
[n=] messagebox ("layout", "text", "title")	Shows a message box. MessageBox function
*Exec Prompt flag/var text	Shows a Yes/No[/Cancel] dialog. Prompting for Yes/No.

windows, tasks, processes

s = caption	Caption [title] of current foreground window
s = captionunder	Caption of window under the mouse
b = visiblewindow(captionlist)	Result is 1 if caption list matches any visible window
b = anywindow(captionlist)	Result is 1 if caption list matches any window, including hidden.
b = activewindow(captionlist)	Result is 1 if caption list matches the currently active window
s = window("pos",captionlist)	Returns the position, size, min/max/visible/topmost state, caption, exename, exe full path, window class, or window handle of any window on the screen or a specifice vdesk. pos is one of "left", "top", "bottom", "right", "height", "width", "minimized", "maximized", "visible", "topmost", "caption", "exename", "exefullpath", "class", "anywindow", "visiblewindow", "firstwindow" (or "vdesk" see below) caption is a caption list to specify the window (e.g. *notepad* for notepad). You can also use "active" as the caption to select the foreground window, "under" to select the window under the mouse, and "Taskbar" to access the Taskbar.
s = window("vdesk", vdeskname)	Returns list of handles of windows on vdeskname
n = lastactivehandle	Handle to last window selected by active button
n = lastautorunhandle	Handle to last window selected by autorun item
n = lastnotehandle	Handle to last active note window.
n = lasttrayid	Tray icon id of last tray icon button pressed.
n = lastscancode	Scan code of last hot key (plus 256 if extended key bit set)
n = lastVKcode	Virtual Key code of last hot key
s = exefilename	File name [no path, no .exe] of exe file for current foreground window
s = exefullpath	Full path to exe name for current foreground window
s = currentdir	path to working folder of current foreground window
n = processcount	Count of active processes.

n = threadcount	Count of active threads.

random number

n = random(n)	Returns a random integer between 0 and n-1

Expressions and Functions

Functions alphabetic list

PowerPro's built in functions

See above for the same list, sorted by category.
That topic also explains the return variables shown in help, such as: s =

Operators used in expressions are listed in Expressions

s = _file_	Stands for the selected file, in the commands for a context menu, folder buttons, menu folder, and for files dropped on buttons.
s = _PickedLine_	The index of the last string chosen from a PickFile or PickString dialog. See Input dialogs topic. Also set after Menu command to 1 if item picked; 0 if menu dismissed without picking an item.
n = abs(val1)	Absolute value of number.
s = acdc	"A" if running on ac power; "D" otherwise
b = activewindow(captionlist)	Result is 1 if caption list matches the currently active window
s = alldesknames	All desktop names, as multi-line string with one name per line.
s = allglobals	Returns a string with the names of all global variables, one name per line. e.g. alpha\rbeta\rgamma\r would be the result if there were three global variables with the shown names. Usage
s = allstatics	Returns a string with the names of all static variables for the script in which the function is used, one name per line. e.g. alpha\rbeta\rgamma\r would be the result if there were three static variables with the shown names.
b = alt	Value 1 if alt key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
b = anywindow(captionlist)	Result is 1 if caption list matches any window, including hidden.
s = arg(n)	Returns nth argument in the call to the script. Only used inside a script called with arguments; using runfile.script(arg1,arg2,…) or run.script(arg1,arg2,…). arg(1) is first argument. arg(0) is the number of arguments.
[s=] assign(var1,var2)	Assigns second string to variable name given by first string. eg assign("var1", "xx") assigns "xx" to var1. Then assign(var1, "000") would assign "000" to variable xx. Result of operation is the assigned value (ie second value).
n = batterypercent	Percent battery power left
s = browserDomain	Domain in current browser window
s = browserSubdomain	Domain and subdomain in current browser window
s = browserURL("brser")	URL in current browser window where browser name given by "brser"; if omitted, uses value from Exec.Setbrowser. Valid values include iexplore, netscape, firefox, maxthon, mozilla.
s = call("filepath@entry",args)	Runs script file given by filepath and (optional) entry with following lists of args. The filepath and entry may be given by any expression. A fully-qualified file path may be given, or a relative one, in which case the usual algorithm for finding the script file is used. The call function returns the result from the called script.
s = caption	Caption [title] of current foreground window
s = captionunder	Caption of window under the mouse

s = case(keyword,string)	Changes string case. String functions and operators
s = cb("entry",args)	Returns a string that can be used as a callback in dialogs or events using the call function. The returned string is "call("++?`entry, args`++")", which allows a full or partial path as the called point and also allows @ entry points such as cb("@xxx"), where xxx is assumed to refer to current script. Use up to 15 arguments. Each arg is separately quoted by ? in the result of cb. Note that the arguments are evaluated and the values of the arguments are passed as the quoted strings. This is needed since the callback will normally occur outside the scope of the call to cb.
n = cdcurtrack	Current CD track
n = cdlasttrack	Last CD track
s = clip	1st line of clipboard. Use clip plugin to access whole clipboard.
b = cliptrackon	Returns 1 if clip tracking is on; 0 otherwise
b = Context	Set to 0 by PowerPro to indicate the first file ...
b = ContextLast	Set to 1 by PowerPro to indicate the last file is being processed, of a group of selected files to which an Explorer context menu item has been applied.
n = cpu	Cpu usage [98 only]
n = crc32(string)	CRC32 of string; use === to compare crc's to force string comparison
b = ctrl	Value 1 if ctrl key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
s = currentdir	path to working folder of current foreground window
s = date	Todays date as 8 digit string yyyymmdd (e.g. 2000 12 31).
n = dayofweek	Day of week as single digit (Sunday is 0, Monday 1, etc.)
n = dayofyear	Day of year (eg 35 is Feb 4).
s = defaultprinter	Name of default printer
b = deskempty	1 if current vdesk empty; 0 otherwise
s = deskname	Name of current virtual desktop
n = desknum	Number 1-9 of current virtual desktop.
b = dialupname	Name of current dialup connection. Returns 0 if not connected
s = disk	Letter of disk where PowerPro was run from [e.g. c]
n = diskspace("kfree","c:")	Returns free kilobytes on drive C: Use any drive letter. Instead of "kfree", you can use "kuser" (space available to current user in kilobytes), "ksize" (size of drive in kilobytes), and "mfree". "muser", "msize" for the same quantities in megabytes.

do(filepath, params[, workfolder[, howstart]]) executes an external command

do(ppcmd.action(params)) executes a built in command using Expression Syntax

do("ppcmd", "action and params", "keywords") executes a built in command using Literal Syntax

do() statements enable expressions to be used for a command and its parameters.

n = dunidle	[95/98] seconds since last character received over dial-up
n = dunrate	[95/98] Dial up rate in characters per second

s = env(string)	Value of environment variable named by string.
s = esc(?+string+,?+c+)	Applies escape character processing based on c to string
s = eval(string)	Evaluates string as an expression and returns results. Eval allows you to build expressions as strings and then evaluate them
s = exefilename	File name [no path, no .exe] of exe file for current foreground window
s = exefullpath	Full path to exe name for current foreground window
s = filemenu(filespec,"pos")	Displays a file as a menu; the selected item is the result. Second parameter is optional. If present, it gives positioning keywords like centerscreen as used for menus.
s = fill(string1,string2)	Combines two strings.
s = floattostring(x)	Returns string representation of floating point number. Optionally, include second argument "%n.mf" for field width n and m digits after decimal. Can use short form name ftos.
s = formatdate(format,time)	Formats a date; formatdate() and formattime().
s = formattime(format,time)	Formats a time; formatdate() and formattime().
n = gdi	GDI resources [95/98 only]
s = if(condition,result)	The result is the second string if the first string is true. Else result is "". Unlike the If(condition)do statement, which is used at the start of a line, this function is used in parameters.
s = ifelse(expr,str1,str2)	With three operands: returns s1 if expr is true (not 0 or "") and s2 it if is.
s = ifelse(str1,str2)	With two operands: the second string should contain a comma; result is everything up to this comma if first string is not zero or empty and everything after the comma otherwise. e.g. ifelse (10<20,"alpha,omega") is "alpha".
n = index(string1,string2)	Index of second string within first.
s = input(title)	Prompts for input using title for the input dialog's title. To limit input length, start title with =n, e.g. input "=15Enter up to 15 chars". Result if whatever input is entered. If Cancel pressed, result is ""
s = inputcancel(title)	Same as input, except that if Cancelled the running script quits
n = inputcolor(def)	Displays color dialog and returns results as single number; def if present is default color for opened dialog
n = inputdate	Displays calendar calculator dialog and returns selected date as 8 digits yyymmdd
n = inputdatetime	Displays calendar calculator dialog and returns date and time as 14 digits yyymmddhhmmss
s = inputdefault("default","title")	Prompts for input string; preset with default value
[s=] inputdialog("var1=title1, var2=title2, [...var6=title6], "dialogtitle")	Input dialogs
s = inputfolder(path)	Displays folder browse dialog and returns results; path is optional starting point.
s = inputpath(path)	Displays file browse dialog and returns results; path is optional starting point.
s = inputpathmulti(path)	Displays file browse dialog and returns results, allowing multiple files to be selected; path is optional starting point. Clears flag 0 if cancel pressed; sets it otherwise. The returned result consists of a string with many lines of text. The first line is the path to the

	folder containing the selected files. The remaining lines are each a file name.
s = inputtext	Prompts for input text. Same as input(title) but you cannot set title
[another input: *Exec Prompt flag/var text	Shows a Yes/No dialog.]
s = join(string1,string2)	Joins two strings. join ("abc", "defg") is "abcdefg" Same as "abc" ++ "defg"
s = keylog	Returns "X" if logging; zero otherwise
s = keylogfile	Name of currently open key log file. Returns "" otherwise
n = lastbuttontype	set from last bar button pressed: 0 if normal, 1 active, 2 tray, 3 folder.
n = lastactivehandle	Handle to last window selected by active button
n = lastautorunhandle	Handle to last window selected by autorun item
n = lastnotehandle	Handle to last active note window.
s = lastclipname	File name of last captured clipboard item
s = lastclippath	Full path to last captured clipboard item
n = lastidletime	Length of last idle period in which an alarm occurred
n = lastmousekeytime	Time since last mouse or keyboard action (in milliseconds since Windows start)
n = lasttrayid	Tray icon id of last tray icon button pressed.
n = lastscancode	Scan code of last hot key (plus 256 if extended key bit set)
n = lastVKcode	Virtual Key code of last hot key
n = length(string)	Compute length of string; length("abc") is 3
s = line(string,n)	Scans the string and returns the nth line. Lines start at 1.
n = line(string,0)	Returns the number of lines in the string.
s = longdate	Todays date in control panel regional long date format
s = max(val1,val2)	Maximum of two values eg: max(var1, 7)
[r=] mci(string)	Execute mci command. MCI function
[n=] messagebox ("layout", "text", "title")	Shows a message box. MessageBox function
s = min(val1,val2)	Minimum of two values eg: min("abcd", vstring)
s = modem	Returns 1 if dial up connected; 0 otherwise
b = mounted(L)	Result is 1 if following string's first letter is a removable disk
b = mouseleft	Value 1 if left mouse button is down, 0 otherwise.
b = mousemiddle	Value 1 if middle mouse button is down, 0 otherwise.
b = mouseright	Value 1 if right mouse button is down, 0 otherwise.
b = mouse4	Value 1 if mouse 4 button is down, 0 otherwise.
b = mouse5	Value 1 if mouse 5 button is down, 0 otherwise.
b = muted	Set 1 if muted sound; set 0 otherwise.
s = nextword(string, "var", delims)	Returns the next word as delimited by characters in delims (defaults to blank and tab if omitted). The contents of the variable named var are replaced by all the characters after this first word. For example, nextword("abc*de*efgh","rest","*") returns "abc" and sets the variable rest to "* de*efgh").
b = not(expr)	Result is 1 if following expression is zero or empty string; else 0

s = onerrorsetting	set from last bar button pressed: 0 if normal, 1 active, 2 tray, 3 folder.
s = paper	Current wallpaper file name
s = pcfname	Filename and extension of current configuration file.
n64 = perfcount	Returns the Windows performance counter.
n64 = perffreq	The frequency that perfcount is incremented. These are both 64 bit integers, so you have to use the int64 plugin to work with them. You can use these functions to do very accurate timings.
s = pickfile(path,title)	Returns a full file path from a dialog of names only.
s = pickstring(lines,title)	See Input dialogs
s = _PickedLine_	The index of the last string chosen from PickFile or PickString
n = pmem	Percent free memory
b = pproflag(n)	Returns value of nth flag
s = pprofolder	Folder of PowerPro configuration [including terminating \]
n = pproversion	Version of powerpro as four digit integer, e.g. 3310.
n = processcount	Count of active processes.
n = random(n)	Returns a random integer between 0 and n-1
s = readline("filepath",n)	reads the nth line from text file filepath.Line numbers start at 1. e.g. var=readline ("c:/path/path.txt",3) assigns third line of file to var. Flag 0 is cleared by readline if a line is read and is set if end of file is reached before desired line number.
n = recycleItems	total number of items in recycle bin on all drives
n = recycleSize	total size of recycle bin on all drives in KB
s = remove(string,number)	Removes n characters from string. remove ("abcd", 2) is "cd". Use a negative number of remove characters from the end of a string. remove("abcd", -3) is "a"
s = repeat(string,n)	Returns string formed by repeating first argument n times
s = replacechars( )	Replaces character[s] by specified replacement char
s = replace(str, pat, rep)	Returns str with first match of pat replaced by rep.
s = replaceg(str, pat, rep)	Returns str with all matches of pat replaced by rep.
n = revindex(str1,str2)	Index of last occurrence of string2 within first See also index()
s = rotate (string,n,pre,min)	Rotates string n chars if it is longer than min.
s = saver	Screen saver file name
b = saveractive	Value 1 if saver active; 0 otherwise. Does not see "Blank" saver.
b = saverenabled	Value 1 if screen saver enabled; 0 otherwise.
n = savertime	Time in seconds to screen saver start.
s = scriptfolder	Folder of currently executing script.
s = scriptname	Name of currently executing script.
s = scriptpath	Script path as set by most recent *Script Path.
s = select(string,number)	Selects n characters from string.
s = SendMessage	Set by some Window SendMessage / PostMessage commands
setnextdialogpos(left,top,width,height)	Sets position, size of next input dialog. See Input dialogs.

b = shift	Value 1 if shift key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
s = shortdate	Todays date in control panel regional short date format
s = squeeze(s1,s2)	Replace runs of characters in s2 from s1
n = StandardConfiguration	Returns 1 if *"Use standard configuration"* is checked
n = strcoll(string1,string2)	Compares two strings using local collating sequence; returns -1 if first less than second, 0 if first equal second, 1 otherwise
n = stricoll(string1,string2)	Compares two strings using local collating sequence, ignoring case of letters; returns -1 if first less than second, 0 if first equal second, 1 otherwise
x = stringtofloat(string)	Convert string to floating point.
s = Subbarname	Name of current subbar
n = threadcount	Count of active threads.
n = time	Time as 6 digit string hhmmss, 24 hour clock, eg. 140515.
n = timer(t)	e.g. timer("c") or timer(3) return the current value of timer c
b = timerrunning(t)	Returns 1 if timer t is running; 0 otherwise. See *timer
n = timesec	number of seconds since midnight January 1, 1970
s = timezone	Name of current timezone.
s = translate(s1,to,from)	Replaces s1 chars which are in to by corresponding from characters
s = trim(s1,out, n)	Removes characters in out from s1.
n = unixtime	number of seconds since midnight January 1, 1970 (same as timesec)
n = uptime	Number of seconds Windows has been running
n = user	User resources [95/98 only]
s = username	The name of the currently signed-on user
b = validpath(string)	Value is 1 if following string is a valid path to a file; string may contain wildcards: * and ?
b = vdeskempty(vdesk)	Returns 1 if vdesk named by followed string is empty; returns 0 if it is not; returns 2 if it does not exist. Vdesk name can be a single digit n for the nth desktop, desktop name, "" for current desktop, 0 for list of locked windows. Example: vdeskempty 1 examines first desktop. Example: vdeskempty x2 examines desktop whose name in variable x2.
b = vdeskhaswindow("vdesk","captionlist")	Returns 1 if vdesk contains a window matching captionlist. vdesk can be a single digit n for the nth desktop, a desktop name, "" for current desktop, 0 for list of locked windows.
b = visiblewindow(captionlist)	Result is 1 if caption list matches any visible window
n = volume	Current master volume (0 - 255); can be set by *Exec VolumeAll
b = win	Value 1 if windows key is down, 0 otherwise. For W2K and later, value is 1 if left down, 2 if right, 3 if both.
s = window("pos",captionlist)	Returns the position, size, min/max/visible/topmost state, caption, exename, exe full path, window class, or window handle of any window on the screen or a specifice vdesk. pos is one of "left", "top", "bottom", "right", "height", "width", "minimized", "maximized", "visible", "topmost", "caption", "exename", "exefullpath", "class", "anywindow", "visiblewindow", "firstwindow" (or "vdesk" see below)

	caption is a caption list to specify the window (e.g. *notepad* for notepad). You can also use "active" as the caption to select the foreground window, "under" to select the window under the mouse, and "Taskbar" to access the Taskbar.
s = window("vdesk", vdeskname)	Returns list of handles of windows on vdeskname
s = windowsversion	four blank-separated numbers: major version, minor version, build, platform id
s = word(string, n, delims)	Scans the string and returns the word given by n; words are separated by any character in delims; if delims omitted, uses blank or tab as separator
n = word(string, 0, delims)	Returns the number of words in the string; words are separated by any character in delims; if delims omitted, uses blank or tab as separator
n = xmouse	Horizontal mouse position in pixels (0 for left edge of screen)
n = xcursor	Horizontal text cursor position in pixels (0 for left edge of screen)
n = xscreen	Screen width in pixels. Current horizontal resolution.
s = xtime	Time in control panel regional time format
n = ycursor	Vertical text cursor position in pixels (0 for top edge of screen)
n = ymouse	Vertical mouse position in pixels (0 for top edge of screen)
n = yscreen	Screen height in pixels. Current vertical resolution.

Expressions and Functions

Calling Functions with Dot Syntax

Overview

The functions which operate on strings can be called with dot syntax so that

str.func(args) //calls func using str as first argument

can be used instead of

func(str, args)

There is no difference in the result. The dot format is a matter of personal preference. However, it is easier to read if you have a series of calls which would by nested using the traditional syntax, such as

file.readall(path).line(3).squeeze(" ").length // length of third file line after blanks squeezed

which is equivalent to

length(squeeze(line(file.readall(path),3), " "))

The dot syntax can also be used with .= as in

myVec[index] .= translate("abc", "def")

which is equivalent to

myVec[index] = myVec[index].translate("abc", "def")

(Of course, ordinary variables as well as vector references can be used).

In addition to the functions in the table below, the dot operator provides these short forms:

str.empty   //same as str.length==0

str.numwords   //same as str.word(0)

str.numlines   //same as str.line(0)

str.upper   //same as str.case("upper"); also can use lower, tonum, fromnum, tonumx, fromnumx, sentence, title, acronym

String Functions Compatible with Dot

n = string.anywindow	Returns 1 if string is matches any window; 0 otherwise. String must not be a window handle (which would call win plugin).
n = string.activewindow	Returns 1 if string is matches active window; 0 otherwise. String must not be a window handle (which would call win plugin).
n = string.visiblewindow	Returns 1 if string is matches any visible window; 0 otherwise. String must not be a window handle (which would call win plugin).
n = string.crc32	CRC32 of string; use === to compare crc's to force string comparison
s = string.case(keyword)	Changes string case.
s = string.eval	Evaluates string as an expression and returns results. Eval allows you to build expressions as strings and then evaluate them
s = string.env	Value of environment variable named by string.
s = string.esc(?+c+)	Applies escape character processing based on c to string
s = fill(string1,string2)	Combines two strings.

s = filespec.filemenu("pos")	Displays a file as a menu; the selected item is the result. Second parameter is optional. If present, it gives positioning keywords like centerscreen as used for menu
s = string1.fill(string2)	Combines two strings.
s = x.ftos(format)	Returns string representation of floating point number. Optionally, include second argument "%n.mf" for field width n and m digits after decimal. Can use short form name ftos.
n =string1 index(string2)	Index of second string within first.
s =string1.if(string2)	Returns string2 if string1 non empty or zero; "" otherwise.
s =string.ifelse(string1, string2)	Returns string1 if string non empty or zero; string2 otherwise.
n = string.length	Compute length of string; length("abc") is 3
s = string.line(n)	Scans the string and returns the nth line. Lines start at 1.
n = string.line(0)	Returns the number of lines in the string.
s = val1.min(val2)	Minimum of two values eg: min("abcd", vstring)
s = val1.max(val2)	Maximum of two values eg: max(var1, 7)
s = str.match(pat)	Regular expression match of pat to str; returns first match.
s =str. matchg(pat)	Regular expression match of pat to str; returns all matches.
n = string.mounted	Returns 1 if string starts with mounted disk volume.
s =string. nextword("var", delims)	Returns the next word as delimited by characters in delims (defaults to blank and tab if omitted). The contents of the variable named var are replaced by all the characters after this first word. For example, nextword("abc*de*efgh","rest","*") returns "abc" and sets the variable rest to "* de*efgh").
s = string.remove(number)	Removes n characters from string. remove ("abcd", 2) is "cd". Use a negative number of remove characters from the end of a string. remove("abcd", -3) is "a"
s = string.repeat(n)	Returns string formed by repeating first argument n times
s = string.replacechars(s,t)	Replaces character[s] by specified replacement char
s = str.replace(pat, rep)	Returns str with first match of pat replaced by rep.
s = str.replaceg(pat, rep)	Returns str with all matches of pat replaced by rep.
s = str.rotate (n,pre,min)	Rotates string n chars if it is longer than min.
n = string21strcoll(string2)	Compares two strings using local collating sequence; returns -1 if first less than second, 0 if first equal second, 1 otherwise
n = string1.stricoll(string2)	Compares two strings using local collating sequence, ignoring case of letters; returns -1 if first less than second, 0 if first equal second, 1 otherwise
s = string.select(n1, n2)	Selects characters from string.
s = string.squeeze(s2)	Replace runs of characters in s2 from string
s = s1.translate(to,from)	Replaces s1 chars which are in to by corresponding from characters
s = s1.trim(out,n)	Removes characters in out from s1. You can omit out and n (default blank and 2).
n = string.revindex(string2)	Index of last occurrence of string2 within first See also index.
x = string.stringtofloat	Convert string to floating point.
x = string.tofloat	Convert string to floating point.
s = string.word(n, delims)	Scans the string and returns the word given by n; words are separated by any character in delims; if delims omitted, uses blank or tab as separato
n = string.word(0, delims)	Returns the number of words in the string; words are separated by any character in delims; if delims omitted, uses blank or tab as separator

n = string.validpath	Returns 1 if string is valid path to file; 0 else.

Note that you can also use the math functions in dot format with floating point variables, as in x.sqrt.

Expressions and Functions

formatdate() and formattime()

two of Powerpro's built in Functions

formatdate

```
s = formatdate ("format string","yyyymmdd")
```

Returns the input date, converted to the specified format.

parameters

"yyyymmdd" is the 8 digit date to be reformatted.

"format string" can be longdate *or* shortdate, to select formats set on Control Panel Regional Settings. Or it can be a combination of the following:

d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month as digits with leading zero for single-digit days.
ddd	Day of week as a three-letter abbreviation.
dddd	Day of week as its full name.
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation.
MMMM	Month as its full name.
yy	Year as last two digits, but with leading zero for years less than 10.
yyyy	Year represented by full four digits.

Case of letters is important. You can use literal characters like / or : in the string.

Note: You can also manipulate dates using the date plugin

examples

```
rsDate = formatdate( "dddd, dd MMMM yyyy", "20040822")
```

returns: Thursday, 22 August 2004

```
rsDate = formatdate( "dd/M/yy", "20040822")
```

returns: 22/6/04

```
RawDate = "20020822"
rsDate = formatdate(longdate, RawDate)
```

returns the input date in the default long date format.

formattime

```
s = formattime("format string","hhmmss")
```

Returns the input time, converted to the specified format.

parameters

"hhmmss" is the time using 24 hour clock

"format string" can be any combination of:

h	Hours with no leading zero for single-digit hours; 12-hour clock

hh	Hours with leading zero for single-digit hours; 12-hour clock
H	Hours with no leading zero for single-digit hours; 24-hour clock
HH	Hours with leading zero for single-digit hours; 24-hour clock
m	Minutes with no leading zero for single-digit minutes
mm	Minutes with leading zero for single-digit minutes
s	Seconds with no leading zero for single-digit seconds
ss	Seconds with leading zero for single-digit seconds
t	One character time marker string, such as A or P
tt	Multicharacter time marker string, such as AM or PM (depending on your regional settings)

Case of letters is important. You can use literal characters like : in the string.

example

rsTime = formattime("h:mm:ss tt","140102") returns "2:01:02 PM"

Expressions and Functions

Input dialogs

This page provides full descriptions of the functions which show input dialogs.

Note: In all input dialogs, if Cancel is pressed flag 0 is cleared. Otherwise flag 0 is set.

PowerPro's built-in input dialogs are quite limited; the dialog plugin provides much more capability.

pickfile

s = pickfile(paths,title)

Returns a full file path from a dialog showing filenames only.

Pickfile displays a dialog with title "title" and an edit box and a list box. The list box is filled with the names of all files in the specified paths (only the file name and type are shown, not the full path). Typing blank-separated strings in the edit box reduces the names in the list box to only those which contain what is typed in the edit box. To select a file, double click on it, click on it and press enter, or press enter to select the top file in the list.

Example

do(pickfile(file.listfiles("c:/mymp3", 1),"Play song"))

displays a list of files in c:/mymp3 and subfolders and plays the selected file.

The index of the picked string is always assigned to the global variable _pickedline_ ; indexes start at 1, as they do for the line function.

pickstring

s = pickstring(lines,title)

Selects a string from a list in a dialog.

Pickstring displays a dialog with title "title" and an edit box and a list box. The list box is filled with the specified strings, one per line. Typing blank-separated strings in the edit box reduces the strings in the list box to only those which contain what is typed in the edit box. To select a string, double click on it, click on it and press enter, or press enter to select the top string in the list.

Example: pickstring("alpha\rbeta\rgamma\rdelta","greek to me") displays the four strings, which are supplied literally in that example. Or these examples use an expression to supply the strings:

pickstring(allglobals,"pick a global variable")

pickstring(alldesknames,"choose a desktop")

pickstring(file.readall("c:/store/quoteslist.txt"),"choose a quotation")

The index of the picked string is always assigned to the global variable _pickedline_ ; indexes start at 1, as they do for the line function.

You can select multiple strings with

s = pickstring(lines,title,1)

The returned string will have multiple lines, one per picked string. The global _pickedline_ will consist of the blank separated indexes of the picked lines. If enter or OK is pressed without selecting any strings, all visible strings are returned.

You can turn off sorting with

s = pickstring(lines,title,2)

You can turn off sorting and pick multiple lines with

s = pickstring(lines,title,3)

filemenu

s = filemenu(filepath) Displays a file as a menu; the selected item is the result.

setnextdialogpos(left,top,width,height,topmost)

Sets position for next inputdialog, pickstring, pickline, inputdefault, input(title), inputcancel dialog. All 5 parameters must be specified; use "" as a parameter to get the default behavior. The first four parameters specify position and size. If the final parameter includes the string "top", the dialog is shown as topmost.

inputtext

s = inputtext Prompts for input text. Same as input(title) but you cannot set title

input(title)

s = input(title) Prompts for input using title for the input dialog's title. To limit input length, start title with =n, e.g. input "=15Enter up to 15 chars". Result if whatever input is entered. If Cancel pressed, result is ""

inputcancel

s = inputcancel(title) Same as input(title), except that if Cancelled the running script quits

inputcolor

n = inputcolor Displays color dialog and returns results as single number

inputdate

n = inputdate Displays calendar calculator dialog and returns selected date as 8 digits yyymmdd

inputdatetime

n = inputdatetime Displays calendar calculator dialog and returns date and time as 14 digits yyymmddhhmmss

inputdefault

s = inputdefault("default","title") Prompts for input string; preset with default value

Use: InputDefault("default", "title") Prompts for input using title for input dialog and "default" for default value. To limit input length, start title with =n, e.g. input "=15Enter up to 15 chars". Result if whatever input is entered. If Cancel pressed, result is "".. You can use auto completionwith the input by putting the auto complete keywords as an optional third parameter.

inputdialog

[b=] inputdialog("var1=title1, var2=title2, [...var6=title6], "dialogtitle")

The InputDialog function displays a dialog with up to six edit boxes, check boxes, or combo boxes to set the values of up to six variables.

In the dialog, the user can also set each variable to a folder name or a file name.

Result is 1 unless cancel is pressed in which case it is 0.

You can use up to six var=title strings, separated by commas. For example,

Inputdialog("x1=Enter new stuff, beta=Value here","Two Variables")

displays a dialog with caption "Two Variables" and two edit boxes. The first box has title "Enter new stuff"; it is initialized to the variable x1, and is used to reset variable x1 to a new value when OK is pressed. The second box has title "Value here" and sets variable beta.

To display a password edit box which echoes asterisks for characters typed, use var==title.

To display a check box rather than an edit box, add two question marks (??) to the end of the title; for example: var2=checkbox label??

To display a combo box rather than an edit box, add two question marks (??) to the end of the title and follow the question marks by the list of combo box items separated by bars; that is use: var=combo title??item1|item2|third item

The user can optionally set each variable to a folder name or a file name. The dialog contains File and Folder buttons to retrieve files and folders. The retrieved names are put in the last edit box which had the keyboard focus before the button is pressed.

To hide the File and Folder buttons at the bottom of the dialog, put a ~ at the start of the var=title string, eg Inputdialog("~x1=Enter no file ","One Variable")

If you need to use a comma, bar, equal sign or two questions marks and want to avoid their special meaning, you can change the character by putting oldchar=newchar at the start of the first string and repeating without spaces for each character to be changed. For example, the following uses a slash instead of a comma and a star instead of an equal sign.

inputdialog"(",=/==*var1*title1/var2*title2 ", "caption for dialog")

inputfolder

s = inputfolder Displays folder browse dialog and returns results.

inputpath

s = inputpath Displays file browse dialog and returns results.

inputpathmulti

s = inputpathmulti Displays file browse dialog and returns results, allowing multiple files to be selected. Clears flag 0 if cancel pressed; sets it otherwise. The returned result consists of a string with many lines of text. The first line is the path to the folder containing the selected files. The remaining lines are each a file name. Usage

To process, use the following type of script

```
multi = inputpathmulti
if (pproflag(0)) do
    path = line(multi,1)
    for (i=2;1;i=i+1)
        file = line(multi,i)
        if (not file)
            break
        ; process path ++ "\" ++ file
    endfor
endif
```

other dialogs

[n=] messagebox ("layout", "text", "title")

Shows a message box. See MessageBox function for details.

*Exec Prompt flag/var text   Shows a Yes/No/Cancel dialog. Prompting for Yes/No

Expressions and Functions

MessageBox function

The messagebox function is a more flexible alternative to the *Message command.

It shows a message box with up to three buttons and returns a result indicating which button was pressed. The form is:

messagebox ("layout", "text", "title")

where "text" is the text for the messagebox, "layout" contains keywords specifying the buttons and icons for message box, and "title" is the title for the messagebox (this third argument is optional).

To set the buttons, the "layout" can contain any of: abortretryignore, okcancel, retrycancel, yesnocancel, yesno, ok

To set the icons, the "layout" can contain: exclamation, warning, information, asterisk, stop, error, question

return value

The result depends on the button pressed.

Cancel=0, OK=1, Abort=3, Retry=4, Ignore=5, Yes=6, No=7.

example

messagebox("yesno question" , "continue with script?")

displays a Yes/No messagebox with a question mark icon.

Expressions and Functions

String functions and operators

PowerPro has functions and operators to help you work with strings of characters.

PowerPro supports strings of any length.

case

s = case(keyword,string)   Changes case of letters in string. The first keyword string tells how to change the case of the second string.

case ("lower", "AbcDEf") yields "abcdef";

case ("upper", "AbcDEf") yields "ABCDEF";

case ("title", "each WORD capital") yields "Each Word Capital;

case ("sentence", "Just the firST") yields "Just the first".

case ("tonum", "abc") translates the characters in the string to space-separated decimal equivalents.

case ("tonumx", "abc") translates the characters in the string to space-separated hexadecimal equivalents.

case ("fromnum", "97 98 99") translates space-separate decimal numbers to corresponding ASCII character

case ("fromnumx", "61 62 63") translates space-separate hexadecimal numbers to corresponding ASCII character

The keyword "acronym" creates an acronym from the first character of each word:

timezone = "Easter Standard Time"
case ("acronym", timezone)

would yield "EST".

env

s = env(string) Value of environment variable named by string. Also:

*Exec RefreshEnvironment Refreshes all env variables from registry (NT and later only).

*Exec SetEnv envarname value Set environment variable to provided text.

esc

s = esc(?+string+,?+c+) Applies escape character processing based on c to string. That is, the characters of string are processed as if c is the escape character. For example esc(?+a_line$n+,?+$+) replaces the $n by a newline.

eval

s = eval(string) Evaluates string as an expression and returns results. Eval allows you to build expressions as strings and then evaluate them

fill

s = fill(string1,string2) Combines two strings. Used to pad a number or string using a padding string. Creates a string of length no less than the first string with the ending characters set by the second number or string.

Example: fill ("0000",12) yields "0012"

Example: fill("***///", "a") yields "***//a".

index and revindex

n = index(string1,string2) Index of second string within first. Used to find one string in another, ignoring case of letters. Returns 0 if the second string is not found. If it is found starting at the nth position, index returns n.

Example: index("to be or not to be", "be") returns 4 (index of first "be").

n = revindex(string1,string2) Index of last occurrence of string2 within first Used to find last occurrence one string in another, ignoring case of letters. Returns 0 if the second string is not found. If it is last found starting at the nth position, revindex returns n.

Example: revindex("c:/path/to/filname.txt", "/") returns 11 (index of last slash).

join and length

s = join(string1,string2) Joins two strings. join ("abc", "defg") is "abcdefg"

Same as "abc" ++ "defg"

n = length(string) Compute length of string; length("abc") is 3

line

s = line(string,n) Scans the string and returns the nth line. Lines start at 1.

n = line(string,0) Returns the number of lines in the string.

min and max

s = min(val1,val2) Minimum of two values eg: min("abcd", vstring)

s = max(val1,val2) Maximum of two values eg: max(var1, 7)

pickstring

s = PickString(lines,title) Shows an input dialog.

remove

s = remove(string,number) Removes n characters from string. remove ("abcd", 2) is "cd". Use a negative number of remove characters from the end of a string. remove("abcd", -3) is "a"

repeat

s = repeat(string,n) Returns string formed by repeating first argument n times

rotate

s = rotate(str, n, pre, min)

Rotates str n characters to the left. If rotation greater than 0, prefixes the start of the string by pre. If string length is less than or equal to min, then no rotation is done. The last two parameters are optional.

replacechars

s = replacechars(string1,string2) Two parameters form.

Replaces single character[s] by specified common replacement character. Characters in the first string are replaced according to the second string. The second string must be at least two characters for replacement to occur. The final character in the second string is the replacing character. All occurrences of any other characters in the second string which occur in the first string are replaced by this final character.

Example: replacechars ("xx.x.abc=*", ".=_")
produces: "xx_x_abc_*". The _ has replaced all occurrences of . or =.

s = replacechars("string", "from", "to") Three parameters form.

Replaces all occurrences of "from" in "string" with "to" and returns the result.

select

s = select(string,number) Selects n characters from the start, end, or middle of a string.

If number is positive, characters are selected from the start:
select ("abcde", 3) is "abc".

If number is negative, characters are selected from the end:
select ("abcde" ,-1) is "e'.

To select characters from the middle of a string, use three operands for select.
For example, to select the 2nd through 4th character, use:
select ("abcde" , 2,4) which yields "bcd".

squeeze

s = squeeze(str1, str2) For characters in str2, replaces runs of two or more repeated characters in str1 by a single character. For example, squeeze("   a   ***b", " *") yields " a *b" . If the str2 is omitted, replaces runs of two or more of any character in str1.

strcoll and stricoll

n = strcoll( ) Compares two strings using local collating sequence. Returns -1 if first less than second, 0 if first equal second, returns 1 otherwise. stricoll ignores case.

translate

s = translate(string, from, to) Each character in string is checked. If the character occurs in the from string, it is replaced by the corresponding character in the to string. If it does not occur in the from string, it is left unchanged. For example

translate("alphabeta", "abc", "xyz") yields "xlphxyetx"

For example.

s1=translate(anystring, "abcdefghijklmnopqstuvwxyz","nopqstuvwxyzabcdefghijklm") s2=translate(s1,"ABCDEFGHIJKLMNOPQSTUVWXYZ","NOPQSTUVWXYZABCDEFGHIJKLM")

sets s2 to ROT13 encoding of any anystring.

trim

s = trim(s1,out,n) - Removes characters in out from start or end of s1. If n is 1, removes from start. If 2, removes from end. If 3, removes from start and end. For example,

trim(string,"0123456789", 1)

removes digits from start.

trim (string, " ", 3)

removes blanks from both ends of string)

word and nextword

s = word(string, n, delims)	Scans the string and returns the word given by n; words are separated by any character in delims; if delims omitted, uses blank or tab as separator
n = word(string, 0, delims)	Returns the number of words in the string; words are separated by any character in delims; if delims omitted, uses blank or tab as separator

s = nextword(string, "var", delims) Returns the next word as delimited by characters in delims (defaults to blank and tab if omitted). The contents of the variable named var are replaced by all the characters after this first word. For example, nextword("abc*de*efgh","rest","*")

Expressions and Functions

MCI function

about MCI

MCI provides you with device-independent capabilities for controlling audio and visual peripherals. You can use PowerPro mci to control any supported multimedia device, including waveform-audio devices, MIDI sequencers, CD audio devices, and digital-video (video playback) devices.

For more general information on mci, see the Microsoft site

http://msdn.microsoft.com/library/en-us/multimed/mci_04dv.asp?frame=true

or search for mci command strings at microsoft.com.

PowerPro mci is normally used with the cdaudio device, which will be used in all following examples. You can also use any of the other devices listed at the Microsoft site; in particular, you may be able to use cdaudio1 for a second audio CD.

using mci commands

Either: myvar = mci ("command cdaudio parameters")

where command is one of the commands listed at Microsoft and parameters are described in the MS documentation of the MCI command. The variable myvar is assigned the result of the mci command.

You can also use an *Info button label to view the results of an mci command, e.g.

*info expr (mci "status cdaudio mode")

However, a warning about using *info is that this will poll the device once per second which means the CD will be constantly whirring.

examples of mci usage

```
mcistatus = mci ("status cdaudio mode")
```

will result in "stopped" if the CD has stopped playing.)

```
mci ("set cdaudio time format tmsf")
```

sets cdaudio so that subsequent positions (e.g. in play) refer to tracks.

```
mci ("play cdaudio from 1 to 2")
```

plays the first two tracks (assuming time format tmsf).

```
mci ("play cdaudio from " join x0 join " to " join x1)
```

plays tracks x0 to x1.

```
mci ("set cdaudio door open")
```

opens the door.

```
mcitrack = mci ("status cdaudio current track")
```

returns the current track.

Expressions and Functions

Plugin usage

how to use plugin calls

see also

How to program plugins is described in the next topic.

distributed plugins

There are several plugins written by PowerPro's author. They are placed in the plugins folder during installation. See the following sections for documentation. The plugins include:

file	- read or write text files one line at a time; read or write an entire file to/form a string; take apart the components of a file name; work with file dates; work with all the files in a folder; run a file and wait for the program to end
win	- The win plugin retrieves information about a window or its child windows. The plugin can also manipulate windows: minimize, maximize, close, and so on. The plugin can send and post window messages. The plugin can play system sounds and also activate the low-level beep. The plugin has services to access the current language, desktop name, and other system metrics. Finally, the plugin provides access to keys and debug functions of PowerPro.

event – working with events

float- working with floating point numbers

int64 - work with 64 bit integers

clipclipboard manipulation with support for multi-line strings

vec - work with vectors

map - work with hash tables

date- add and subtract dates

regex - work with regular expressions

note- programmatic access to text and appearance of PowerPro notes

additional plugins

PowerPro also allows other programmers to add new features by writing additional plug-ins. For example, plugins have been written to work with the registry, with ini files, and with com objects. Many useful plugins are available for download.

how to use plugins

A plug-in is a .dll file that you must place in the same folder as your PowerPro .exe file or in the subfolder \plugins\ (usually c:\program files\powerpro\plugins).

Each plugin offers one or more services.

To use a plugin service, you call it with this syntax: dllname.service(arg1,arg2,arg3,...) The arguments (also known as parameters) are separated by commas. The case of the dllname and the service are ignored.

Different services require a different number of arguments when they are called.

If the plugin service requires no arguments, you can call the service by writing either dllname.service() or dllname.service.

Each of the arguments is supplied by using any of the usual expression forms:

a built-in Function such as: timesec or length(string)

or plain text in quotes: "c:\docs\myfile.txt"

or a whole number in digits: 543

or another plugin call: win.height("calculator")

or a complex expression such as: "c:\" ++ myfoldervariable ++ "\myfile.txt"
which will be evaluated before being sent to the plugin.

You can use plugin calls in two ways:

wherever an expression is expected, such as in the parameter of a command.

calls to plugin services may be used as a stand-alone statement, acting as a command.

Some plug-ins, like file, win, note, event, create handles to allow you to work with specific files, windows, notes, events. A shorter syntax can be used in plug-in calls with handles. If han is variable containing such an handle, then you can use

han.service(params)

as a short form for

dllname.service(han, params)

The plug-in documentation will describe when you can use this syntax.

examples of plugin calls used as commands

When used as a command, the plugin call appears as the first element of a script line, or in the *Command* text box in a command editing dialog.

file.close(handle1)

Calls the **file** plugin, service **close**, with argument **handle1**.

File.CloseAll

Calls the **file** plugin, service **closeall**

(The **file.closeall** service requires no arguments. Notice the case of the plugin name and service name will be ignored)

examples of plugin calls used in expressions

• they can be used as the condition in an IF() statement:

```
if(file.isfolder("c:/tempfolder")) do
```

checks whether c:\tempfolder exists.

The following (at the start of a line) is a legal command:

```
file.isfolder("c:/tempfolder")
```

but since the only function of this particular call is to return a result depending on whether the plugin's argument "c:/tempfolder" is a folder, and since this result is not stored or used, this plugin call would have no effect when used as a command.

• they can be used in the value assigned to a variable:

```
result = float.add(vara,varb,varc)
```

calls the **add** service of the **float** plugin, with three arguments supplied in user variables.

```
days = date.diff(date.today(), test_date) + 5
```

sets the user's variable `days` to the result of calling the **date** plugin with service **diff**. The first argument calls the **date** plugin again, with service **today**.

• they can be used in the parameter of a command:

```
file.copy(win.exepath("mystery window"), c:\investigate)
```

Note that the plugin call `file.copy` is used as a command here and the call `win.exepath` is used as an argument.

unloading plugins

If you wish, you can unload a plugin from memory when you are finished with them by calling its unload service, e.g. `date.unload`. PowerPro will call the unload service if it exists, then unload the plugin. Only do this if you do not plan to use the plugin again since it takes time to reload a plugin.

loading plugins

If you wish, you can load a plugin into memory with plugin.load, e.g. `date.load`. This will load the plugin into memory and run a load service, if it exists (no error if load service does not exist). This allows you to control when the delay of initially loading a plugin occurs. Use of the load service is optional. If you do not use it, the plugin is loaded when it is first used.

Expressions and Functions

Date Plugin

date plugin overview

The Date plugin lets you work with dates: adding days to a date, finding the number of days between two dates, finding the weekday of a date, finding the day in the year of a date, and finding the week number of a date.

Dates processed by the date plugin follow the same format as the dates produced by PowerPro keywords and functions like date and formatdate. Dates are represented as eight digit number yyyymmdd, such as 20021028 for October 28, 2002 or 20030101 for January 1, 2003.

The Date plugin also accepts 14 digit values which represent an 8 digit date followed by a six digit time, using a 24-hour clock. If you provide 14 digit date to addDays or addSeconds, then a 14 digit date is returned.

As well, if you use a 6 digit value, the plugin takes this as a time only and returns a six digit time for addDays and addSeconds.

list of services

Date.Today	todays date, same as PowerPro keyword date
Date.Now	todays date and time, as 14 digit string
Date.AddDays	adds a given number of days to a date
Date.AddSeconds	adds a given number of seconds to a date
Date.Sub	subtracts two dates, given the number of days between them
Date.SubSeconds	subtracts two dates, given the number of seconds between them
Date.WeekDay	gives day of week for given date, 0=Sunday, ..., 6=Saturday
Date.YearDay	gives day number in year of date
Date.WeekNum	gives week number in year of date
Date.Get	select a date from a calendar

description of services

In the following descriptions, d1, d2 and d3 represent eight or fourteen digit dates, and n1 is a number of days or seconds. In all cases, n1 may be positive or negative.

date.today	Returns todays date; same as internal keyword date.
d2 = Date.AddDays(d1,n1)	Adds n1 days to d1, returning resulting date. d2 = Date.AddDays(20021031, 2) sets d2 to 20021102. d2 = Date.AddDays (20021031, 365) sets d2 to 20011031)
d2 = Date.AddSeconds(d1, n1)	Adds n1 days to d1, returning resulting date.
d3 = Date.Sub(d1, d2)	Gives the number of days between d1 and d2. d3 = Date.Sub(20021102, 20021031) sets d3 to 2.
d3 = Date.SubSeconds(d1, d2)	Gives the number of seconds between d1 and d2.

n=Date.weekday(d1)	Gets the day of the week for a the date; Sunday is day 0. n= Date.WeekDay(20021029) sets d1 to 2 (Tuesday)
n=Date.YearDay(d1)	Gets the day in the year for a date. d1 = Date.YearDay(20021029) sets d1 to 302. d1 = Date.YearDay(20021231) sets d1 to 365
n=Date.Weeknum	Gets the week number in the year. The week number definition follows the ISO standard: Week 1 of any year is the week that contains 4 January, or equivalently Week 1 of any year is the week that contains the first Thursday in January. d1 = Date.WeekNum(20021029) sets d1 to 44
d1 = Date.Get	Displays a calendar and returns selected date. Returns 0 if Cancel pressed.

Expressions and Functions

Event Plugin

event plugin overview

The event plugin lets you schedule commands to execute repeatedly at a specified interval. You can optionally specify the number of times the event should repeat. You can optionally specify a test expression; if specified, the plugin evaluates this test expression each interval and only executes the command if the expression is not 0 or "".

The event plugin is a more flexible alternative to the wait command and to timers.

The event plugin can support at most 50 simultaneously active events.

Only global variables can be used in the expression and commands used in an event commands or test expressions, since the command or tests can execute regardless of which scripts, if any, are active; you cannot use local or static variables. The cb function can be useful to create callbacks involving local or static variables from the script which creates an event since it evaluates the variables arguments immediately and creates constants for the call backs.

list of services

event.create	Creates a new event and returns its handle.
event.createms	Creates a new event based on an interval in milliseconds and returns its handle.
event.message	Creates a count down message box
event.destroy	Removes the specified event.
event.destroyall	Removes all active events.
event.destroythis	When used in an event's associated command, removes the event.
event.exists	Returns 1 if event exists; 0 otherwise.
event.this	When used in an event's associated command, returns the event\s handle.
event.count	Returns remaining number of times event will execute.
event.countthis	When used in an event's associated command, returns remaining number of

	times event will execute.
event.remaining	Returns remaining number of times command of event will execute (using maxtimes argument)
event.remainingthis	When used in an event's associated command, returns remaining number of times command will execute.
event.ran	Returns remaining number of times command of event has executed (including any currently underway execution).
event.ranthis	When used in an event's associated command, returns number of times this event has executed.

description of services

In the following descriptions, e is an event, s is an interval in seconds, m is an interval in milliseconds, c is an integer count, cmd is a string containing a PowerPro command or file to be run, and expr is a PowerPro expression.

For example, the following creates an event which shows the event count if notepad is active.

```
event.create(1,100,?+win.debug(event.countthis)+, ?+visiblewindow("*notepad*")+)
```

Note that the ? form of string is used to avoid having to precede double quotes in the commands with escape character (backslash).

You can use multiple commands in the command by separating each command with \r. For example,

```
event.create(3,0,"win.debug(var)\r if (shift)\r event.destroythis")
```

creates an event which shows var1 in a debug window until shift is pressed upon which the event is removed.

e=event.create(s,c,cmd, expr, maxtimes)

Creates an event which repeats every s seconds. Returns a handle greater than zero which can be used to refer to the event. If c is greater than zero, after c occurrences, the event is removed If c is zero, event continues until removed by an event.destroy. The string cmd specifies a command to be executed after each interval. The expression expr may be omitted, in which case the cmd is always executed. If the expr is specified, than it is evaluated after each interval and the cmd is only executed if the result is not 0 and not "". If you specify and count and an expression, the count _is_ decremented whether or not the command is executed; this way, the count can act as a watchdog to end the event in case the expression is never true. Use maxtimes to control the number of times a command will execute when an expression is specified: if maxtimes is greater than 0, then the cmd is executed at most maxtimes.

For example,

```
e=event.create(3,0,"win.debug(testvar)")
```

creates an event which shows the variable testvar in a debug window once every 3 seconds. The event continues indefinitely.

For example.

```
e=event.create(2,10,"keys abc\r event.destroythis", \+
?=visiblewindow("*notepad*")=)
```

creates an event which checks for a visible notepad window, sends key abc to it, and then ends. The event continues until notepad appears or 10 seconds elapse. Note how \+ is used to extend the command over two lines (for scripts in files).

```
e=event.create(2,10,"keys abc",?=visiblewindow("*notepad*")=,
```

1)

Also creates an event which checks for a visible notepad window, sends key abc to it, and then ends. The maxtimes argument is used to destroy the event after one command execution.

e=event.createms(m,c,cmd,expr, maxtimes)	Same as create, except that the interval is specified in milliseconds. The minimum is 20 milliseconds.
e=event.message(count,cmd,message,title,ontop,left, top)	Creates a message box with given title and message which counts down starting from count, once per second. If zero is reached or the Start Now button is pressed, the cmd is executed. The global variable _exit_ is set 0 if Start Now pressed, 1 otherwise. If ontop is "1", the message box is set topmost. If both left and top and present, they give the screen coordinates for the message box; otherwise, it is centered on the screen.
e=e.destroy	Removes event e and returns 0 which can be assigned to a variable to set it to an invalid event handle.
e=event.destroythis	Must be used in the command assigned to an event when it is created. If executed, removes the command. Returns 0 to assign to any handle of the event.
event.destroyall	Removes all active events.
event.this	Must be used in the command assigned to an event when it is created. Returns the event handle. create(1,20,".myscript(event.this)",visiblewindow(\"=winword\")") creates an event which calls script file myscript with argument equal to event handle when MS word has a visible window. Event will run at most 20 seconds.
e.exists	Returns 1 if e is a valid event; 0 otherwise.
e.count	Returns the number of intervals remaining for an event scheduled with a count.
event.countthis	Must be used in the command assigned to an event when it is created. Returns the number of intervals remaining for the event. create(1,10,"win.debug(event.countthis)") puts 1, ..., 10 in the debug window.
e.remaining	Returns the number of command executions remaining based on the maxtimes argument. Will be zero if this argument is not specified. The remaining times exclude any currently executing event.

event.remainingthis	Must be used in the command assigned to an event when it is created. Returns the number of command executions remaining for the event based on the maxtimes argument. Will be zero if this argument is not specified. The remaining times exclude the currently executing event.
e.ran	Returns the number of command executions for the event, including the current one, if an execution is underway.
event.ranthis	Must be used in the command assigned to an event when it is created. Returns the number of command executions for the event, including the current one.

Expressions and Functions

File Plugin

file plugin overview

The file plugin let you work with files. There are services for:

- reading and wrting files line by line
- reading and writing entire files
- deleting, moving, copying filesworking with all files in a folder
- checking for existence of files and folders
- extracting name, type, folder, name.type from full file path
- getting file sizeworking with shortcuts (.lnk files)
- running programs and waiting until they are done
- have a command executed if a folder contents change

Some of the services provided by the plugin allow you to open files, read or write to them, and then close files. This can be more efficient than the built in PowerPro functions readline and *Exec ToFile since the built-in routines close and re-open the file with each operation. Other services let you read or write entire files to/from a variable.

list of services

open	opens a file and returns a file handle used to access the file line-by-line
close	closes the file and frees the file handle
closeall	closes all open files
restart	restarts reading or writing of a file from the beginning
readline	reads and returns next line from file including any trailing \r or \n
readstring	reads and returns next line from file without trailing \r and \n
eof	returns 0 if last read (**not** next read) did not encounter eof; non zero otherwise; always use immediately after readline or readstring
writeline	writes a string and following \r and \n

writestring	writes the string but does not output a \r or \n
setmaxline	set maximum line length for subsequent calls to file.readline
delete	deletes a file; the file name can contain wildcards
deletenorecycle	deletes without sending file to recycle bin
copy	copies a file to a destination file or folder
move	moves (same as renames) a file to a destination file or folder
validpath	checks to see if argument is a valid file or folder path
isfolder	checks to see if argument is a valid path to a folder (directory)
attrib	returns file attributes as string
allfiles	counts or runs a command on all files in a folder (and optionally its subfolders)
listfiles	returns a list of file names from a single folder in a multi line string
readall	reads entire file into a single string
writeall	writes entire file or appends to file a single string
type	returns the type (extension) of a file path excluding the leading period
name	returns the file name of a file path, excluding the folder and type
nametype	returns the name and type from a full file path
folder	returns the folder from a file path without the final \
size	returns file size in bytes
size64	returns size as 64 bit integer; use int64 plugin to work with
ksize	returns file size in kilobytes (size/1024)
lastmodified	returns the last modified date and time of the file as a string of 12 digits: yyyymmddhhmmss.
setdate	sets the modified or created date of a file
getdate	gets the modified or created data of a file
version	returns file version info as four blank separated numbers
volume	returns volume name
alldrives	returns multi line string with line for each valid drive
createshortcut	creates a .lnk file shortcut to another file
getshortpath	returns short path to specified file
resolve	resolves shortcut and returns specified information
runwait	run exe and wait until it terminates
runcallback	run exe and execute PowerPro command when it terminates
runas	run file under another userid. Only for W2000 and later.
runaswait	run file under another userid and wait for it to end. Only for W2000 and later.
doverb	execute file action or open file properties dialog
watchfolder	run a command when a folder's contents change

services for accessing files line-by-line

The following standard variable names are used:

- fh is used to identify the File Handle: a value assigned by the plugin service file.open when a file is opened.
- str is used to identify a string or a variable holding a string.

The plugin can handle long lines of text. You specify the maximum line length with file.setmaxline; the default starting maximum line length is 4K characters.

The plugin allows a maximum of 15 files to be open simultaneously.

fh = File.Open(str1, str2)	The Open service opens the file with path given in str1. If present, the second string, str2, must be one of "r" opens the file to be read "w" opens the file to be written; will be overwritten if it exists "a" opens the file to have new information written after existing information If omitted, "r" is assumed. The variable fh is set to a positive integer which must be saved and used in subsequent file operations. If an error is encountered while opening a file, fh is set to 0 or a negative integer. Once opened, the file handle fh may be used in the shortform syntax fh.service(params).
fh.Close	Closes the file given by fh.
file.CloseAll	Closes all open files.
fh.Restart	Next operation will occur at start of file.
str = fh.Readline	Reads the next line of the file and returns it to be assigned to str. The maximum line length starts at 4K characters but can be set as large as 64K characters with file.setmaxline.
str = fh.ReadString	Reads the next line of the file and returns it to be assigned to str. The maximum line length starts at 4K characters but can be set as large as 64K characters with file.setmaxline. The trailing \r and \n are removed.
File.SetMaxLine(n)	Sets maximum line length for subsequent file.readline calls. If n <264, 264 is used. If n>64K, 64K is used.
fh.Eof	Returns 0 if last read did not encounter eof; non zero otherwise. The standard way to read a file is

```
fh = File.Open(strPath, "r")
if (fh > 0) Do
   for (lineNum=1;1;LineNum=LineNum+1)
```

```
    str = fh.Readline
    if (fh.Eof)
      break
    ;  process the file line in variable str
  endfor
else
  MessageBox ("ok", "Error opening file "++strPath)
endif
```

Note that you must check for fh.eof immediately after reading the line. If you check before, you will process an extra blank link at eof.

fh.WriteLine(str)	Writes the line given in str to the file, adding a \r\n to make a complete line. If the str already ends in a \r or \n, none is added. You can also use File.WriteLine(fh, "") to write just a \r\n and terminate any lines partially written with File.WriteString.
fh.WriteString(str)	Writes the string str to the file with no terminating \r\n. Subsequents WriteLine or WriteString will add text to same line. Use file.writeline(fh,"") or file.WriteString(fh,"\n") to terminate the line.

example of file input/output

The following script prompts for a file path, then copies the file at that path to c:\textout.txt. Each line in the output is prepended by three asterisks (***).

```
fp = file.open(inputpath,"r")
if (fp>0) do
  fpout = file.open("c:/textout.txt", "w")
  if (fpout>0) Do
    for (1)
      line = fp.readLine
      if (fp.eof)
        break
      fpout.writeString("***")
      fpout.writeLine(line)
    endfor
    fpout.close

  else
      win.debug("cannot open output file")

  endif
fp.close
else
```

```
   win.debug("cannot open input file")
endif
```

other file services

file.DELETE(filepath)	Deletes the indicated file. The filename can contain wildcards.
file.DELETENORECYLCE(filepath)	Deletes the indicated file and does not send it to the recycle bin. The filename can contain wildcards.
res=file.COPY(filepath1, filepath2, noerr)	Copies the first file to the second. The second path can be either a folder name or a full file name. If third argument is present and starts with digit "1", no error messages will be produced. res is 1 if success, zero otherwise.
res=file.MOVE(filepath1, filepath2, noerr)	Moves the first file to the second; same as rename. The second path can be either a folder name or a full file name. If third argument is present and starts with digit 1, no error messages will be produced. res is 1 if success, 0 otherwise.
file.VALIDPATH(filepath)	Returns 1 if filepath1 is a valid path to a file or folder; 0 otherwise. The file name may contain wildcards.
file.ISFOLDER(path)	Returns 1 path is a valid path to a folder; 0 otherwise. The path may _not_ contain wildcards.
file.ATTRIB(path)	Returns a string of single letters indicating the path's attributes. The string consists of zero or more of the following letters: a for archivable, c for compressed, d for directory, h for hidden, o for offline, r for read only, s for system, t for temporary. For example, "ahr" indicates a hidden, read only, archivable file. Use the index function to check for a particular attribute, eg index(file.attr(path), "h")>0 checks for hidden file.
file.ALLFILES(path, command, subfoldersflag, pumpflug, hidden)	Cycles through all files in filepath and runs the command for each one. The variable _file_ is set to each file name and can be used in the command, Or the command may contain the character |; in this case, the | is replaced by the file being processed each time the command is run. Returns a count of all files processed. The subfolders flag can be: 0: run on files in path only 1: run on files in path and all of its subfolders 2: run on files and folders in path only 3: run on files and folders in path and all of its subfolders (use file.isfolder to select folders) 4: run on folder only in path

5: run on folders only in path and all of its subfolders

Pumpflag is used to keep PowerPro responsive while files are being processed; if present and greater than 0, then every n files, PowerPro checks for user input to processes (e.g. through a bar), where n is the value of the pumpflag parameter.

If the hidden parameter is 1, hidden files are included.

The command may be omitted or set to "", in which case AllFiles simply returns a count of all files in the path.

The path may contain wildcards or it may simply be a path to a folder in which case all files in the folder are processed. However, if the path contains wildcards, then subfolders will not be processed. Examples:

File.AllFiles("c:/windows","","1")

counts all files under "c:/windows".

File.AllFiles("c:/mypath/*.tmp", "file.delete(_file_)")

deletes all .tmp files in c:/mypath but not its subfolders.

File.AllFiles("c:/mypath", "win.debug(_file_)")

lists all files in c:/mypath but not its subfolders.

Do not use ALLFILES to rename files as the renamed files may be reprocessed after renaming.

File.ListFiles(path, subfoldersflag, pumpflag)	Returns a list of all the files names in the folder given by path; the path may include a file name with wildcards or the path may not have a file name in which case all files in the path's folder are listed. Each entry in the list has the full file path. The names in the list are each on a separate line so that PowerPro line function can be used to cycle through each one or vec.createfromlines can create a vector of results. The list can also be used as an argument to the pickstring or pickfile function to display the files and choose one. The remaining arguments service the same purpose as for the file.allfiles service. To select and play a file, use: do(pickfile(file.listfiles("C:/MP3",1),"play"),"") The path may contain wildcards or it may simply be a path to a folder in which case all files in the folder are processed. However, if the path contains wildcards, then subfolders will not be processed.
var = file.readall(path)	Reads entire file from path and assigns to variable var.

file.writeall(path, var)
file.writeall(path, var, "a")
	First form overwrites the file at path with contents of variable var. Second form appends contents of variable var to file at path.

str = file.name(path)	returns file name without extension or folde e.g. file.name("c:/path/to/fname.ext") is fname.
str = file.type(path)	returns file type (extension) without leading period e.g. file.type("c:/path/to/file.ext") is ext.

str = file.nametype(path)	returns file name and type without folder, e.g. file.nametype("c:/path/to/file.ext") is file.ext.
str = file.folder(path)	returns folder, e.g. file.folder("c:/path/to/file.ext") is c:/path/to.
n = file.size(path)	returns file size in bytes; returns "" if file cannot be opened.
n = file.ksize(path)	returns file size in kilobytes (size/1024); returns "" if file cannot be opened.
str=file.lastmodified(path)	returns the last modified date and time of the file as a string of 12 digits : yyyymmddhhmmss.
str=file.getdate(path, s)	If s == "c" or "C", returns the created date and time. If s == "m" or "M", returns the modified date and time. Returned value is a string of 12 digits : yyyymmddhhmmss.
str=file.setdate(path, s, date, time)	If s == "c" or "C", changes the created date and time. If s == "m" or "M", changes the modified date and time. The date must be specified as eight digits yyyymmdd and the time as six digits ddhhmmss. If date is omitted, today is assumed; if time is omitted, now is assumed; if s is omitted, "m" (modified date) is assumed.
str=file.version(path))	Returns file version info as four blank separated numbers. Returns "" if no version info.
str = file.volume(path)	Returns the volume name for the file path.
str = file.alldrives	Returns a multi line string, with one line per valid drive. Use line function to work with each drive.
res=file.createShortcut(target, fpathlnk, desc, params, iconpath, iconindex)	Creates a shortcut to target with command line arguments params in fpathlnk and sets description to desc (defaults to target if omitted). If present, iconpath and iconindex give path and zero based index to icon. Adds .lnk extension to fpathlnk if not there. Returns 1 if successful, zero otherwise. Only target and fpathlnk need be specified; other arguments are optional.

str=file.resolve(fpath)
str=file.resolve(fpath, "info"))
	Resolves shortcut fpath and returns target file if first form used. If second form used, info can be any of parameters, description, work, icon, and the indicated info is returned. Only the first three characters (eg "par") are needed.

res=file.getShortPath(fpath)	Returns short (8.3) path and file name to specfied file path and name.

exitcode = file.runwait(maxwait, "c:/path/to/progr.exe", "params", "work","howstart")	Runs the specified program (which can be .exe, .bat, .cmd) with specified parameter and workdirectory and waits up to maxwait milliseconds for it to end. Howstart can be min, max, hide, normal. The final three parameters are optional. If maxwait is zero, then there is no maximum wait. The result, exitcode, is set to the exit code of the program or to 9999999 if maxwait was exceeded.
file.runcallback(maxwait, "PPro Cmd", "c:/path/to/progr.exe", "params", "work","howstart")	Runs the specified program (which can be .exe, .bat, .cmd) with specified parameter and workdirectory. Howstart can be min, max, hide, normal. The final three parameters are optional. This call returns immediately, but when the program ends or maxwait milliseconds elapse, the PowerPro command PPro cmd is executed (this command could be a script call). In addition, the global variable _exit_ is set to the exit code or to 9999999 if maxwait is exceeded. If maxwait is zero, then there is no maximum wait. Each runcallback activates a new thread, so you can use as many as you want.
res=file.runas("userid", "password", "c:/path/to/prog.exe", "cmd line", "c:/folder")	Runs prog.exe under userid with password. The last two parameters are optional. If present, c:\folder gives current directory for program. If present, cmd line gives command arguments. Depending on the program run, the cmd line argument may have to start with the path to the program being run (in double quotes if blanks included). Returns 1 if successful, 0 otherwise.
res=file.runaswait("userid", "password", "c:/path/to/prog.exe", "cmd line", "c:/folder",maxwait)	Runs prog.exe under userid with password and waits for it to complete. The last two parameters are optional. If present, c:\folder gives current directory for program. If present, cmd line gives command arguments. Depending on the program run, the cmd line argument may have to start with the path to the program being run (in double quotes if blanks included). Returns result of command. If maxwait is present and greater than 0, waits at most maxwait milliseconds.
res=file.doverb(fpath, verb, nocontext)	Runs verb on fpath. The verb can be "properties" or any of the actions available for the file type (eg, "open", "print") as listed under Explorer|Tools|File Options|File Types|Advanced, or any verb that appears on explorer right click context menu. However, if the third argument nocontext is present and set to 1, only verbs in the registry are allowed.
han= file.watchfolder(path, cmd, maxwait, keywords)	Monitors activity in the folder given by path: when a change occurs, executes the cmd. If maxwait is specified, it gives the maximum wait time in milliseconds. If it is omitted or 0, there is no maximum. The keywords give a string of blank-separated keywords, eg "subtree size". The allowed keywords are: subtree: monitor subfolders of path as well file: monitor for new, deleted or renamed files folder: monitor for new, deleted or renamed folders size: monitor for changes in file size lastwrite: monitor for changes in last write date of file security: monitor for changes in security settings of file If keywords is omitted, then "file" is assumed. Each watchfolder

activates a new thread, so you can use as many as you want.

The global variable _exit_ is set according to how the folder changed, the global variable _folder_ is set to the folder, and and the global variable _file_ is set to the name of the changed file (omitting the path to the watched folder). The possible values of _exit_ are 0 for unknown, 1 for timeout, 2 for file added, 3 for file deleted, 4 for file attributes changed, 5 for file renamed. For Win 95/98/Me, _file_ is always set to "" and _exit_ to 1 or 0.

For W2K and later, if the cmd ends in a comma then a parenthesis, for example, "myscript(,)", then the folder name and the file name are inserted as the last two arguments. It is recommended that this approach be used if you have multiple outstanding watches, as if more than one watch ends at the same time, it is possible that the global variables will be overwritten.

The returned handle can be used to end the watch with the command

file.endwatchfolder(han)

Expressions and Functions

Math Plugin

Math overview

The math plugin provides the following standard mathematical functions:

sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, log, log10, exp, ceil, floor, sqrt. For example

x = math.sin(3.14159/2)

In addition, the service isNan can be used to check if the floating point number is set to the value Nan (Not a Number) due to invalid operation. The service isfinite checks for finite values.

You can round a value with math.floor(x+0.5).

If x is a floating point number, you can use dot syntax to call functions on x, such as

x.sin

x.format(5)

The following service is provided for formatting floating point numbers.

r2 = Math.Format(r1, "z", nwide, ndec, regional)

Formats number r1 in a field of width nwide with ndec decimal places. Set nwide to 0 to get minimum field with to display number. The character z can be one of

e - exponent notation

f - fixed decimal

g - use f if possible, e otherwise.

If regional is set to character "r", regional settings are used. Otherwise, the period is used.

You do not have to specify all the arguments. The following shortened versions are allowed:

Format(r1) // same as Format(r1, "g")

Format(r1,"%nwide.ndecz") // specify format in one string

Format(r1, ndec) // same as Format(r1, "g", 0, ndec)

Format(r1, nwide, ndec) // same as Format(r1, "g", nwide, ndec)

Expressions and Functions

Int64 Plugin

int64 plugin overview

The int64 plugin lets you do arithmetic with 64 bit integers.

These numbers are not directly supported by PowerPro, so you must put any numbers greater than 2**31-1 in quotes when you use the plugin. Quotes are optional for smaller numbers For example, int64.mul("1234567891234", 100) yields "123456789123400".

Since 64 bit numbers are stored as strings, the plugin has to convert the numbers to and from strings when working with them. This means that this is not a very efficient way to do arithmetic. But is good enough for casual use in a PowerPro script.

list of services

Int64.Add	Adds two to nine nine numbers and returns result in string.
Int64.Sub	Subtracts two numbers.
Int64.Mul	Multiplies two to nine numbers.
Int64.Divide	Divides two numbers.
Int64.Mod	Returns remainer of integer division of two numbers.
Int64.Compare	Compares two numbers, returning 1, 0, or 1.

Expressions and Functions

Note Plugin

note plugin overview

The note plugin lets you create notes and manipulate the text in notes. (Notes are windows so you can also use the win plugin on note handles).

list of services

note.open	Creates a new note and returns its handle (which is the window handle for the note).
note.handlelist	Returns a blank separated list of handles of open notes.
note.close	Closes and keeps note.
note.delete	Closes and deletes note.
note.gettext	Returns all or a selection of the characters in a note.
note.getline	Returns a line in a note.
note.replacetext	Replaces or deletes text in a note.
note.inserttext	Inserts characters into a note.
note.appendtext	Adds text to end of note.
note.insertline	Inserts a line into a note.
note.replaceline	Inserts a line into a note.
note.appendline	Adds a line to the end of a note.
note.replacesel	Replaces the selection in a note.
note.getcategory	Sets category of note.
note.find	Finds text in a note.
note.getsel	Gets start and end character positions of any selected text in note.
note.getsel1	Gets start character position of any selected text in note
note.getsel2	Gets end character position of any selected text in note
note.setsel	Sets start and end of selection.
note.linefromchar	Returns line index corresponding to character position.
note.charfromline	Returns characters position corresponding to start of line position.
note.getcaretpos	Returns character position of text caret
note.getcaretline	Returns line position of text caret
note.settab	Sets the tab position for the note.
note.charcount	Returns the number of characters in a note.
note.linecount	Returns number of lines in a note.
note.left	Returns left window coordinate of Note ion pixels.
note.right	Returns right window coordinate of note.
note.width	Returns width of note in pixels.
note.height	Returns height of note in pixels.
note.right	Returns right coordinate of note.
note.bottom	Returns left coordinate of note.
note.move	Moves note to new position
note.roll	Rolls or unrolls a note
note.show	Hides or shows a note.
note.ontop	Makes a note topmost or not topmost.
note.isvisible	Returns visibility state of note.
note.isvalid	Returns 1 if note valid note handle supplied.

note.isrolled	Returns 1 if note is rolled.
note.isontop	Returns 1 if note is ontop.
note.gettextcolor	Gets text color
note.getbackcolor	Gets back color
note.settextcolor	Sets text color of note.
note.setbackcolor	Sets background color of note.
note.settab	Sets tab in note.
note.setcategory	Sets category of note.
note.vscroll	Scrolls lines or pages, up or down.
note.scrollto	Scrolls to indicated line or page.
note.scrollcaret	Scrolls note so that caret is in view.
note.cycle	Cycles though open notes, one at a time, showing them as it goes.
note.raise	Cycles through open notes and removes always on top property.

description of services

In the following descriptions, nh is a note window handle returned by note.open or note.handlelist, str is a string, fpath is a full path to a file stored in string, lindex is the zero-based index of a line in a note, and cindex is the zero-based index of a character in a note.

All of the set functions return a note handle to allow service calls to be chained.

nh = note.open(fpath, keywords, show_flag)	Creates a new note and returns its (window) handle. All of the parameters are optional. If the fpath is specified, it provides a file containing text for the note; this file may be an ordinary text file or a saved note. If the keywords string is specified, it provides initial settings for the look, position, and text of the note. If show_flag is omitted or specified as "1", the note is shown but does not become the foreground window; if show_flag is 2, the note is shown and activated; if it is 0, the note is created but not shown. To create a note and set its look (eg position, color), you can create the note as hidden, set the look with note services like note.size and note.settextcolor, then show the note with note.show(hh, 1).Or, you can also use the keywords to set the note look. To see the values of the keywords, use the configuration dialog to create a note open command. Press the binoculars (find) button beside "enter keywords" edit box, use the resulting dialog to select desired options, then copy the resulting keywords from the edit both to the note.open command.
nhList = note.handleList(strcat, invisible_flag)	Returns a blank separated list of all handles of open notes, limited to about 55 open notes. If strcat is specified and not "*", returns only those notes with the specfied category. If invisible_flag is specified as "1", returns both visible and hidden notes, else just returns visible notes.
nh.close	Closes and saves the note with handle nh.

nh.delete	Closes and deletes the note with handle nh.
str=nh.gettext(cindex1, cindex2)	Returns the characters in the note with handle nh starting at cindex1 and ending and cindex2, and including cindex2. Both parameters are optional: if cindex1 is omitted, it is taked as zero (start of note); if cindex2 is omitted, it is taken as the last character of the note. So note.getText(nh) returns all of the text in the note, and vec.createFromLines(note.getText(nh)) creates a vector with one element per line of note.
nh.replaceText(str, cindex1, cindex2)	Deletes the text starting at cindex1 and ending at cindex2 (inclusive) and replaces it by str. Set str to "" to simple delete the text. Omit cindex1 and cindex2 to replace all text. If cindex1 is greater than cindex2, nothing happens. If cindex1 is greater than the length of the note, the characters are added to the end of teh note. If cindex2 only is omitted, it is taken as the end of the note.
nh.insertText(str, cindex)	Inserts the text in str before cindex. Use cindex of 0 to insert at start. Omit cindex or use cindex bigger than lenght of note to add to end.
nh.appendText(str)	Inserts the text in str at the end of the note.
strLine=nh.getLine(lindex)	Sets strLine to the contents of line lindex (starting at line 0) for note with handle nh; remove terminating /r/n from text
nh.replaceLine(str, lindex)	Replaces the indicates line by the str. A \r\n is added to the end of str if not already present. To delete line lindex, set str to "". If lindex is 1, the line with the current selection is replaced. If line is greater than the length of the note, the line is added to the end
nh.insertLine(lindex, str)	Inserts the the indicates line before lindex. A \r\n is added to the end of str if not already present. If lindex is 0, inserts at start of note. If line is greater than the length of the note, the line is added to the end.
nh.appendLine(str)	Appends the str at the end of the note. A \r\n is added to the end of str if not already present. If lindex is 0, inserts at start of note.
nh.replaceSel(str)	Replaces any selected text in the note with the str. Does nothing if no selection. Deletes any selection if str is "".
str=nh.getsel	Sets string to two blank separated numbers indicated the first and last character of selected text. Str is set to "" if no selection.

nh.getsel(cindex1, cindex2)	Sets start and end of selection to cindex1 through cindex2 inclusive.
str=nh.getsel1	Sets string to zero based number indicated the first character of selected text. Str is set to "" if no selection.
str=nh.getsel2	Sets string to zero based number indicated the last character of selected text. Str is set to "" if no selection.
lindex=nh.linefromchar(cindex)	Returns the line number corresponding to a specific character.
cindex=nh.charfromline(lindex)	Returns the character index corresponding to a specific line.
cindex=note.getcaretpos(nh)	Returns the character index corresponding to the current position of the caret (blinking text cursor).
cindex=nh.getcaretline	Returns the line index corresponding to the current position of the caret (blinking text cursor). Example: str=note.getline(nh,note.getcaretline(nh)) reads the line containing the care.
n=nh.find(text,start,case,nobeep)	Finds text in note with handle nh. If start is specified and greater than or equal to zero, starts seach there; else starts just past current selection. Ignores case unless case is 1. Beeps if text not found unless nobeep is 1.
nh.setTab(tabpos)	Sets the tab position for the note; must be at least 1.
len=nh.charCount	Returns the number of characters in the note.
lines=nh.lineCount	Returns the number of lines in the note.
pixels=nh.left	Returns the position or size of note. You can also use nh.top, nh.bottom, nh.right, nh.height, nh.width.
nh.getCategory	Resturns category of note.
nh.move(left,top)	Moves the note to the specified position.
nh.size(width,height)	Sets the size of the note.
nh.show(indicator)	Hides note if indicator is zero. Shows it if indicator is 1.

	Reverses setting otherwise. Shown notes are not activated.
nh.ontop(indicator)	Makes note not topmost if indicator is zero. Makes it topmost if indicator is 1. Reverses setting otherwise.
nh.roll(indicator)	Rolls note if indicator is 0. Unrolls it if indicator is 1. Reverses setting otherwise.
note.isValid(nh)	Returns 1 if nh is a valid note handle, zero otherwise.
res=nh.isVisible	Returns 1 if note is visible, 0 otherwise.
res=nh.isRolled	Returns 1 if note is rolled, 0 otherwise.
res=nh.isontop	Returns 1 if note is topmost, 0 otherwise.
str=nh.getTextColor	Returns text color as string of three numbers separated by blank: "red green blue".
str=nh.getBackColor	Returns background color as string of three numbers separated by blank: "red green blue".

nh.setTextColor(red,green,blue)
nh.setTextColor("red green blue")nh.setTextColor(rgbnumber)
	Sets text color; red, green, blue are numbers between 0 and 255. There are three ways to specify a color: specify the red, green, and blue components are three separate parameters; specify the red, green, and blue components in a single blank separated parameter string; or specify the red, green, and blue components as a single integer with red in the lowest 8 bits (as returned by win.getpixel and inputcolor). Numbers are decimal unless they start with 0x in which case they are taken to be hexadecimal. For example, all of the following specify the same color with red=4, green=128, blue=255:
	4, 128, 0xff
	"4 0x80 255"
	4 + 256*(128+256*255))
	0xff8004

nh.setBackColor(h,red,green,blue)
nh.setBackColor("red green blue")
nh.setBackColor(rgbnumber)
	Sets note background color; red, green, blue are numbers between 0 and 255. See SetTextColor for description of ways to set colors.

nh.setCategory(str)	Changes category of note.

nh.vscroll(n)	Scrolls note depending on n: 2 scrolls up one page, 1 scrolls up one line, 0 does not scroll, 1 scrolls down one line, 2 scrolls down one page, otherwise scrolls done one line.
nh.scrollTo(hchar,lindex)	Scrolls note to horizontal character hchar of line lindex.
nh.ScrollCaret	Scrolls note so caret is in view.
note.cycle	This service cycles through your open notes (works like alt+tab). It's probably a good idea to assign this service to a hotkey.
note.raise	This service cycles through your open notes (works roughly like alt+esc). It's probably a good idea to assign this service to a hotkey. This service will remove the "always on top" properties of your notes as it cycles through them.

Expressions and Functions

Win Plugin

win plugin overview

The win plugin retrieves information about a window or its child windows and can also manipulate windows: minimize, maximize, close, and so on. It provides many more services than the built-in window command. However, whereas the window command will usually work with all windows matching a given caption list, the win plugin works with only the first visible one (or first hidden one, if no visible windows match).

The plugin can send and post window messages.

The plugin can play system sounds and also activate the low-level beep.

The plugin has services to access the current language, desktop name, and other system metrics.

Finally, the plugin provides access to keys and debug functions of PowerPro.

list of services

The plugin contains many services; here are lists by category:

DIMENSIONS	left, right, top, bottom, width, height, clientwidth, clientheight
HANDLES	handle, mainhandlefrompoint, handlelist, handlefrompoint, childhandlelist, handlefromindex, menu

FOCUS	getfocus, setfocus, resetfocus
INFORMATION	exists, gettext, getprocessid, caption, class, exepath, exename, visible, maxxed, minned, topmost, rolled, trayminned, enabled, hascaption, resizable, maxable, minable, toolwindow, fullscreen, parent, owner, visibleowner, getworkingdir, match
MANIPULATE	close, closeforce, rollup, enable, traymin, ontop, trans, transmouse, show, showna, hide, minimize, maximize, move, size, settext, dropfile, back, flash, tofile, recttofile,settool
CHILD WINDOWS	parent, idfromhandle, handlefromid, handlefromindex,indexfromhandle, childhandlelist, childtextbyindex, childtextbyid
COORDINATES	getworkarea, makerect, mappoints, getrect, getclientrect,setrect, size, move, setpos10000
SYSTEM	sendkeys, keys, beep, messagebeep, debug, debugshow, debugclear, setdebug, message, hex, getdesktopname, lastmouse, sendmessage, postmessage, sendcopydata, setworkarea, greatworkarea, getprinters, setdefaultprinter, getpixel, getpixel1, getsystemmetrics, getdisplayrect, area, getlanguage, crosshair

description of services

In the following descriptions, cl is a caption list as defined in the PowerPro help. Note that this means it could be window handle, since a window handle is a valid captionlist. The win plugin first searches for a visible window matching cl; if none is found, it then tries to find an invisible window matching cl.

In the following descriptions, han represents a window handle obtained from win.handle, win.handlelist, etc.

For any services which take a window handle or a captionlist as the first parameter, you can use the syntax

han.service(params)

to call the service (as a shorter alternative to win.service(han, params)

window dimensions

left(cl)	returns left window coordinate of first window matching cl
right(cl)	returns right window coordinate of first window matching cl
top(cl)	returns top window coordinate of first window matching cl
bottom(cl)	returns bottom window coordinate of first window matching cl

width(cl)	returns width first window matching cl
height(cl)	returns height of first window matching cl
clientwidth(cl)	returns client width first window matching cl
clientheight(cl)	returns client height of first window matching cl

window handles

handle(cl)	returns window handle of first window matching cl
handle(cl,"text")	returns handle of child window contain text of first window matching cl
mainhandlefrompoint(x, y)	returns window handle of main (parent) window at under screen position x,y
handlefrompoint(x,y)	returns window handle of window at under screen position x,y
handlelist(cl, inv)	return blank separate list of window handles matching cl. Set inv to 1 to include hidden windows. Set inv to 2 to include hidden windows, excepting those with blank captions.
childhandlelist(cl, "text")	return blank separate list of child window handles for main window matching cl. Omit "text" for all. Set "text" to "c=classname" for only those child windows of class classname. If text is anything else, only windows with window text matching text are retrieved.
handlefromindex(cl, ix)	returns handle of child window of main window matching cl where child window has index number ix
menu(cl, param)	Shows a menu of all active windows matching cl and returns handle of selected one, or zero if menu dismissed with no selection. Set cl to "" for all windows. Optionally include param and set to one of "hidden", "traymin", "onlyhidden" as for the *window menu command to select those types of windows.

window focus

getfocus()	returns handle of window with focus; may be a child window
setfocus(han)	sets focus and foreground to window with handle han; may be a child window
resetfocus	if PowerPro bar or Taskbar icon is foreground, makes previous window the foreground

get window information

exists(han)	returns 1 if han is a valid window handle (may be a child window); 0 otherwise
gettext(han)	issues WM_GETTEXT to window with handle han; the receiving window may return its text
getprocessid(han)	returns the process id of the process for the window with handle han
caption(cl)	returns window caption of first window matching cl
class(cl)	returns window class of first window matching cl
exepath(cl)	returns full path to exe of first window matching cl
exename(cl)	returns exe name (no file extension)of exe of first window matching cl
visible(cl)	returns 1 if first window matching cl exists, 0 if hidden exists, empty string otherwise
maxxed(cl)	returns 1 if first window matching cl is maximized, 0 otherwise
minned(cl)	returns 1 if first window matching cl is minimized, 0 otherwise
topmost(cl)	returns 1 if first window matching cl is topmost, 0 otherwise
rolled(cl)	returns 1 if first window matching cl is rolled up, 0 otherwise
trayminned(cl)	returns 1 if first window matching cl is tray minned, 0 otherwise
enabled(cl)	returns 1 if first window matching cl is enabled, 0 otherwise
hascaption(cl)	return 1 if first window matching cl has a caption, 2 if toolbar caption, 0 otherwise.
resizable(cl)	returns 1 if first window matching cl is resizable, 0 otherwise
maxable(cl)	returns 1 if first window matching cl has maximize box, 0 otherwise
minable(cl)	returns 1 if first window matching cl has minimize box, 0 otherwise

toolwindow(cl)	returns 1 if first window matching cl is toolwindow, 0 otherwise
fullscreen()	returns 1 if foreground window is full screen; 0 otherwise.
fullscreen(1)	returns 1 if any window of the same process as the foreground window is full screen.
parent(han)	returns the parent of the window with handle han
owner(han)	returns the owner of the window with handlehan
visibleowner(han)	returns the owner of the window with handlehan if it is visible and not zero-sized
getworkingdir(cl)	gets working directory of window matching cl
match(han, cl)	returns true if main window with handle han matches cl

manipulate windows

All of the manipulate services return the handle to the manipulated window so that these services can be chained, as in

myhandle.ontop(1).transmouse(100).move(50,200)

close(cl)	closes first window matching cl
closeforce(cl)	closes first window matching cl, unsaved information is lost
rollup(cl)	rolls up window matching cl; unrolls if already is rolled up
enable(cl,flag)	enables window matching cl if flag is anystring not starting with "0"; disables otherwise
traymin(cl)	tray minimizes first window matching cl; un tray mins if already is
ontop(cl)	makes first window matching cl topmost; removes topmost if it already is
ontop(cl,flag)	if flag is 0, makes window not topmost; if 1, makes it topmost; else reverses state
trans(cl,num)	makes window matching cl transparent num, num<=255. If num negative, reverses transparency state.

transmouse(cl,num)	makes window matching cl transparent, num<=255. If num negative, reverses transparency state. Mouse clicks go through window as well.
show(cl,howshow)	shows first window matching cl. Uses PowerPro internal window activation process unless second argument howshow present and set. Use: 1 for restore if icon and show; 2 ignore icon state, just show; 3 set foreground only.
showna(cl)	shows the window matching cl but does not activate it
hide(cl)	hides first visible window matching cl
minimize(cl)	minimizes first visible window matching cl
maximize(cl)	maximizes first visible window matching cl
restore(cl)	restore first visible window matching cl
back(cl)	sends window to back (bottom) of visible windows
move(cl,x,y)	moves first window matching cl to position x, y; you can also specify as win.move(cl, "x y")
size(cl,x,y)	sizes first window matching cl to size x,y; you can also specify as win.size(cl, "x y")
settext(cl, "newtext")	Issues WM_SETTEXT which may set the text of the window matching cl to newtext. cl could be a handle retrieved by handle(cl) for main windows or handle(cl,"text") for child windows
flash(cl, n)	Flashes Taskbar button for first window matching cl up to n times or until window put in foreground. Omit n for flash until foreground.
dropfile(cl, filepath)	simulates drag and drop of file name onto first window matching cl. Does not work for all programs.
tofile(cl, filepath, width, height, cflag)	Write contents of window to bmp filepath. Window resized to given width and height; if these omitted, no resizing done. If cflag present and set to 1, only client portion of window is written.
recttofile(rect, filepath, width, height)	Write contents of screen rectangle to bmp filepath. Resized to given width and height; if these omitted, no resizing done. Rect format is

	same as win.makerect.
settool(cl, flag)	Changes toolwindow style of first window matching cl; toolbar style windows are not shown on the Taskbar and have a narrow caption. Set flag to 0 to clear tool setting, to 1 to set, to –1 to toggle.

child windows

parent(han)	returns the parent of the window with handle han
parent(han,1)	returns the top level parent of the window with handle han
idfromhandle (han)	returns window id for child window with handle han.
handlefromid(cl, id)	returns handle for child window with id id and parent cl
handlefromindex(cl, ix)	returns handle of child window of main window matching cl where child window has index number ix
indexfromhandle(han)	returns index of child window with handle han
childhandlelist(cl, "text")	return blank separate list of child window handles for main window matching cl. Omit "text" for all. Set "text" to "c=classname" for only those child windows of class classname. If text is anything else, only windows with window text matching text are retrieved.
childtextbyindex(cl,n)	returns the text in the nth childwindow of first window matching cl. If there are less than n child windows, sets variable _EOF_ to 1; else sets _EOF_ to 0. First looks for visible windows matching cl; if none, tries hidden windows matching cl.
childtextbyid(cl,n)	returns the text child window with id n of first window matching cl. If there is no such id, sets variable _EOF_ to 1; else sets _EOF_ to 0. First looks for visible windows matching cl; if none, tries hidden windows matching cl. Child window ids are not necessarily unique.

window coordinates

In the following, rect represents is a string of four blank-separated integers giving a rectanble as "left top right bottom"; pt represents a screen point as "left top".

rect=win.getworkarea	returns screen work area (area outside visible Taskbar)
rect=win.makerect(left, top,right, bottom)	builds rect structure from four numbers

win.getclientrect(cl)	returns the client rectangle of the first window matching cl
win.getrect(cl)	returns the rectangle of the first window matching cl
win.setrect(cl, rect)	sets the size and position of the first window matching cl
win.size(cl, pt)	sets the window size of first window matching cl
win.move(cl, pt)	sets the window position of first window matching cl
win.setpos10000(cl,"s or w", left, top, right, bottom)	Sets position of first window matching cl. Second parameter indicates whether position set relative to full screen or work area. Integers left, right, top, bottom are numbers between 0 and 10000 and give position on virtual 10000 by 10000 screen to be mapped to actual screen positions at current resolution. Any of these can be set to "=" or "" or omitted to leave current value unchanged. Left or top (or both) can be set to "c" to center window in that dimension.
pts = mappoints(rect1, rect2, ptsIn)	ptsIn contains one or more point as blank separated numbers. These are mapped from coordinate system in rect1 to coordinate system in rect2. For example: mappoints(win.makerect(0,0,xscreen, yscreen), win.getworkarea, han.getrect) maps from whole screen to within work area mappoints("0 0 1000 1000", win.getworkarea,"0 0 250 250") maps to upper left quarter of work area

system actions

sendkeys(sz)	sends keys in string sz to active window
keys(sz)	sends keys in string sz to active window (same as sendkeys)
mouse(sz)	executes mouse commands in sz (same command format as *Mouse)
mouseto(x,y)	moves mouse cursor to screen position x, y
beep(freq, dur)	uses internal speaker to produce tone at frequency freq for dur milliseconds
messagebeep("type")	produces system sound; set "type" to one of "ok", "hand", "exclamation", "question"
debug(sz1, sz2)	up to 6 arguments can be specified; they are joined and shown in a debug window.
debugshow(sz1, sz2)	up to 6 arguments can be specified; they are joined and shown in a debug window which is made the foreground window.

debugclear	Clears debug window.
setdebug(n)	if n is 0, pauses debug output. If n is one, resumes. If n is omitted, returned current setting. If sz is "file", debug output is redirected to file debug.txt in PowerPro folder; if "both" debug output goes to both file and screen; if "screen", debug output goes to debug window only.
setworkingdir(path)	sets PowerPro working directory
message(message, title)	displays a messagbox
hex(n)	returns hexadecimal equivalent for n
getdesktopname()	returns name of Windows desktop; returns "" if desktop cannot be opened

lastmouse(n)	if n=0, returns 1 if left was last mouse button up, 2 for middle, 3 for right. If n>0, sets default mouse return for next getlastmouse(0). To see which mouse button was used for vec.showmenu, use
	win.lastmouse(1)
	vec.showmenu(...)
	mouse=win.lastmouse(0)

sendmessage(cl, msg, wp, lp)	sends message msg to first window matching cl
postmessage(cl, msg, wp, lp)	posts message msg to first window matching cl
sendcopydata(cl, dword, string)	sends wm_copydata first window matching cl, with dwdata=dword and lpdata=string
getworkarea()	returns the screen work area as four blank separated integers: left, top, right, bottom
setworkarea(words)	sets the work area according to the four blanks separated integers in words: left, top, right, bottom
setworkarea(left,top,right,bottom):	sets the screen work area
setworkarea()	sets screen work area to full size of physical screen

getprinters()	returns multi line string with one printer per line
setdefaultprinter("printer")	sets default printer; returns 1 if successful, 0 otherwise. Only for W2000 and later.
getpixel(x,y)	return red, green, blue pixel at screen position x, y where (0,0) is top left. Returned as three separate numbers. Use word function to access any one of them: word(win.getpixel(100,200),2) returns blue value at screen position 100, 200;
getpixel1(x,y)	return red, green, blue pixel at screen position x, y where (0,0) is top left. Returned as a single number.
getsystemmetrics(string)	calls windows function to access system metrics; see www.microsoft.com for documentation of this function (try http://msdn.microsoft.com/library/default.asp?url=/library/en us/sysinfo/base/getsystemmetrics.asp Note that this function is called with a text string set the the manifext constant name, eg win.GetSystemMetrics("SM_CXSCREEN"). Or you can use the integer value directly as shown on the MS reference web page. eg win.GetSystemMetrics(0).
getdisplayrect(digit, workflag)	returns screen rectangle as four blank separated integers: left, top, right, bottom for monitor number digit (0<=digit<=9). If workflag is "1", returns work area, else returns full screen rectangle. The digit number for one of multiple displays is arbitrary and could be changed by Windows on each boot.
area(x,y)	returns window area (eg client, caption, max button) under screen position x,y; eg win.area(xmouse, ymouse) looks at window under mouse. The complete list of strings which can be returned is shown below.
crosshair(x, y)	Draws a cross hair centered at x, y. Omit x, y to draw at mouse coordinates. Draw again with same coordinates to erase.
getlanguage(cl)	Returns an integer representing current language being used by first window matching cl. The following table gives this values in hexadecimal, ie this table gives the value return by win.hex(win.getlanguage)

strings returned from area(x,y)

BORDER In the border of a window that does not have a sizing border.

BOTTOM In the lower horizontal border of the window.

BOTTOMLEFT In the lower-left corner of the window border.

BOTTOMRIGHT In the lower-right corner of the window border.

CAPTION In a title-bar area.

CLIENT   In a client area.

GROWBOX   In a size box.

HSCROLL   In the horizontal scroll bar.

LEFT   In the left border of the window.

MAXBUTTON   In a Maximize button.

MENU   In a menu area.

MINBUTTON   In a Minimize button.

CLOSE   In a Close button.

NOWHERE   On the screen background or on a dividing line between windows.

REDUCE   In a Minimize button.

RIGHT   In the right border of the window.

SYSMENU   In a Control menu or in a Close button in a child window.

TOP   In the upper horizontal border of the window.

TOPLEFT   In the upper-left corner of the window border.

TOPRIGHT   In the upper-right corner of the window border.

TRANSPARENT   In a window currently covered by another window.

VSCROLL   In the vertical scroll bar.

hexadecimal language codes returned by getlanguage(cl)

0x0400 Process or User Default Language

0x0800 System Default Language

0x0436 Afrikaans

0x041c Albanian

0x0401 Arabic (Saudi Arabia)

0x0801 Arabic (Iraq)

0x0c01 Arabic (Egypt)

0x1001 Arabic (Libya)

0x1401 Arabic (Algeria)

0x1801 Arabic (Morocco)

0x1c01 Arabic (Tunisia)

0x2001 Arabic (Oman)

0x2401 Arabic (Yemen)

0x2801 Arabic (Syria)

0x2c01 Arabic (Jordan)

0x3001 Arabic (Lebanon)

0x3401 Arabic (Kuwait)

0x3801 Arabic (U.A.E.)

0x3c01 Arabic (Bahrain)

0x4001 Arabic (Qatar)

0x042b Windows 2000/XP: Armenian. This is Unicode only.

0x042c Azeri (Latin)

0x082c Azeri (Cyrillic)

0x042d Basque

0x0423 Belarusian

0x0445 Bengali (India)

0x141a Bosnian (Bosnia and Herzegovina)

0x0402 Bulgarian

0x0455 Burmese

0x0403 Catalan

0x0404 Chinese (Taiwan)

0x0804 Chinese (PRC)

0x0c04 Chinese (Hong Kong SAR, PRC)

0x1004 Chinese (Singapore)

0x1404 Windows 98/Me, Windows 2000/XP: Chinese (Macao SAR)

0x041a Croatian

0x101a Croatian (Bosnia and Herzegovina)

0x0405 Czech

0x0406 Danish

0x0465 Windows XP: Divehi. This is Unicode only.

0x0413 Dutch (Netherlands)

0x0813 Dutch (Belgium)

0x0409 English (United States)

0x0809 English (United Kingdom)

0x0c09 English (Australian)

0x1009 English (Canadian)

0x1409 English (New Zealand)

0x1809 English (Ireland)

0x1c09 English (South Africa)

0x2009 English (Jamaica)

0x2409 English (Caribbean)

0x2809 English (Belize)

0x2c09 English (Trinidad)

0x3009 Windows 98/Me, Windows 2000/XP: English (Zimbabwe)

0x3409 Windows 98/Me, Windows 2000/XP: English (Philippines)

0x0425 Estonian

0x0438 Faeroese

0x0429 Farsi

0x040b Finnish

0x040c French (Standard)

0x080c French (Belgian)

0x0c0c French (Canadian)

0x100c French (Switzerland)

0x140c French (Luxembourg)

0x180c Windows 98/Me, Windows 2000/XP: French (Monaco)

0x0456 Windows XP: Galician

0x0437 Windows 2000/XP: Georgian. This is Unicode only.

0x0407 German (Standard)

0x0807 German (Switzerland)

0x0c07 German (Austria)

0x1007 German (Luxembourg)

0x1407 German (Liechtenstein)

0x0408 Greek

0x0447 Windows XP: Gujarati. This is Unicode only.

0x040d Hebrew

0x0439 Windows 2000/XP: Hindi. This is Unicode only.

0x040e Hungarian

0x040f Icelandic

0x0421 Indonesian

0x0434 isiXhosa/Xhosa (South Africa)

0x0435 isiZulu/Zulu (South Africa)

0x0410 Italian (Standard)

0x0810 Italian (Switzerland)

0x0411 Japanese

0x044b Windows XP: Kannada. This is Unicode only.

0x0457 Windows 2000/XP: Konkani. This is Unicode only.

0x0412 Korean

0x0812 Windows 95, Windows NT 4.0 only: Korean (Johab)

0x0440 Windows XP: Kyrgyz.

0x0426 Latvian

0x0427 Lithuanian

0x0827 Windows 98 only: Lithuanian (Classic)

0x042f Macedonian (FYROM)

0x043e Malay (Malaysian)

0x083e Malay (Brunei Darussalam)

0x044c Malayalam (India)

0x0481 Maori (New Zealand)

0x043a Maltese (Malta)

0x044e Windows 2000/XP: Marathi. This is Unicode only.

0x0450 Windows XP: Mongolian

0x0414 Norwegian (Bokmal)

0x0814 Norwegian (Nynorsk)

0x0415 Polish

0x0416 Portuguese (Brazil)

0x0816 Portuguese (Portugal)

0x0446 Windows XP: Punjabi. This is Unicode only.

0x046b Quechua (Bolivia)

0x086b Quechua (Ecuador)

0x0c6b Quechua (Peru)

0x0418 Romanian

0x0419 Russian

0x044f Windows 2000/XP: Sanskrit. This is Unicode only.

0x043b Sami, Northern (Norway)

0x083b Sami, Northern (Sweden)

0x0c3b Sami, Northern (Finland)

0x103b Sami, Lule (Norway)

0x143b Sami, Lule (Sweden)

0x183b Sami, Southern (Norway)

0x1c3b Sami, Southern (Sweden)

0x203b Sami, Skolt (Finland)

0x243b Sami, Inari (Finland)

0x0c1a Serbian (Cyrillic)

0x1c1a Serbian (Cyrillic, Bosnia, and Herzegovina)

0x081a Serbian (Latin)

0x181a Serbian (Latin, Bosnia, and Herzegovina)

0x046c Sesotho sa Leboa/Northern Sotho (South Africa)

0x0432 Setswana/Tswana (South Africa)

0x041b Slovak

0x0424 Slovenian

0x040a Spanish (Spain, Traditional Sort)

0x080a Spanish (Mexican)

0x0c0a Spanish (Spain, Modern Sort)

0x100a Spanish (Guatemala)

0x140a Spanish (Costa Rica)

0x180a Spanish (Panama)

0x1c0a Spanish (Dominican Republic)

0x200a Spanish (Venezuela)

0x240a Spanish (Colombia)

0x280a Spanish (Peru)

0x2c0a Spanish (Argentina)

0x300a Spanish (Ecuador)

0x340a Spanish (Chile)

0x380a Spanish (Uruguay)

0x3c0a Spanish (Paraguay)

0x400a Spanish (Bolivia)

0x440a Spanish (El Salvador)

0x480a Spanish (Honduras)

0x4c0a Spanish (Nicaragua)

0x500a Spanish (Puerto Rico)

0x0430 Sutu

0x0441 Swahili (Kenya)

0x041d Swedish

0x081d Swedish (Finland)

0x045a Windows XP: Syriac. This is Unicode only.

0x0449 Windows 2000/XP: Tamil. This is Unicode only.

0x0444 Tatar (Tatarstan)

0x044a Windows XP: Telugu. This is Unicode only.

0x041e Thai

0x041f Turkish

0x0422 Ukrainian

0x0420 Windows 98/Me, Windows 2000/XP: Urdu (Pakistan)

0x0820 Urdu (India)

0x0443 Uzbek (Latin)

0x0843 Uzbek (Cyrillic)

0x042a Windows 98/Me, Windows NT 4.0 and later: Vietnamese

0x0452 Welsh (United Kingdom)

Expressions and Functions

Plugin programming

this topic is intended only for programmers of plug-in dlls

see also

How to use plugins is described in the previous topic.

basics of programming plugins

A plugin is a dll which PowerPro loads dynamically. The services are the exported routines. Within the dll, the service names must be in lower case. Neither the plug-in name nor the service name can contain blanks. You can see sample plugins in the plugins zip file in the PowerPro folder.

You cannot use the names run or runfile for plugins.

the service declarations should be

_declspec(dllexport) void service(LPVOID unused1, LPVOID unused2,

int (*GetVar)(LPSTR szVar, LPSTR szVal), void (*SetVar)(LPSTR szVar, LPSTR szVal),

DWORD* pFlags, UINT nargs, LPSTR* szArgs, PPROSERVICES* ppsv)

(assuming chars are one byte).

Remember that lower case must be used for service names.

The first two arguments are always NULL and should be ignored.

The functions GetVar and SetVar access the variable given in the first string and get/set the value in the second string. GetVar set szVal to the first 531 characters of the variable with name in szVar.. If you want to work with variables of any length, use the PPROSERVICES function GetVarAddr. SetVar sets the value of the variable with name szVar to the string pointed at by szVal; this string can be of any length.

You can use (*GetVar)("pproversion", szVer) to access the PowerPro version as a four digit string.

The 32 bit unsigned values pointed at by pFlags holds PowerPro's 32 flags. Flag 0 is in the least significant bit. You can change or read any flag.

The variable nargs is a value between 0 and 23 to indicate the number of arguments used to call the plugin service. Pointers to the arguments are stored in string array szArgs. Argument 1 is at *(szArgs+1), argument 2 is at *(szArgs+2), and so on. The pointers can point at strings of any length. For arguments greater than narg, the parameters will point at empty strings.

You can return a result from the plugin from setting *szargs to the result if it is less than 531 characters in length or by using the AllocTemp and ReturnString functions of PPROSERVICES as described below.

PPROSERVICES

The pointer ppsv points to a PPROSERVICES structure which is a list of function pointers to functions provided by PowerPro.exe.

```
typedef struct tagPProServices
{
    void (*ErrMessage)(LPSTR, LPSTR);
    BOOL (*MatchCaption)(HWND, LPSTR);
    HWND (*FindMatchingWindow)(LPSTR,BOOL);
    BOOL (*IsRolled)(HWND hw);
    BOOL (*IsTrayMinned)(HWND hw);
    void (*GetExeFullPath)(HWND hw, LPSTR szt);
    void (*RollUp)(HWND hw);
    void (*TrayMin)(HWND hw);
    void (*SendKeys)(LPSTR sz);
    BOOL  (*EvalExpr)(LPSTR sz, LPSTR szo);
    void  (*Debug)(LPSTR sz1, LPSTR sz2,LPSTR sz3, LPSTR sz4, LPSTR sz5, LPSTR sz6);
    LPSTR (*AllocTemp)(UINT leng);
    void (*ReturnString)(LPSTR sz, LPSTR* szargs);
    LPSTR (*GetVarAddr)(LPSTR var);
    LPSTR (*SetVar)(LPSTR var, LPSTR val);
    void (*IgnoreNextClip)();
    void (*Show)(HWND h);
    void (*RunCmd)(LPSTR szCmd, LPSTR szParam, LPSTR szWork);
    BOOL (*InsertStringForBar)(LPSTR szStr, LPSTR szCmd);
    void (*ResetFocus)();
HWND (*NoteOpen)(LPSTR szFile, LPSTR szKeyWords, BOOL bActivate);
BOOL (*PumpMessages)();
BOOL (*RegForConfig)(void ( *callback )(LPSTR szList), BOOL bReg );
void (*SetPreviousFocus)(HWND h );
UINT (*SetDebug)(LPSTR sz,LPSTR sz2 );
UINT (*ScriptCancel)(LPSTR sz );
void (*GetCurrentDir)(HWND h,LPSTR szt);
void (*RegisterNonModal)(HWND h,BOOL b);
UINT (*GetVarSize)(LPSTR p);
BOOL (*RegisterSigOld)(BOOL b, LPSTR sig, LPSTR sig2, LPSTR szPlugName,
                void (*callback)(LPSTR sz), LPSTR szGet, LPSTR szSet,LPSTR szDo );
void (*FreeIfHandle)(LPSTR sz);
int (*LastMouse)(UINT u);
BOOL (*RegisterSig)(BOOL b, LPSTR sig, LPSTR sig2, LPSTR szPlugName,
    void (*callback)(LPSTR sz), LPSTR szGet, LPSTR szSet,LPSTR szDo, LPSTR szDoSet);
void (*ReturningFreeableHandle)();
LPSTR (*GetStaticVarAddr)(LPSTR szName, LPSTR szScript);
BOOL (*SetStaticVar)(LPSTR szName, LPSTR szScript, LPSTR szVal, BOOL bCreate);
HWND (*ActiveMenu)(LPSTR szCap, LPSTR szSw);
void (*SaveClip)(LPSTR szFileName, BOOL bTextOnly, BOOL bVerbose);
void (*LoadClip)(LPSTR szFileName, BOOL bTextOnly, BOOL bVerbose);
LPSTR (*EncodeFloat)(double x, LPSTR szBuff);
double (*DecodeFloat)(LPSTR szBuff);
void GetCaretPosScreen(HWND h, POINT * pt);
} PPROSERVICES
```

calls to pproservices

(ErrMessage)(LPSTR, LPSTR)
to show an error message consisting of the first string and then the second string; if the plugin is called from a script, the user will have the opportunity to stop all scripts.

(ppsv->MatchCaption)(hwnd, szCaptionList)
to use PowerPro's caption list matching engine to see if the window with handle hwnd matches the caption list string.

(ppsv->FindMatchingWindow)(szCaptionList, bHidden)
to use PowerPro's caption matching engine to find the first window matching the caption list szCaptionlist; set bHidden to 1 to include invisible windows.

(ppsv->IsRolled)(hw)
to check to see if the specifyied window is rolled-up by PowerPro; function returns 1 if it is and 0 otherwise.

(ppsv->IsTrayMinned)(hw)
to check to see if the specified window is trayminned by PowerPro; function returns 1 if it is and 0 otherwise.

(ppsv->GetExeFullPath)(hw, sz)
to get full path to exe of window hw

(ppsv->RollUp)(hw)
to rollup window hw if it is not already rolled up and to show it otherwise.

(ppsv->TrayMin)(hw)
to tray minimize window hw if it is not tray minimized and to show it otherwise.

(ppsv->SendKeys)(sz)
to send keys using sz; this is equivalent to using *keys command on sz. The string sz cannot

contain more than 500 characters.

(ppsv->EvalExpr)(szExpr, szResult)
to evaluate the expression stored in szExpr and return the result in szResult. The result is always truncated to 531 characters.

(ppsv->Debug)(s1,s2,s3,s4,s5,s6)
to display up to six strings in a line in the debug window; the strings are separated by blanks.

(ppsv->AllocTemp)(len)
to allocate a temporary variable of len bytes. PowerPro will automatically free this memory after the expression involving the plugin is complete (unless the temporary is assigned to a variable with SetVar). You must never free space allocated by AllocTemp and you should always use AllocTemp to allocate memory so that it can be freed when your plugin completes.

(ppsv->ReturnString)(szRes, szargs)
to return the string in szRes as the result of the plugin call. This string can be any length. A pointer to the string is stored in *szargs, prefixed by '\01' so that PowerPro can distinguish the result from a returned string stored directly in *szargs.

(ppsv->GetVarAddr)(szvar)
to get the address of the string stored in the variable named in szvar. You should consider the memory at this address to be read-only. If you want to change a variable, use (ppsv->SetVar).

(ppsv->SetVar(szvar, szval))
to set the variable named in szvar to szval. This is the same functions as (*SetVar) from the plugin argument list.

(ppsv->IgnoreNextClip)
in plugins setting the clipboard where the set text is not to be tracked; it causes PowerPro clipboard tracking to ignore the next copy of text onto the clipboard.

(ppsv->Show)
to show a window which may be tray-iconized; it invokes the same processing as *Window show..

(ppsz->RunCmd)
to run commands through PowerPro. The command line is contained in szCmd, the command parameters in szParam, and the initial folder area in szWork. If you run a *Command like *Menu, the action keyword and parameters can be split between szCmd and sz Param in any convenient way, e.g. (ppsv->RunCmd)("*menu show", menuvar, "centerscreen"). The maximum length of szCmd joined with szParam is 531 characters.

(ppsz->InsertStringForBar)(szStr, szCmd)
to insert the string szStr wherever the bar character | appears in szCmd.

(ppsz->ResetFocus)()
before a command which requires the window focus to be set properly. If the command was activated by pressing a PowerPro bar button or tray icon, it will be reset to window that had focus before PowerPro.

(ppsv->NoteOpen(szFile, szKeyWords))
to open a note and return its window handle. The szFile parameter may be "" or a file name containing either a note or text to put in the note. The szKeyWords parameter is a string of keywords setting the note format; see the note plugin text for a list of these keywords.

(ppsv->PumpMessages)()
to keep PowerPro responsive during long plugin operations. If the function returns 1, it means that PowerPro is reconfiguring or quitting, and so the plugin should return immediately.

(ppsv->SetPreviousFocus)(HWND h)
sets the handle to the window with the focus before PowerPro to h.

(ppsv->SetDebug)(LPSTR sz)
If starts with "0", pauses debug output. If sz starts with "1", resumes debug output. If sz is "", returns current setting of debug output without change.

(ppsv->ScriptCancel)(LPSTR sz)
If starts with "0", clears script cancel flag (which scripts to stop processing). If sz starts with "1", sets script cancel flag (setting the flag causes scripts to stop processing). If sz is "", returns value of flag as unsigned integer. If this flag is 1, the plugin should stop any loops which call scripts.

(ppsv->GetCurrentDir)(HWND h, LPSTR sz)
Returns working directory for program owning window h in sz.

(ppsv->RegisterNonModal)(HWND h, BOOL b)
If b is TRUE, registers a non-modal dialog window h with PowerPro which causes PowerPro to check for dialog messages to this window as part of PowerPro's message loop. If b is FALSE, stops the check for h.

(ppsv->GetVarSize)(LPSTR szvar)
to get the amount of memory allocated to the variable with name stored in szvar.

(ppsv->RegisterSigOld)

Obsolete. Use RegisterSig.

(ppsv->FreeIfLocal)(LPSTR sz)

To be used by plugin authors when freeing storage which may itself contain handles allocated by other plugins; the service will use that plugin registered free routine to free the handle and any storage it has allocated. See example in vec plug.

(ppsv->LastMouse)(UINT u)

if u is zero, returns last mouse button up or down message (eg WM_LBUTTONUP). Otherwise, uses u to set default for next message if no mouse action before LastMouse(0) call.

(ppsv->RegisterSig)

(BOOL b, LPSTR sig, LPSTR sig2, LPSTR szPlugName, void (*freecallback)(LPSTR sz), LPSTR szGet, LPSTR szSet,LPSTR szDo, LPSTR szDoSet);

(ppsv->ReturningFreeableHandle)

See registering signatures below.

(ppsv->GetStaticVarAddr)(LPSTR szName, LPSTR szScript)

Returns the address of the static var szName from script szScript. The script need not be running. Returns NULL if no such static variable defined in the script.

(ppsv->SetStaticVar)(LPSTR szName, LPSTR szScript, LPSTR szVal, BOOL bCreate)

Sets static var szName from script szScript to value szVal. The script need not be running. If the variable does not exist and bCreate is 0, returns 0. Else returns 1 (and creates the variable, if needed).

(ppsv->ActiveMenu)(LPSTR szCap, LPSTR szSw)

Displays menu of all active windows matching caption list szCap and returns handle of selection (NULL if none selected). The string szSw can contains onlyhidden, hidden, traymin, as discussed in *Window menu help.

(ppsv->SaveClip)(LPSTR szFileName, BOOL bTextOnly, BOOL bVerbose);
Saves clipboard in filename; flags specify only text, debug text on types of info saved.

(ppsv->LoadClip)(LPSTR szFileName, BOOL bTextOnly, BOOL bVerbose);
Loads clipboard from filename; flags specify only text, debug text on types of info loaded.

(ppsv->EncodeFloat)(double x, LPSTR szBuff);

Encodes double x in string buffer szBuff (up to 19 characters), removing zero bytes so encoded value can be treated as string. Returns pointer to szBuff.

(ppsv->DecodeFloat)(LPSTR szBuff);

Decodes and returns double in szBuff and returns it; szBuff can contain either an encoded double or a normal string representing a float (e.g. "3.2") which is decoded with strtod.

(ppsv->GetCaretPosScreen)(HWND h, POINT* pt);

Set pt to screen position of text cursor (mouse cursor if there is no text cursor). If h is non-NULL, sets foreground window to h first.

registering Signatures

The **RegisterSig** service allows the plug-in to request that PowerPro perform any or all of the followering services for the plugin: call the plugin to free storage when a local variable or temporary containing a plugin handle is freed; process array [] for the plugin; process dot notation handle.service for the plugin.

To use this service, the plugin author must choose two signatures and store the signatures at the start of all handles created by the plugin. The signature is a sequence of two or three bytes. The second byte cannot be 0x06. At least one of the bytes must be less then 0x20.

(A handle is a value stored by a plugin to use as a "pointer" to storage or table slots that the plugin has allocated. For example, the vec.create function returns a handle starting with "v\06" which is stored in the variable representing the vector; vec.create also allocates separate storage for the vector elements.)

The arguments are used as follows:

BOOL b – set 1 to register the signatures, 0 to unregister. Make sure to unregister at unload.

LPSTR sig – the first signature; only variables starting with this signature will be freed through the freecallback argument supplied to RegsiterSig.

LPSTR sig2 – the second signature; not freed by freecallback but processed for [] and dot. This signature should be used by plugin’s localcopy service and should be same length as sig1. See vec plugin help for explanation of local copy.

freecallback – if not NULL, this routine is called to free plugin storage whenever a local variable starting with the first signature sig is freed (by a script exiting).

LPSTR szGet – if present, this points to a string "get"; whenever PowerPro sees a refererence to han[expr] where han starts with sig or sig2, then PowerPro calls plugin.get(han, expr)

LPSTR szPut – if present, this points to a string "put"; whenever PowerPro sees a refererence to han[expr] = expr2, where han starts with sig or sig2, then PowerPro calls plugin.set(han, expr, expr2)

LPSTR szDo – if present and set to "", when PowerPro sees han.service(params) where han starts with sig or sig2, then PowerPro calls plugin.service(han, params). If szDo is present and set to string (say) "invoke", then using han.service(params) results PowerPro executing plugin.invoke(han, "service", params).

LPSTR szDoSet – if present and set to (say) "InvokeSet", PowerPro will process property assignments for the plugin. Suppose ExprLeft is any expression which returns a handle to the plugin, then

ExprLeft.prop(params) = ExprRight

results in the plugin call

Plugin.InvokeSet(ExprLeft, "prop", params, ExprRight)

Note that PowerPro will call localfree for all handles in ExprLeft, so the plugin should avoid aliasing handles. That is, if an plugin calls creates a new handle which is equal to an existing one, the plugin should allocate new storage or table slots for the new handle, and not attempt to re-use the existing slot or storage.

To free storage or table slots allocated to handles stored in temporary variables, the **ReturningFreeableHandle** service must be called whenever a new handle is created in a service. PowerPro will call the freecallback routine for all returned handles marked by a call to ReturningFreeableHandle, except for handles that are assigned. (If the handle is assigned to a local variable, then it will be freed when the script which declared that local variable terminates.)

useful sendMessages

SendMessage(g_hwndPowerPro, WM_USER+501, 0, 0)

returns pointer to PowerPro services, ie ppsv.

SendMessage(g_hwndPowerPro, WM_USER+502, 0, 0)

returns PowerPro version as single integer eg 4405, or 0 for all versions before 4.4.05.

plugin memory

PowerPro normally does not free a loaded plug-in until PowerPro exits. This can be awkward when debugging. So you can force PowerPro to unload using FreeLibrary by using a service name of unload, eg plugin.unload.

To save memory if multiple plugins are loaded, consider using the dll msvcrt.dll of standard dll services by compiling with library MSVCRT.LIB and link options /nodefaultlib:"libcmt". Plugins written in C should normally be 10K or less; if not, then you may not be using memory efficiently by using dlls instead of statically linked libraries.

For VC6, try

```
#pragma comment(linker, "/opt:NOWIN98")
```

In VC7, accomplish the same thing via

Properties\Linker\Optimization\OptimizeForWindows98.

It aligns the program's file sections at 512 byte boundaries instead of 4KB.

sending commands from other threads or processes

You should normally use (ppsv->RunCmd) to send commands to PowerPro. But if you have created a new thread in your plugin, or are running from a different process, then RunCmd is not safe since PowerPro is not threadsafe internally.

There are two other ways to send commands back to PowerPro: One is to create a full command line, including a full path to PowerPro, and WinExec it. The other is to create the PowerPro command only (without the path to PowerPro) and use a special SendMessage WM_COPYDATA. The advantage of the WM_COPYDATA is speed and the fact that the command is executed synchronously, that is you know it is done when the SendMessage returns.

(Note: PowerPro uses a WM_COPYDATA message, rather than a plain WM_USER+xxx, because this message is sometimes needed to cross process boundaries).

The following code shows how to use the WM_COPYDATA message by running *Script Run IAmDone

```
#define VAR_SIZE 532
char szCommand[VAR_SIZE+24];
char szName[VAR_SIZE+1];
COPYDATASTRUCT cd;

strcpy(szCommand, "*Script Run IAmDone ");
cd.dwData = 1;
cd.cbData = strlen(szCommand)+1;
cd.lpData=szCommand;
SendMessage(g_hwndPowerPro, WM_COPYDATA, 0, (LPARAM)&cd);
```

Here g_hwndPowerPro has been set to the PowerPro hidden control window:

```
g_hwndPowerPro = FindWindow("PowerProMain",NULL);
```