Windows PowerPro

Version 3.7

Feb 2002

PDF Created 1/24/03

Launch Help File

Launch Scripting Tutorial

- operator 109
!= ne operator 109
% operator 109
& and operator 109
* operator 109
*Bar Command 51
*Clip Command 62
*Desktop Command 65
*Exec CD 72
*Exec Commandline 69
*Exec Commands 67
*Exec mute 72
*Exec Prompt 72
*Exec ToFile 72
*Exec volumewav 72
*File Commands 74
*Format Command 84
*Keys Command 78
Example 80
*Menu Commands 87
*Menu Explorer 96
*Menu Folder
Demo 145
*Message Command 97
*ScreenSaver Command 97
*Shutdown Command 51, 97
*Timer Commands 45
*Wait Command 126
*Window Command 128
Specifying Actions 129
Specifying WindowID 131
*Windows Commands
Demo 144
/ operator 109
| or operator 109
+ operator 109
++ operator 109, 110
< lt operator 109
<= le operator 109
== eq operator 109
> gt operator 109
>= ge operator 109
2XExplorer 96
activewindow function 108
Alarms
Suspending 68
alt keyword 106
anywindow function 108
Arrays (single-dimension vectors) 178
assign function 110
Assign variable 98, 100
Autochanges
Media (Sounds, Wallpaper, Screensavers) 48
AutoIt DLL Plugin 166
Autorun when a Window Opens
Demo 146
Bar
Buttons from Files and Folders 29
Bars 51
Adding Buttons 57
Appearance 60
Autohide Bar 59
Changing Buttons 58
Click Main to Show Another 53
Creating 59
Deleting Buttons 58
Drag and Drop onto 56
Formatting 84
In or Beside Foreground Window 60
Invisible 23
Keyboard Access 56
Moving Floating Bars 58
Positoning 58
Removing 60
Screen Edge 53
Section Buttons 55
Using Subbars 54
Bars and Windows
Only open with Specific Applications 95
batterypercent keyword 107
browserDomain keyword 107
browserSubdomain keyword 107
browserURL keyword 107
Caps Lock 11
caption keyword 106
captionunder keyword 106
case function 110

Case string operator 111
CD Audio Player controls 72
cdcurtrack keyword 106
cdlasttrack keyword 106
Clip Filters 64
clip keyword 107
Clipboard
History Tracking 63
Manipulation, Tracking and Copying 62
cliptrackon keyword 107
Command
Entry Controls 49
Command Line Execution 136
Command List
Items 16
Properties 21, 27, 28, 53
Setup Dialog 24
Command Lists 15
Command Scripts 98
Commands
Autorun when Windows Open 31
PowerPro Built-in 50
Running Multiple 49
Conditional processing in Scripts 98, 101
Configuration 3
Advanced Options 6
GUI Control Options 11
Importing and Exporting Text Files 8
Setup Dialog 4, 24
Configuration Files
Sharing with other Users 159
Contact for Questions or Support 160
Copy file 74
cpu keyword 106
ctrl keyword 106
currentdir keyword 106
Date Calculations 71
Date Format 18
date keyword 106
Date Plugin 167
dayofweek keyword 106
dayofyear keyword 106
Debug Scripts 98
defaultprinter keyword 106
Delete file 74
Demonstrations and Samples 143
deskempty keyword 106
deskname keyword 106
desknum keyword 106
Desktop Command 65
Dialogs
Open/Save and Favorite Files and Folders 134
disk keyword 106
diskspace function 110
dunidle keyword 106
dunrate keyword 106
env function 108
Event Plugin 169
Examples 143
*Menu Folder 145
*Window Commands 144
Hot Keys and Mouse Actions 145
Keyboard Macros 146
Keys Commands 80
Menus and Context Menus 143
Running Commands when a Window First Opens 146
Sending Keys with *Keys 144
Subbars and Manually Shown Bars 143
Exec Commands 67
exefilename keyword 106
exefullpath keyword 106
Explorer
Customizing the Context Menu 32
Explorer List and View Settings 6
Explorer Windows 96
Expressions 104
refer to Expressions.Txt 115
Sample Script 115
FAQ 139
Favorite Folders and File/Open Save Dialogs 134
Features
Overview 1
Summary 141
File
Change Extension 74
Copy 74
Delete 74
Move 74
Random File useage 74, 75
Rename 74

Writng Entries To 72
File Plugin 172
filemenu function 108
Files
Favorites used in Dialogs 134
fill function 110
Fill string operator 111
Flags in Scripts 98, 104
Float Plugin 175
Folder
Working with Large Tree 94
Folders
Favorites used in Dialogs 134
Special Folders for *Menu Commands ..91
FormatDate and FormatTime 113
formatdate function 110
formattime function 110
Formatting Menus and Bars with *Format ..84
Frequently Asked Questions (with Answers) 139
gdi keyword 106
Handles of Windows 131
Hot Keys 35
Applicable to Specific Windows 38
Hotkeys and Mouse Actions
Demo 145
Icons
Saving and Restoring Desktop Postions ..65
Tray Icons from Other Programs 120
if function 110
ifelse function 110
index function 110
Index string operator 111
Info Displays 20
input function 108
inputcancel function 108
inputcolor keyword 107
inputdate keyword 107
inputdatetime keyword 107
inputdefault function 110
InputDialog 113
inputdialog function 110
inputfolder keyword 107
inputpath keyword 107
inputtext keyword 107
Items
Formatting 84
Join function 109, 110
Join string operator 111
Keyboard Macros 133, 141
Demo 146
keylogfile keyword 107
Keys
Caps Lock, Num Lock, and Scroll Lock 11
Hot Keys 35
Special Keys 77
Keystrokes
Logging 70
Keyword Values 106
Labels
Dynamic on Buttons and Menus 17
Labels in Scripts 98, 104
lastactivehandle keyword 107
lastautorunhandle keyword 107
lastclipname keyword 107
lastclippath keyword 107
lastidletime keyword 107
length function 108
Length string operator 111
License and Warranty 2
Log
Alarm 42
Timer 46
longdate keyword 106
Loops in Scripts 98, 102
Macros
Keyboard 133, 141
max function 110
mci function 108
Menu Folders
Appearance 89, 92
Menus
Displaying 87
Formatting 84
Menus and Context Menu
Demo 143
Message
Displaying with *Message 97
MessageBox 112, 173
messagebox function 110
min function 110
modem keyword 107

mounted function 108
Mouse
Automatically Moving the Mouse Cursor to a Dialog Button 4
Pressing Buttons Automatically 14
Scrolling 12, 13
Sending a Sequence of Clicks and Moves 85
Mouse Actions 35
mouseleft keyword 106
mousemiddle keyword 106
mouseright keyword 106
Move file 74
muted keyword 107
not function 108
Notes
Creating and Maintaining Reminders .148, 149, 150
Num Lock 11
paper keyword 107
Plugins 161, 162
Au (AutoIt DLL) 166
Date 167
Event 169
File 172
Float 175
RegEx 183
Vec 178
Win 181
pmem keyword 106
pprofolder keyword 106
pproversion keyword 106
processcount keyword 106
Purchasing PowerPro 138
Random Files 75
random function 108
readline function 111
recycleItems keyword 107
recycleSize keyword 107
RegEx Plugin 183
Regular Expressions 183
Reminders 148, 149, 150
remove function 110
Remove string operator 111
Rename file 74
replacechars function 110
ReplaceChars string operator 111
Resolution
Changing Screen 66
Resource Usage Displays 19
revindex function 110
RevIndex string operator 111
Run commandlist 98
RunFile 98
saver keyword 107
saveractive keyword 107
Scheduling Commands 3, 39, 41, 42, 48
Scheduling Commands to run when system is idle 42
Screen Saver
Accessing with *ScreenSaver 97
Autochanging 48
Script
Commands 98
Keywords 106
PPST Tutorial 2, 98
Running from a file 117
Sample 115
Wait Command 126
Scroll Lock 11
select function 110
Select string operator 111
Sending Frequently used Folder Names to Dialogs 82
Sending Keys
Demo 144
Selecting Keys from a Menu 81
to Other Windows 77, 78
to Programs when Started 81
shift keyword 106
shortdate keyword 106
Shutdown Windows or PowerPro 51, 97
Skins 150, 151, 152, 153
Creating 152
Using 150
Sound Volume 72
Sounds
Autochanging 48
Stiletto 137
String
Escape characters in 105
Operators 111
Subbarname keyword 106

Subbars and Manually Shown Bars
Demo....143
threadcount keyword....106
Time Format....18
time keyword....106
timer function....108
Timers....3, 44
timesec keyword....106
timezone keyword....106
Tiny Type and Run Dialog....69
Tool Tips....23
Tray
Buttons....29
Icons....28, 120
Minimize Window to....5
uptime keyword....106
user keyword....106
validpath function....108
Values....105
Variables....100
vdeskempty function....108
vdeskhaswindow function....111
Vec Plugin....178
Vectors....178
Virtual Desktop
Demo....147
Virtual Desktop Sample....147
Virtual Desktops....121, 123, 124, 125
visiblewindow function....108
volume keyword....107
Wait Command....126
Wallpaper....121
Autochanging....48
win keyword....106
Win Plugin....181
Window
ActiveWindow Operator....114
AnyWindow Operator....114
Information (e.g., Position, Size, State, Class, Caption, Handle)....14
Manipulate Running Program's....128
Switching Active with Buttons....26
VisibleWindow Operator....114
Window Operator....114
Window Command....128
window function....111
Window Handles....131
Windows
Autohiding....10
Hiding....30
Resources Display....19
Word Exclusions....30
word function....110
xmouse keyword....107
xscreen keyword....107
xtime keyword....106
Yes/No Prompts....72
ymouse keyword....107
yscreen keyword....107

Table of Contents

Overview ... 1
PowerPro License and Warranty ... 2
Configuring PowerPro ... 3
Setup Dialog ... 4
  Automatically Moving the Mouse Cursor to a Dialog Button ... 4
  Minimizing a Window to the Tray ... 5
  Changing Explorer List and View Settings ... 6
  Advanced configuration options ... 6
  Exporting and Importing Configurations Using Text Files ... 8
  Automatically Hiding Windows ... 10
GUI Control Configuration Options ... 11
  Caps Lock, Num Lock, and Scroll Lock ... 12
  Scrolling with Mouse Movements ... 12
  Manual Scrolling with Mouse ... 12
  Automatic Scrolling with the Mouse ... 13
  Automatically Pressing Buttons when Mouse is Stopped over Them ... 14
  See Mouse Cursor Position and Window Information ... 14
Command Lists Dialog ... 15
  Configuring a Command List Item ... 16
  Special Labels for Buttons ... 17
  Date and Time Format ... 18
  Resource Usage Displays ... 19
  Other Displays ... 20
  Command List Properties ... 21
  Tool Tip Setup ... 23
  Working with Invisible Bars ... 23
  Command List Setup Dialog ... 24
  Active Window Switching with Buttons ... 26
  Working with Tray Icons ... 28
  Creating Bar Buttons from the Files and Subfolders of a Folder ... 29
  Tray Icon Buttons ... 29
  Omitting Windows and Words from Active Window Lists ... 30
  Hiding Windows ... 30
  Automatically Running Commands when Windows Open ... 31
  Adding Entries to the Explorer Right Click Context Menu ... 32
    Purpose ... 32
    Configuration ... 32
    Displaying Different Items for Different Files ... 33
    Example ... 33
Hot Keys and Mouse Action Commands ... 35
  Hot Key and Mouse Action Setup ... 35
  Entering Hot Key/Mouse Action Information ... 36
  Hot Key/Mouse Action Explanations ... 37
  Window-Specific Hot Keys ... 38
Scheduler ... 39
  Entering Information for A Scheduled Command ... 40
  Scheduler Setup ... 41

Running Programs After the System is Idle for a Specified Time 42
Alarm Log 42
Timers 44
Setting Timers and Associated Commands 45
Setting Timer Value and State 46
Timer Logs 46
Media Dialog 48
PowerPro Sounds 48
Command Entry Controls 49
Running Multiple Commands 49
PowerPro Built-In Commands 50
Bars and the *Bar Command 51
Screen Edge Positions 53
Showing Other Bars when you Click a Main Bar 53
Using Subbars to Display Different Parts of Bars 54
The Section/Subbar Approach to Configuration 55
Drag and Drop onto the PowerPro Button Bar 56
Using the keyboard to access the button bar 56
Adding a Button 57
Changing a Button 58
Deleting a Button 58
Moving Bar 58
Positioning the Bar 58
Creating an Autohide Bar 59
Creating a New Bar 59
Removing a Bar 60
Bar Look 60
Positioning PowerPro Bars in or beside the Foreground Window 60
Clipboard Manipulation, Tracking and Copying 62
Clipboard History Tracking 63
Clip Filters 64
Desktop Command 65
Saving and Restoring Desktop Icon Positions 65
Changing Screen Display Resolution 66
Using *Exec 67
Suspending Alarms 68
Tiny Type and Run Dialog 69
Logging Keystrokes 70
Date/Calendar Calculations and Display 71
Prompting for Yes/No Information 72
Sound Volume 72
CD Functions: 72
Writing Entries to a File 72
PowerPro *File Commands 74
Working with a Randomly Selected File 75
Sending Keys to Other Windows 77
Specifying the Window to Receive the Keys 78
Specifying the Keys to be Sent using *Keys 78
Examples of Keys Commands 80
Sending Keys to Programs When They Are Started 81

Selecting some Keys to be Sent from a Menu 81
Creating Menus or Bars of Favorite Folders using *Keys 82
Formatting Menus and Bars with *Format 84
Changing the Look of an Item with *Format Item 84
Sending a Sequence of Mouse Clicks and Moves 85
Displaying Menus with *Menu 87
*Menu Folder 88
Format of *Menu Folder 89
Special Folders for *Menu Folder 91
Entering Format Information for Folder Contents Command 92
Using *Folder Contents Menu with a Large Folder Tree 94
Window-Specific Bar and Menu Contents 95
Working with Explorer Windows 96
Displaying a Message with *Message 97
Accessing the Screen Saver with *ScreenSaver 97
Shutdown Windows or PowerPro 97
Command Scripts 98
Keyword Values 106
MessageBox 112
InputDialog 113
FormatDate and FormatTime 113
Help on Expressions 115
Sample Script 115
Running a Script from a File 117
Working with Tray Icons from Other Programs 120
Training PowerPro to Recognize Tray Icons from Other Programs 120
Changing the Wallpaper with *Wallpaper 121
Virtual Desktops 121
Explanation of Virtual Desktop Menu 123
Virtual Desktop Setup 124
Initializing Desktops Using the Configuration Dialog 125
Wait Command 126
Manipulating Windows of Running Programs 128
Specifying the Action for *Window Command 129
Specifying the WindowID for the*Window Command 131
Window Handles 131
Keyboard Macros 133
Favorite Folders and File/Open Save Dialogs 134
Windows PowerPro Command Line 136
Information for Stiletto Users 137
Purchasing PowerPro 138
Frequently Asked Questions (with Answers) 139
Power and Flexibility of PowerPro 141
Demonstrations and Samples 143
Demonstration of Menus and Context Menus 143
Demonstration of Subbars and Manually Shown Bars 143
Demonstration of *Window Commands 144
Demonstration of Sending Keys with *Keys 144
Demonstration of Hot Keys and Mouse Actions 145
Demonstration of *Menu Folder 145

Demonstration of Keyboard Macros 146
Demonstration of Running Commands when a Window First Opens 146
Virtual Desktop Sample 147
Notes 148
Skins 150
Using Skins 150
Creating Skins 152
Sharing PowerPro Configurations 159
Contact for Questions or Support 160
Plugins 161
General Information 161
How to Program Plugins 162
Sending Commands to PowerPro 164
Plugin Memory 165
AU PLUGIN (Version 2002 12 22) 166
DATE PLUGIN (Version 2002 10 29) 167
EVENT PLUGIN (Version 2003 01 01) 169
FILE PLUGIN (Version 2003 01 19) 172
FLOAT PLUGIN (Version 2002 10 28) 175
Sample PLUGIN (Version 2002 12 22) 177
VEC PLUGIN (Version 2002 12 28) 178
WIN PLUGIN (Version 2002 12 22) 181
RegEx Plugin (12/26/02 Version) 183

Overview

Windows PowerPro incorporates these features:

- Any number of small-footprint button bars (e.g. fit over title bar of maximized window).
- Use of any mouse button to launch commands.
- Drag and drop files to start commands.
- Up to 95 user-configurable command menus with submenus.
- Floating button bar, choice of many resolution-independent standard positions, or place in active window caption.
- Direct access to start menu
- Hot key, tap key, and mouse action activation of commands.
- Hot keys which depend on the active program.
- Activate commands by mouse actions including press and hold, click caption, horizontal/vertical movements.
- Show menus from hot keys, mouse actions.
- Menu subsections and button bars which are displayed only if a given program is active.
- Display menus built dynamically from folder contents.
- Switch to or close any active task, from a button bar or a menu.
- Text label, icon, clock, date, timer, or resource display on any button.
- Built-in commands for screen saver, windows exit/restart, browsing and running files (with history), moving the button bar, playing sounds, and others.
- Control of Caps Lock/Shift and Scroll Lock.
- Scrolling with the middle mouse.
- Tray minimization.
- Virtual desktops.
- Alarms, regular chimes, and scheduled activation/termination of commands.
- Wallpaper display and switcher/randomizer.
- Save and restore desktop icon positions.
- Screen saver switcher/randomizer.
- Randomization and testing of system and application sounds.
- Send a sequence of keys to a running program or to a program that you start with PowerPro.
- Tool tip (balloon) help to display the commands for any button.

Windows PowerPro is intended to supplement the Win95/98/NT 4/5 shell by providing quick, minimal-mouse click access to your most used commands while taking up little desktop space, and to provide utilities related to Windows start-up and time, with one consistent interface.

PowerPro License and Warranty

The PowerPro scripting tutorial in PPST 1.00.chm was developed by Alexander Cicovic and the PPTF.

The PowerPro icon was created by Jonas Hjortland.

THIS SOFTWARE IS DISTRIBUTED "AS IS," WITHOUT WARRANTY AS TO PERFORMANCE OF 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.

The 32 bit version of Windows PowerPro is free.

Windows PowerPro may not be sold nor be used in any profit-oriented endeavor without the express written permission of the author with the exception that Windows PowerPro may be distributed freely via media intended to make shareware available to the public for trial. All files, including the Windows PowerPro program, DLL, help file, readme file, license file, and all others in the Windows PowerPro zip file, must be included.

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.

Configuring PowerPro

Purpose

You configure Windows PowerPro buttons, menu contents, media, hot keys, and alarms with the Configure Windows PowerPro set of tabbed dialogs.

You start this dialog by right-clicking anywhere on the Windows PowerPro bar with the Ctrl key pressed, by pressing and holding down any button on a bar, by running the Windows PowerPro configure program pproconf.exe from the Accessories section of your Start menu, or by associating the *Configure built-in command with a button, menu item, or hot key.

Configuration

The command 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.

Setup sets functions which customize your Windows interface and allow you to enter the code you obtained when you purchase Windows PowerPro.

GUI Control sets functions which customize your Windows interface.

Command Lists allows you to change the contents of command lists.

Key/Mouse allows you to assign commands to hot keys, mouse actions, or screen corners.

Scheduler allows you to add or change alarms.

Timers controls the value and commands of timers.

Media controls the sounds, wallpaper, and screen saver allows you to specify how Windows PowerPro should automatically change them.

The *Configure command lets you use the action AddNewReminder to add a new scheduled message.

Windows PowerPro can position the tabbed configuration dialog as always-on-top; you can change this with advanced dialog.

Setup Dialog

Purpose

The Setup dialog allows you to move the cursor to the default button of new dialogs and optionally press the button; to automatically tray minimize applications when they are minimized, to automatically hide new windows when they are opened, to track explorer windows as you open them, and to force explorer windows to a given view. You can also access the Registration, virtual desktop setup , advanced setup dialogs, and import/export dialogs. If PowerPro is running, you can save and restore desktop icon positions with the corresponding buttons. You can also show all hidden bars with "Show All Bars".

Configuration

This dialog is activated by clicking on the "Setup" tab of the configuration dialog.

Cursor to Default Button
You can have Windows PowerPro automatically move the mouse cursor to the default buttons of a dialog.

Automatic Tray Min
Enter a caption list windows to be minimized to the tray instead of the task bar.

Automatic Hide
Enter a caption list windows to be minimized to the hidden when created.

Automatic Tray Min
Enter a caption list windows to be minimized to the tray instead of the task bar. 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.

Explorer Windows
You can specify that Windows PowerPro track Explorer Windows for use with the *Menu Explorer command.

You can specify that Windows PowerPro should force settings for Explorer view and arrangement.

Automatically Moving the Mouse Cursor to a Dialog Button

Check the "Cursor to default button" checkbox on the Setup dialog to have Windows PowerPro automatically move the mouse cursor to default button on a dialog.

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 Windows PowerPro automatically push the default button by including the caption of the window in the "Press default button" edit box at the bottom of the dialog. You need not enter the whole caption: enter xxx* for captions starting with xxx, *yyy for captions ending in yyy, and *zzz* for captions containing zzz anywhere. Windows 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, Windows PowerPro only moves mouse cursor and presses the default button for captions specified in the "Press default button" edit box

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.

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 copy of the five previous configurations are kept is "!Previous auto backup of …". You can restore these backups using buttons on the Setup dialog. This button can also be used to restore the results of the most recent use of the Apply button.

Minimizing a Window to the Tray

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, Windows 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, bar button, or menu item; set the program to start initially as tray minned on the command entry controls; or place the caption or exe file name in the "Automatically minimize to tray" edit box on the Setup configuration dialog.

You can replace normal minimization to the task bar by minimization to the tray by using the 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 also select windows to be minimized by using =filename to work with the program filename.exe (omit path and .exe).

A convenient way to manually access tray minimizing is to assign the *Window traymin command to a hot key corresponding to right-clicking the minimize box.

Normally, Windows 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 name to match the caption of the window to be tray minimized, and set the

command list item 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 to match all programs called ExeName. Finally, use the drop down box on the command list setup dialog to select the command list.

If you are using virtual desktops, showing a tray icon will also switch to the virtual desktop it was part of when it was tray-minimized.

Changing Explorer List and View Settings

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

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 where Explorer also stores a setting, but if you hold down the shift key while opening the new window, Windows PowerPro will not override the Explorer settings.

For a convenient way to change the settings for Explorer windows while you are working with them, send keys to the active window (of course, you can use the tool bar as well). For example,

Command *Keys

Parameter "a-v i d"

sends **Alt-V**, then **i**, then **d** to the active window which would set date sort arrangement for Explorer. You could attach the above command to a hot key or a menu attached to a hot key.

You can also use 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 Commands: *Keys {to **attach} "a-v g"

This 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 **+**attach** tells Windows PowerPro to wait until a window with caption ending in **attach** appears before sending the keys.

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

If you send keys to Explorer when it is launched from Windows PowerPro, the settings will replace any settings forced by the Setup dialog.

Advanced configuration options

configuration dialog which lets you set many less-used Windows PowerPro options. There are four tabs: configuration lets you change options controlling the configuration dialog, Other

controls miscellaneous features, characters let you define or change special characters used in commands, and limits set timer and count limits.

Configuration

Include desk 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 tasking automatic backups of configuratino 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 *Format Items.
Config on top	Check to display configuration dialog 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 command list click	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.

Other

Show all windows	If checked, all windows for a task are shown whenever any window for that task is activated.
Show tray min	Shows windows which PowerPro has tray minimized if they are activated, eg 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 (eg 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 (eg ctrl-tab)
Allow only alphas	Allows only alphanumerics in file names formed from captured clipboard items
User internal folder	Check to use folder icon in PowerPro.exe file; uncheck uses system folder icon (which may fail in NT).
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 restart	If checked, PowerPro will never prompt for a restart after execution of *Exec Resolution.
Parameter dialog position	Select starting position for dialogs displayed by input and inputdialog operators

Characters

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 with &(expression)
Quote must precede	Affects how the quote is used with expression char and prompt char. If checked, then the quote must precede either for the script insertion or prompting to occur. If unchecked, then a preceding quote stops the script insertion and prompting.
Process expression	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.
Reverse results	Use at the start of hot key targets in window caption matching strings to select windows which do not match the following strings.
File name character	Used with open/save file tracking, this is the underlined character in the title beside the file name edit box.
Command separator	Obsolete. Use this character to separate multiple commands; leave blank for preferred {enter} (start each command on new line).
Clipboard character	Obsolete. Leave blank and instead use &(clip) instead, where & is expression character, to insert the first line of the clipboard.
Prompt character	Obsolete. Leave blank and instead use &(input "title") or &(filemenu "c:\path\file.txt") instead to prompt for input, where & is expression character.
Allow single character variable names …	Meant for compatibility with scripts created for older versions of PowerPro. You should leave this unchecked, which means than the expression insertion character must always be followed by (to be processed.

Limits

Max menu row	Sets maximum number of rows in *Menu Folder and *Menu Explorer
Explorer Windows	Sets maximum number of entries in *Menu Explorer; you may need to restart PowerPro if you change this value.
Button held down	Button configuration dialog shown after button held down for this number of milliseconds; set to a large number to disable.
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.
Marker Window	Sets the number of pixels used in the marker for hidden bars.
Time mouse hovers	hover time in milliseconds for cases when bars are set to activate left click after mouse hovers over button.
Send keys delay	Set delay in milliseconds for first key and subsequent keys
Hook disable	Set non-zero to disable internal hooks. Only needed in rare configurations.

Exporting and Importing Configurations Using Text Files

You can use the Import from Text and Export to Text buttons on the Setup dialog to write parts of the configuration file to a text file or to read text back into a configuration file. You might find this useful to make mass changes to a configuration. The configuration information that can be read or written is limited to command lists, scheduled events, timers, and hot keys.

Exporting to a text file produces a file with this format:

A line [*Hot] signals the start of the hot keys. Each hot key is then written on one line as follows:
hotkey "target" switches command!`work
where
hotkey is an integer giving the hot or mouse action
target is the target, always in double quotes
switches is an integer encoding the how to start, switch to, and on top settings
command is the command text; if the exe name contains blanks, it is followed by a þ
!` is a separator between the command and work strings
work is the starting folder or formatting keywords for certain built-in commands

Each command list in the text file is started by a line [name], where name is the command list name. There are five lines for each command list entry:
"name" icon*tooltip
L switches command!`work
M switches command!`work
R switches command!`work
F flag width textcolor backgroundcolor

where
"name" is the item name, always in quotes
icon gives the number and file for the icon
* separates the icon from the tool tip
tooltip is the tool tip text
L starts the left command; see hot key for command format
M starts the middle command; see hot key for command format
R starts the right command; see hot key for command format
F gives the own text and own background flags, the width, and the text and background colors as RGB integers.

When you import files, you only need to include the command lists you want to import. If a command list of the same name already exists, it is overwritten. The [*hot] section is also optional. If present, it overwrites existing hot keys. Scheduling is cumulative, however; imported events are added to existing events (but duplicates are ignored).

The import file must follow the above format. However, for command lists, you only need to include the L, M, R, F values where the command is not (none) or the F values are not all zero.

Command Line Operations

If you want to read text into an existing or empty configuration file, start the pproconf.exe program as follows:

c:\yourpath\pproconf.exe c:\anypath\newconfig.pcf

If newconfig.pcf exists, it is read and then can be modified by importing text. If it does not exist, Windows PowerPro will create an empty configuration (after prompting to confirm).

You can import or export using only the command line.

c:\yourpath\pproconf.exe /i:"c:\path\import.txt"

imports into the standard configuration. Note the colon (:) and double quotes which follow the /i. Both must be included and without spaces. You must put the full path to the import text file within the double quotes. Also, the configuration program pproconf.exe cannot already be open.

To export the whole configuration, use

c:\yourpath\pproconf.exe /x:"c:\path\exportall.txt"

To export just scheduled events, use

c:\yourpath\pproconf.exe /s:"c:\path\exportsched.txt"

To export just command lists using the Left command only format, use

c:\yourpath\pproconf.exe /L:"c:\path\exportleft.txt"

The character following the / is the letter L.

You can precede the / by a .pcf path if you are not using the standard pproconf.pcf in order to import or export from another .pcf file.

Automatically Hiding Windows

You can specify that Windows 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 Setup dialog. For example,

HideMe

in this edit box will cause any windows with a caption containingHideMe to be hidden.

GUI Control Configuration Options

Purpose

This dialog configures many options related to the way you interact with Windows.

Configuration

The GUI Control dialog is selected by clicking on the "GUI Control" tab from the Configuring command.

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 Windows PowerPro should show more of windows which it activates and which are mainly off the screen.

You can specify that Windows 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 Windows 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 Windows PowerPro should enlarge the file list windows use 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 examople, setting this to 22 produces a button consisting solely of the icon.

You can use the middle mouse button and mouse movement to scroll windows.

You can indicate that Windows 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 Windows 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 Windows 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 Windows PowerPro automatically track text pasted to clipboards.

You can specify the sort order for the task bar. Check "Sort taskbar" to sort entries alphabetically. For further control, you can specify the exact order of windows by listing their captions. Use xxx* to match any captions starting with xxx, *xxx to match any caption ending in xxx, and *xxx* to match any caption containing xxx. Separate entries in the edit box with commas. Put a dash (-) in front of an entry to force it to the right end of task bar. For example, if you put *notepad, *agent*, -*explor* in the edit box, then all windows with captions ending in Notepad would be placed first on the task bar, followed by all windows with agent anywhere in their caption, followed by all other windows in alphabetical order, followed by all windows with explor in the caption.

Caps Lock, Num Lock, and Scroll 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 Windows PowerPro**.** Or you can disable the caps lock key completely.

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.

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 setting the set the key permanently off; gray check to set the key permanently on.

Scrolling with Mouse Movements

Purpose

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.

Manual Scrolling with Mouse

You can start manual scrolling in one of two ways: by attaching a command to a hot key/mouse action or by the middle mouse button.

To start scrolling with a hot key, attach this command to the key:

Command *Exec

Parameter scroll

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.

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

To scroll a window, activate the *Exec scroll hot key or hold down middle mouse and move in 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 Scroll interval advanced option.

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 Windows PowerPro scrolling for these or any 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 MS IE windows.

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. The slowest scrolling speed is set by the scroll interval on the advanced dialog.

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

You can optionally set the scroll speed by including 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 Windows PowerPro command otherwise. To do this, define the hot key twice, and use the target window feature on the second 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	

Automatically Pressing Buttons when Mouse is Stopped over Them

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

Use the check box on the GUI Control tab to do this. You can also set the stop time with the spin box. If you want Windows PowerPro to automatically select standard menu items too, gray check the box.

By default, Windows 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. Windows 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 use the "except" edit box to specify a caption list of windows where the automatic press is not to occur.

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

*Exec

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 activate the hot key. If the window type is not currently one that is automatically pressed, Windows PowerPro will add it to its list. If it is one that is automatically pressed, Windows PowerPro will remove it. In both cases Windows PowerPro notifies you of the results with a message box.

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

See Mouse Cursor Position and Window Information

You can have Windows PowerPro display a small window with the mouse screen position and the size and position 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 (ie 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:

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)

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

total window size: width x height

client window size and aspect ratio

window caption

window class

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.

The display window uses the same colors and font as the tool tip window.

If the mouse cursor is over an Edit box, the contents of that box are shown are the caption. This can be useful to see password fields.

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 option "Show Window Contents While Dragging" activated; this option is available in NT4 and Win 98 or in Win95 with MS Plus!.

Command Lists Dialog

Purpose

The command list dialog is used to create the list of commands for:

display as a menu

display as a bar

displaying tray icons

running as a script

creating keyboard macros

creating commands to run when windows first open

There are two steps involved with using these features: use the Command Lists dialog to create the list of commands and then use a command like *Menu show or *Bar show to display the command list as a menu or a bar.

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.

Select an item and use the controls at the right of the dialog to add or 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, Windows PowerPro takes note of which command (left, right, middle) you clicked on, and starts the command edit dialog with this command.

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.

By dragging them from Explorer or a desktop icon or the start menu itself (Win98 only): Use "Programs" to open an explorer window or use any other explorer window and drag/drop to set the left command, icon, and label from the dropped item. 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.

Press Apply to immediately test changes using Windows PowerPro. If you cancel the configuration dialog, applied changes will be removed.

If you want Windows PowerPro to automatically display the command list as a bar, check the "Show as Bar" box at the right of the dialog. You can also use the *Bar command to control display of bars.

Configuring a Command List Item

Name: To set the item label, type a label of up to 127 characters into the label combo box or select a special label using the info… button. Leave the label edit control blank to omit a label (eg 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 from the keyboard.

Tool tip: Enter tool tip text. You can create multi-line tool tips with the separating character specified on the tooltip 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.

Background and text color: Check the box and use the Set button.

Hide: check to hide this item when bar or menu displayed (does not apply to *Format commands).

Hide icon: Check to hide icon. Can be set programmatically with *Format item.

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 over button.

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 default height or for height set on command list properties; else set button height in pixels.

Icon: 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. Or browse for an icon file with the **...** button. Use the spin box to choose a specific icon from a file.

Commands: Items can have three associated commands: use the left/middle/right buttons to access each set of command entry controls for each item. Use the find button or the command dropdown list to set a program or enter a builtin command.

Special Labels for Buttons

Purpose

You can display dynamic text to monitor time, date, resources on your system, or other information. This text can be displayed on button labels, as menu item labels, on tool tips for bars, menus or tray icons, or in the system tray replacing the clock.

Configuration

Set the start of the item label or tool tip to *Info and then use the Info button in the top left 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 three types of keywords:

time/date

resources

other , such as clipboard contents, virtual desktop name, variable contents, free disk space, timer value

You can also put arbitrary 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 (eg MMMM, HH).

For bar labels and the system clock, 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).

Tray icon text 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.

*Info displays on bar labels 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 Format

Use the *Info labels of the command item dialog to specify date or time display. Use these sequences of letters. The case of the letters is important:

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
h	Hours with no leading zero for single-digit hours; 12-hour clock
time	Time format (as set on Control Panel|regional)

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

Resource Usage Displays

Windows PowerPro can display Windows resources on bar button or menu item labels or tool tips by using the *Info entry from the name field of the Command List Item dialog.

Use any combination of the following keywords in any order to display the resources. Keywords must be in lower case. Keywords must be in lower case.

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	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 (95/98/2000)	Percent of battery power remaining; 255% means no information available
allbattery (95/98/2000)	The 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.

Any other text in the *Info item label is displayed without change. For example:

*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.

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 gid resources separated by a slask, them physcial memory followed by "k"; sample output would be 78/80 123K

Other Displays

Other *Info button labels (all keywords must be in lower case)

clip n	Shows text on clipboard; use n to limit to first n characters.
var v	Show variable v (only variables a-z can be used, not x0-x9)
deskname	Current desktop name.
desknum	Current desktop number 1-9.
disk x	Free space for disk x.
timer x	Value of timer x in hours and minutes; label can be any text.
timerdays x	Value of timer x in days, hours, and minutes; label can be any text.
timersec x	Value of timer x in hours, minutes, seconds; lab can be any text.
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
defaultprinter	name of default printer.
keylog	X if logging keys, empty otherwise.
keylogfile	name of currently open key logging file, empty otherwise.
expr (ex)	Evaluates the expression ex, e.g. expr (dunrate/10).

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. One appropriate use of expr is a more convenient replacement of var v where v is set by a complex expression in a script.

Command List Properties

Purpose

The Command List Properties dialog sets the menu/bar position for a command list, and the color, visibility, base height. You can configure a bar to be invisible.

Configuration

To access the dialog, select the command list to be configured in the drop down 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.

Bar and Menu Properties

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 (menus only display 16x16 or 32x32 icons). (Note: PowerPro only reads true 48x48 and 64x64 icons from .ico files; from other files it uses stretched versions of the 32x32 icon).

You can also set a background color, a text color, and a font for this command list. You can set the maximum number of characters of text to be displayed (set to 127 to display all text specified in the item label). You can also access tool tip setup.

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.

The hide after feature shown under Bar properties can also be used to close a menu after the mouse cursor is off of it for the specified time.

For both menus and bars, you can set a .bmp file to be tiled for bar background. Use * to have PowerPro set the background bitmap to the wallpaper under the bar or menu. For menus with backgrounds, 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 bar position to eg caption to enable the offset field for entry).

Bar Properties

These apply when the command list is displayed as a bar.

Use "Position" to set the bar position.

Use "height" to set button height in pixels. All buttons must have same height. A zero height uses the default but you may have to manually set height for vertical or rotated text.

Use "hide after" to set whether PowerPro should hide the bar or menu -- see visibility (hiding also applies to both bars and lists shown as menus).

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 the *Bar showcommand. Note that "showing" bars applies to two cases: invisible bars and non-top most bars (which are hidden by other windows).

For Win98/2000, you can choose how to use slide animation when hiding and showing the bar. You must also check "Use slide animation" on command list|setup.

Use "marker" to create a small marker strip when bars are hidden.

Use "offsets" to specify offsets for positioning in or near the active window and screen edge positions.

You can check the following format options for a bar

border	check to draw a thin black border around the bar
3d Frame	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.
no flicker:	Eliminates bar flicker (but uses more memory to draw bar).
vertical bar	buttons are aligned beneath one another
hover clicks	if mouse hovers over button, that button is left clicked; set hover time on advancedsetup
flat	check for flat bar; gray check to avoid button border when mouse passes over
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 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).
right icon	icons are shown on the right of text
gradient	bar color varied specified number of steps with specified background color in middle of color range
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 (eg) Arial using "Own Font"
text under	check to display text under the icon on bar buttons
center text	centers text label

You can also ctrl+right click the bar to get the configuration menu and set 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 left 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

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; see below.

Folder Buttons

Specify a folder whose entries will appear as buttons at end of bar.

Tool Tip Setup

Use the Tool Tip setup button on the Bar and Menu Properties or the Command List Setup dialogs to access a dialog which controls the look of tool tips. With this dialog, you can set:

Background and text color for tool tips.

A character used to create multi-line tool tips. Whenever this character is found in tool tip text, a new line is started. For ecample, you could create one line per command for buttons or menu items which have left, middle, and right commands.

Whether slide animation is used to show the tooltip (Win98/2000 only).

Whether PowerPro will draw tool tips for its tray icons; this allows multi-line tool tips.

The delay until tool tips appear for bars.

The delay until tool tips appear for *Menu Folder and *Window menus (zero means no tool tip).

Active Buttons

Set the number of active 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.

Working with Invisible Bars

The Bar Properties configure dialog contains options for hiding Windows PowerPro.

Set the time for autohide 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 this screen edge and holding the mouse at the edge for the time specified at the Hot Key setup dialog 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".

If you want a small marker window to show at the screen edge of a hidden bar, check "Show Marker Window".

For Win98/2000, you can choose how to use slide animation when hiding and showing the bar. You must also check "Use slide animation" on command list|setup.

You can also show a bar by assigning the *Bar Show or *Bar ToMouse commands to a hotkey, another bar button, or menu item and then executing this command.

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 from the accessories pproconf.exe program and reset the bar properties to avoid hiding.

Command List Setup Dialog

Purpose

This dialog controls general appearance of command lists displayed as bars or menus. It is also used to specify the command lists used for tray icon, auto run, active button icon, tray minimized icons.

Configuration

The Menu and Bar Setup 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 menus only, bars only, both bars and menus, and for command lists being used for special purposes such as tray icons.

All Menus

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, Windows PowerPro will force the mouse cursor over newly opened menus; especially useful if you autohide menus after the cursor is moved off of them.

Right selects command in menus: if checked, selecting a menu item with right will execute the corresponding 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

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 extra space used to draw this border.

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.

Hide caption bars if no active windows: check to hide bars positioned in the captions when there is no active window.

Clicking active button of foreground minimizes it: check to PowerPro minimize the window of the active button when clicked and it is the foreground 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.

Dynamically update text on active buttons: check to have text 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 Windows systems.

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.

Move Bars to edge: If checked, PowerPro automatically moves screen edge bar positions to the appropriate screen edge.

Drag to move: If checked, left click and drag any button to move bar; if unchecked, you must ctrl-left click and drag to move.

Use slide animation: Bars are shown and hidden using a slide animation (Win98/2000 only).

You can set the marker window size in pixels. Marker windows can optionally be displayed for hidden bars.

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

You can set the default 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 list of captions for hidden windows to appear on active buttons.

Default background If set, this background 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 Command Lists

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, eg updating bar item format. 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 tooltips: If checked, PowerPro will draw tool tips for its tray icons; this allows multi-line tool tips.

Using the drop down boxes, you can specify that the commands on a command list should be used for:

to display tray icons

to provide a command list to control commands to be run when a specified window first opens

to provide a command list control the icons assigned to tray minimized windows

to provide a command list control the icons assigned to active buttons

Active Window Switching with Buttons

Purpose

If you want to configure your own version of the task bar, you can create Windows PowerPro buttons which automatically track each top-level window on your system so you can quickly switch to a new active window by left clicking a button and activate a menu to close or minimize by right clicking the button. This is an alternative to the list of active windows menu item which can also be used for active window switching. The advantages of buttons are that all active windows are immediately visible on the button bar and that switching is done with a single click. The disadvantage of the button bar is that extra desktop space is used.

Configuration

You set up active task buttons with the Command List Properties dialog. To display active buttons, set the maximum number of buttons to a number greater than zero. Windows PowerPro will display a button for each active task. You can have active buttons along with 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 for Setup" on the active button tab.

Normally both button and text will be displayed on an active button. To display the icon only, check the "Icons Only" check box.

To select which windows appear as active buttons, enter captions in the edit box of Active Button properties.

To ensure that active button text is dynamically updated, check "Dynamically update active button text" on Command List|Setup|All Bars.

You can control the look of active buttons and the commands for all active buttons by checking "Last item 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 for on the "Last item for setup" button; otherwise the button width will be set to the width of an icon.

Further Information

When displaying the icon for a window on a button, Windows PowerPro normally uses the window class icon. You can specify your own icons for programs by creating a special command list and selecting this command list with the icon menu drop down on the Command List Setup dialog. Create one entry in the list for each program with an icon that you want to specify. Set the list item name to =exename, where exename is the name of the program exe file without the path and without the .exe extension (eg =winword for Microsoft Word). Leave the item commands set to (none). Set the item icon to the icon you wish to use for all windows from exename.

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 cause activated task windows to be centered using a switch on the GUI Control dialog. Gray check to center the mouse cursor as well.

You can control whether or not active buttons are sorted using Command List|Setup. If sorted, you can control the sort order of active button by using the "Sort order for active buttons" edit box on the command list setup. Enter a list of caption entries separated by commas and using *xxx* for captions containing xxx, xxx* for captions starting with xxx, *xxx for captions ending in xxx. Then any active buttons with captions matching the first entry will sort to the start of the bar, followed by any matching the second, and so on; captions not matched appear at the end. For example, if this field was set to *explor*,*agent* then any windows containing explor in the caption would start, followed by any containing agent, followed by all others.

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 Advanced dialog.

Working with Tray Icons

(Win98/95 only)

You can access all icons in the system tray with a *Menu Tray. or with active buttons. For the menu or a button, left or right click to activate the correspond tray icon function; middle click (or left+shfit) to access the double click function of the tray icon.

The text on the menu or button is set from the tool tip of the tray icon.

Before using this feature, you must install tray support. To do so, activate the configuration dialog, for example by ctrl-right clicking a bar and selecting configure, select the setup tab, and press the "Install Tray Support" button. You will need to reboot your system to start this feature (this allows PowerPro to capture all tray icons as they are created).

You can also use the *TrayIcon command to work with or hide individual tray icons.

If you want to display only certain tray icons, 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. To find out the exe file name, you can use the Windows 98 program "System Information" or you can use *Exec WindowInfo over a window displayed by the tray icon.. To find the window caption, try running the command *Menu Show menu hidden which will show all windows and their captions. 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).

If you logoff, you will lose tray icon support. You must reboot to restore it.

This feature only works reliably in Win95/98. Some users have got it to work in W2K and XP, but others reports it makes their systems unstable.

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 List | Propertiesdialog and set the name of the folder in the edit box "Show Entries in this Folder as Left-Click Buttons at end of Bar. Use a wildcard file name to select the files to display (eg *.* 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 text" to have the file name shown as text beside the icon. You can set the maximum number of characters in the label with Maximum Text Label on Properties.

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

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.

Tray Icon Buttons

Using the Command List Setup dialog, you can specify that Windows PowerPro display a the items in a command list as a tray icon on the Win95/NT4 task bar. Configure a command list to contain the tray icons you want and then select that command list on the Command List Setup dialog.

You can also use this feature to replace the text of the clock in the system tray by any dynamically varying text using a *Info. Configure any command list item name with a text label and this label will replace the clock.

Left/middle/right mouse clicking on the tray icon runs the commands configured on the item.

You can include tool tips for any tray icon. These tool tips can contain dynamically varying text using a *Info labels; note however that Windows 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 Windows 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).

Omitting Windows and Words from Active Window Lists

You can use the omit list edit box on the Menu and Bar Setup dialog to omit either words from a window name or to omit windows completely from the list of active windows or 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).

Hiding Windows

You can use the *Exec Hide Window built-in command to hide windows. You might use this if you do not want a window to appear on the Wndows TaskBar, or the Windows PowerPro the list of active windows or the active task buttons.

When you execute a hide window 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 Windows PowerPro window or the desktop window. Some other programs will also refuse to be hidden.

If you execute hide window but then decide you do not want to hide a window, left click the mouse on the desktop or on a Windows PowerPro window to cancel the operation.

If you want to show a hidden window, configure a Windows 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.

Automatically Running Commands when Windows Open

You can automatically run commands when a window with a specified caption is first created. A command could send keys, or press a button, or set the window position, or move the window to an existing virtual desktop, or execute any other Windows PowerPro command.

Use a command list to do this. Each item on the command list corresponds to a command you want to run when a window opens. The command list item name specifies a caption list of the window. 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. Use c=*xxx* to match windows with a class containing xxx. 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.

Specify the command list name in the "Open" drop down box on the Command List Setup 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, Windows PowerPro will execute the corresponding command from the command list.

To press specific buttons on the windows, use send keys to send alt-x, when x is the button mnemonic letter, with {to autorun} at start of the *Keys sequence.

To position the newly opened window on the screen, use *PowerPro Window Control with autorun as the target window id of the command.

To show a menu, you can use *Menu Show. However, you may have to put *wait 1 in 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.

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 **@**.

Windows PowerPro normally executes all commands in the command list which match the caption. However, if the caption matches a 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 its name on the Command List Setup.

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).

Adding Entries to the Explorer Right Click Context Menu

Purpose

When you right click a file or a folder in Explorer, a context menu is displayed. Actually, two different menus are displayed: one for files and one for folders. PowerPro lets you add options to either or both of these menus. You can configure to display menu items for all files/folders and you can also configure to selectively display other items for selected file names or types.

Configuration

First, you must install PowerPro context menu support by pressing the button on the Setup tab of the configuration dialog. Then you need to create a command list to hold the items you want to add to the menus. If you want to add to the menu for files, you must create a command list called Context. If you want to display menu items beside folders, you must create a command list called ContextFolder. The command list names must be Context for files and ContextFolder for folders.

The Context item names and associated commands will be added to the right click menu for all explorer files. If you select one of the items, the left command will be executed with the selected file 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.

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 |. Use || to get the folder excluding the file name.

c:\prog\salamand.exe "||

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

If you right click on a shortcut in explorer, the file pointed at by the shortcut will be used.

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

The command list will be inserted directly into the main menu. Start with a *Format StartSubmenu if you want to insert the items on a submenu.

Before processing commands, the variable Context is set to 0; this allows the files in multi-file selection to be counted or special processing for the first file by using Context as a flag.

Displaying Different Items for Different Files

You can use *Format Context to make the items displayed depend on the file selected. The file name is matched against the *Format text; for example, if the *Format Context test 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.

Restrictions

You can display at most 128 items. You can display at most 9 submenus. You cannot embed *Window menu or *Menu Folder commands.

Example

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

Item Name	Command, Action, Parameters
Purge	*Exec Prompt 1 Purge
	*Script if flag 1
	*File DeleteNoRecycle
Edit	*Format StartSubmenu
WordPad	C:\windows\wordpad.exe
NotePad	C:\Windows\notepad.exe
Myeditor	C:\Program Files\Myeditor\myeditor.exe

```
                *Format EndSubmenu
                *Format Context *.bmp, *.jpg
Wallpaper       *Wallpaper ChangeTo
Edit            C:\program files\image\image.exe
                *Format EndContext
```

Clicking on a file will display a file purge item and a submenu of editor selections. If the selected file is a .bmp or .jpg file, then items for wallpaper changes or processing with the image program will also be displayed.

Hot Keys and Mouse Action Commands

Purpose

Hot/Keys and mouse action commands let you associate a hot key or a mouse movement with starting a program, changing the look of a window, changing your windows configuration, sending keys, showing a menu, or performing any other Windows PowerPro function. Hot keys let you expand the way you interact with Windows.

Configuration

The Keys/Mouse dialog is displayed when you click on the Keys/Mouse tab. Press Add to add a new hot key, Delete to remove one, and Edit (or right or left doubleclick) to change a hot key.

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 Setup dialog:

You can create global macro keys to paste text phrases or paragraphs by assigning the *Keys built-in command or the *Clip File command to a hot key. Or create a menu of *Keys commands and *Menu Show the menu using a hot key/mouse action.

You can assign double click to a mouse action by associating the mouse action with the *Mouse command.

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.

Hot Key and Mouse Action Setup

Purpose

Use the Setup tab to fine tune Hot Key/Mouse Action command execution.

Configuration

You can use the check boxes to 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).

You can use the check box to specify that Windows 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, Windows 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.

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 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 hot 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 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.

Entering Hot Key/Mouse Action Information

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.

Disable: Check "Disable" 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.)

The command entry controls at the bottom of the edit hot key dialog are used to change the command or builtin (*) command run when the hot key is activated.

Hot Key/Mouse Action Explanations

You can use these actions to activate commands with hotkeys.

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	moving mouse to screen edge
left anywhere	left mouse click anywhere
middle anywhere	middle mouse click anywhere
right anywhere	right mouse click 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 on anywhere 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
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
middle double anywhere	middle double click
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

left drags 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 Configure|Keys|Setup.

while 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

while forward only or back only move mouse wheel one position forward or back. Use Key|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)

Window-Specific Hot Keys

Purpose

You can define hot keys which function depending on whether or not windows you specify are active or whether or not the mouse is at a specified position. 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 list of windows for the hot key.

To define a hot key which functions for all but 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.

For example, 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*

You can define the same hot key several times if you want to use the same command for several programs or you can define the same hot key to mean different things in different programs.

Instead of windows, you can specify that the mouse must be at a screen corner or screen edge or quadrant with @topleft, @topright, @bottomleft, @bottomright, @top, @bottom, @left, @right, @quadtopleft, @quadtopright, @quadbottomleft, @quadbottomright.

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, Windows 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.

Scheduler

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, when an idle period ends, and for when Windows 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 button to add a new alarm, the Delete button to remove an alarm, or use the Edit button or double click on an alarm to change it. You can also right click on the list of alarms to access a popup menu.

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 the alarm to execute a *Script run command. To run a series of commands at a startup, associate a startup alarm with a script.

You can specify the year, month, day ordering for dates, and other aspects of alarms, using the Setup dialog.

Windows 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.

When a message box alarm rings, you can change the message text and re-schedule it, if you like. You can select the time until the next alarm from a drop down box or by entering at as months:days:hours:minutes. When the alarm message is shown, for message box alarms which you show repeatedly, 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.

Entering Information for A Scheduled Command

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 Windows 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.

Interval: Scheduled commands can be configured to repeat: use the interval drop down to control whether and when the alarm is recycled to be re-used. You can select a standard recyle 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.

Command: Enter the command to be run in the command entry controls at the bottom. Use the builtin (*) command *Message to enter a reminder message.

Run if missed: Check to have command run if it occurs when PowerPro is not running: 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.

Scheduler Setup

Purpose

Windows 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 Windows PowerPro ring alarms which occur when Windows PowerPro is not active. Otherwise, missed alarms are not rung but are recycled or discarded according to the alarm setting. (However, alarms less than four minutes old are always rung).

Play the alarm sound when an alarm displays a message box.

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 the to message

Specify whether or not alarm messages should be shown on top of the active window when the alarm rings. Gray-check to specify messages to be shown "always on top".

Specify whether Escape 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.

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 (eg every 15 minutes).

Set a resource warning level percentage to have Windows 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.

Use the Media dialog to set the sound associated with alarms and chiming.

Running Programs After the System is Idle for a Specified Time

You can run a program after the system has been idle for a specified time by using an alarm. For Windows 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. You can also use post-idle alarms to run programs after the idle period ends.

Use the radio button to select an idle alarm. 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 *Script variable lastidletime is set to the total number of seconds the computer was idle (including idle time before the idle alarm occurs).

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, Windows PowerPro will not restart it.

Windows 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 Windows PowerPro activates idle alarms in error.

Alarm Log

You can ask Windows 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 configuration file used of Windows PowerPro, except that the file extension will be .alarmlog. For example, the log file for the default configuration is PowerPro.pcfg.alarmlog. The log is always placed in the same directory as the Windows PowerPro .pcfg 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.

Timers

Purpose

Windows 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. Windows PowerPro can produce a timer log to detail this tracking information.

Configuration

You can change timer settings using the dialog or using commands.

To access timers from a dialog, select the Timers tab from the configuration dialog.

You can also start, stop, toggle, and clear any of the timers with built-in *Timer commands. This command allows the following actions:

Start	starts the indicated timers
Stop	stops in indicated timers
StartStop	start the timer if stopped; stops it if it is running.
Clear	zeros the timer
Set	sets and starts or stops the timer

For Start, Stop, StartStop and Clear, you also 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 letter of the timers.

You can also use the Set Timer command to start, stop, or toggle timers and to set their value.

Using the item dialog, you can have Windows PowerPro place a timer as the label on any button. Use the *Timer entry for the item name field in this dialog to indicate which timer is to be displayed. You can also a label is to be shown with the timer value.

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**.

To automatically clear a saved timers once per day, set up an 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 Windows 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)

You can ask Windows 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.

Setting Timers and Associated Commands

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.

Use the check boxes and buttons to set or clear any timer, start or stop it, and assign a label to the timer (the label can be displayed on the button with the timer and in the timer log.

You can specify that the timer should start automatically when Windows PowerPro starts. You can specify that the timer values should be saved and restored when Windows PowerPro starts and stops. You can indicate that the timer should count down.

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).

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. Windows 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: eg 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".

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 (eg netscape for Netscape Communicator) Windows PowerPro will arrange for the timer to be running only when the specified program is the foreground (active) program

You can also associate a command with starting, stopping, and resetting the timer using the command entry controls .

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. For 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 built-in commandto set a timer value and state.

Setting Timer Value and State

Use the built-in *Timer Set command to set the value and state of one or more timers. The parameters edit box of the command is structured as follows:

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 letter @ or $ if you want to add or subtract the value, rather than setting the value. Omit the @ and $ to set the timer.

Next come the single letter timer ids of the timers to be adjusted, with no blanks.

Finally, the new timer value is indicated as three numbers: hours, minutes, seconds, separated by blanks.

Examples

+a 0 0 0 Clear timer a and start it.

+a 0 0 120 Start timer a at 120 seconds.

be 0 10 20 Reset timers b and e to 10 minutes, 20 seconds; leave their running/stopped state unchanged.

-c 1 0 0 Stop timer c and set its value to one hour.

a q 2 3 0 Adds 2 hours and 3 minutes to timer q.

Timer Logs

You can ask Windows PowerPro to log timer events by using the Timer Log check box on the Timer dialog or the Log check box on the individual timer configuration.

The log file will have the same name as the configuration file used for Windows 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 Windows PowerPro .pcf file.

A log file entry will be written whenever a timer starts, stops, or is re-set. As well, when Windows PowerPro shuts down, a stop timer entry will be written for any running timers. When Windows 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, ie 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.

Media Dialog

Purpose

The Media dialog is used to set sounds, screen saver, and wallpaper, and to control automatic changes of these files by Windows PowerPro.

Configuration

To change sound, wallpaper, or screen saver information, double click on an entry from the list or use the Edit button.

When you select an entry to be edited, you will be able to change the associated file and to 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 each 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.

Windows PowerPro allows you to use jpeg files as well as bmp files as wallpaper.

When Windows PowerPro changes the screen saver, you can set whether or not Windows PowerPro will stop any running saver and restart with the new saver using the Setup dialog.

PowerPro Sounds

Windows 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.

Windows PowerPro supports access to many standard Windows sounds in the Registry plus these sounds:

Windows PowerPro Chime	Plays whenever Windows PowerPro chimes (see Scheduler Setup dialog)
Windows PowerPro Alarm	Plays whenever Windows PowerPro alarms (see Scheduler Setup dialog). Use a single asterisk to have the PC Speaker beep for alarms.
Windows PowerPro Clip	Plays whenever Windows PowerPro captured a clipboard item.

Command Entry Controls

Purpose

Windows 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 start with an asterisk (*).

Configuration

To enter a command: type the command into the edit box, select it from 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, or use the wizard button to select a command using wizard menus. Following is a description of the remaining controls:

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 show dialog controls specific to that command.

Use the Copy and Paste buttons to copy and paste commands between different sets of command entry controls.

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 this 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 prompt for input or insert text from calculations by using expressions. Set the "expression follows" character to (say) & on setup|advanced dialog, dialog tab chars. The to prompt for input anywhere in the command or parameter, use
&(input "title")

You can also insert the results of an expression directly as text anywhere in a command or parameter, for example
&(date select 4)
would insert the 4four digit year.

To use the character & without change, precede it by a single quote (‘).

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 want to check "Play .wav internally" on advanced dialog to have PowerPro play the sound (rather than the associated program for .wav).

Running Multiple Commands

There are two ways to run multiple commands:

Put all the commands on a command list and use *Script. Using a script is the most general 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 set of command entry controls. This is convenient for a small number of commands but there is limited space..

Using the command entry controls, you can specify three or four commands in the More Commands edit box to enter the second and subsequent commands. Start each command on a new line. Enter command directly or use the button for a dialog.

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 format file 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)
For example
c:\windows\notepad.exe edme.txt !`d:\mydir *min
starts notepad minimized on file edme.txt with starting folder d:\mydir.

Example:

Command:	Notepad
More Commands	*wait 1
	*keys hello

Starts notepad, waits for 1 second, then sends the "hello".

PowerPro Built-In Commands

Windows PowerPro comes with a set of built-in commands which let you manipulate running programs and control the Windows configuration. You find the built in commands in the Command drop down control of the Windows PowerPro command entry controls .

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
*Menu	Display menus built from command lists, folder contents.
*Message	Display a message.
*Mouse	Send a series of mouse actions to another window.
*Screen	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, tray min, rollup, etc any window.

Bars and the *Bar Command

Purpose

Use bars to execute commands on command lists from a toolbar. Some features of toolbars:

- You can use left, middle, and right clicking to execute different command.
- Configure bar positions and other features by Ctrl-right clicking a bar.
- Move bars by Left dragging; resize using sizing border (position must be "Floating" and "Bar size to sum of buttons" must be unchecked).
- Bars can be automatically hidden
- Bars can be positioned in the caption of 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 on bar buttons from explorer; left drag drop runs the button command and right/drag drop configures a button.

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 set use the bar setup dialog to require that ctrl be down for dragging a bar to move it. In this case, 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".

Bars are configured using command lists and their look is set using Bar Properties. Bars can be shown automatically at start-up by checking "Show as Bar" on the Command Lists dialog, or they can be shown using the *Bar show command.

The buttons on bars can come from three sources: the entries in a command list, the files and folders in a folder on your disk, and the currently active windows on your system.

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.

Configuring the *Bar Command

Use the following actions with *Bar:

Show	Shows a bar.
Hide	Hide a bar but keeps it in memory (for faster reshow).
HideShow	Hide a bar if visible; show otherwise.
Close	Closes a bar and removes it from memory.
ToMouse	Temporarily moves bar to mouse. Usually used with hotkey. Floating position only.
Keys	Readies bar to receives keys.
SelectSubBar	Shows a subbar.
SelectSubBarToMouse	Shows a subbar at the mouse
SelectSubBarToButton	Shows a subbar aligned to a button

For Win98/2000, 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. If it is, you can also use *none to override any default slide animation. For example

For example

*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, eg

*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 *Window to access bar. The commands hide, hideshow, show, position, and close can be used. 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.

Screen Edge Positions

You can position a PowerPro bar at a screen edge and have it reserve a strip screen space like the Windows taskbar by selecting the Top, Bottom, Right, or Left Screen Edge position on the Bar Properties dialog or on the Position submenu of the configuration menu shown when the bar is Ctrl-right clicked.

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

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 List 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.

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.

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.

Showing Other Bars when you Click a Main Bar

You can configure a bar to show other bars at your mouse when you click a button. For an example, try the BarShowsBars configuration which you should find installed in your Powerpro folder and which you can access 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 the bars which appear when you click the main bar. Each of these other bars are autohide bars with "Auto Show as Bar" checked.

Ctrl-right click the bar and note the configuration of Bar command list. The Saver buttons 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 the 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 (win98/2000 only).

To show the vertical bars when the mouse hovers of the main bar, select Properties for Command List bar and check hover clicks.

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

Using Subbars to Display Different Parts of Bars

Subbars let you can display some buttons on a bar and hide others. You can use a *Bar command or the virtual desktop name to indicate which parts of the bar should be displayed.

Configuration

There are two steps to creating these subbars.

First, you put a *Format StartSubBar at the start of each subbar and a *Format EndSubBar at the end. These must be Left commands. Use the list item name of the *FormatStart SubBar to set the name of the subbar. Buttons which are not within subbars are always displayed. (You cannot nest subbars. You can repeat the same name on different *FormatSubBar commands.)

Second, you use the command

*Bar

SelectSubBar

xxx @PartBar

to show the subbar called PartBar on bar xxx. All other subbars are 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 an always-shown button on bar xxx, or on another bar, or on a hot key, etc.

Buttons which are not within subbars are always displayed. You cannot nest subbars. You can repeat the same name on different *FormatSubBar commands (ie 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.

In addition to using the SelectSubBar command, you can also show a subbar when you switch to a virtual desktop. Give the subbar the same name as the virtual desktop. Check "Show subbar of same name as vdesk" on virtual desktop setup.

Example

Suppose a bar called mybar is configured as follows (middle button omitted for clarity):

Item Name	Left	Right
Select	*Bar SelectSubBar mybar @edit	*Bar SelectSubBar @misc

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

The Section/Subbar Approach to Configuration

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 subbars and 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, eg 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:

```
Editors     *Bar                SelectSubbar    @editors_
Internet    *Bar                SelectSubbar    @Internet_
Utilities   *Bar                SelectSubbar    @Utilities_

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 List|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.

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.

Drag and Drop onto the PowerPro Button Bar

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

Left drag and drop starts a command with dropped file(s) as the parameter. Right-drag/dropping file(s) onto the bar activates a menu allowing you to select the button to receive the file or to be configured or to quickly add a new button using the dropped file as the left command.

Sometimes you want to drag and drop files in 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.

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).

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

Using the keyboard to access the button bar

You can use the keyboard to access the commands on a Windows PowerPro button bar.

First, you need a way to activate the bar 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 the hot key, the mouse cursor will be moved to the bar and the bar will be ready to receive any of the following keystrokes:

L	activate left command of current button (you can also use Enter)

M	activate middle command of current button
R	activate right command of current button

left arrow	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 position preceding *Bar Keys command

Using *Bar Format to Change Bar Format

Use *Bar format to change the background, autohide interval, or position of a bar.

To configure the command, select *Bar command, Format action, and set the command list drop down to the bar to be accessed. 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 alternative 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. Set 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.

Adding a Button

1. Ctrl-right click bar and select configure.
2. Select command lists tab.
3. Select bar name from drop down.
4. Select current item to precede or to follow new button.
5. Press Add New After or Add New Before.
6. Enter new name, icon, command on edit item dialog.
7. Press OK.
8. Press OK to close configure.
9. If "Bar size to sum of buttons" is unchecked, resize bar using sizing border to see new button.

Changing a Button

1. Ctrl-right click bar and select configure.
2. Select command lists tab.
3. Select bar name from drop down.
4. Double click on item to be changed.
5. Enter new name, icon, command on edit item dialog.
6. Press OK.
7. Press OK to close configure.
8. If "Bar size to sum of buttons" is unchecked, resize bar using sizing border to see new button if needed.

Tip: Press and hold a button to quickly configure it.

Deleting a Button

1. Ctrl-right click bar and select configure.
2. Select command lists tab.
3. Select bar name from drop down.
4. Click item to be deleted.
5. Click delete button or key Del.

Moving Bar

1. Ctrl-right click bar, select look submenu, make sure Floating is selected.
2. Left click anywhere on bar and drag to new position (you will also need to hold down Ctrl if this option is set on Command List Setup dialog).

Tip: You can assign *Format Drag command to any button; then click and drag on that button to move bar even if you have set Command List setup to require Ctrl key.

Tip: Use submenu of Floating entry to quickly align bar on screen.

Positioning the Bar

Floating Position: Bar can be dragged by ctrl-left click. Bar can be resized if sizing border is checked and "Bar size to sum of buttons" is unchecked.

Locked: Bar cannot be moved or resized.

Caption/Above/Below/Left/Right: Bar moved to caption or positioned near active window. Size the bar before selecting this option. Bar can be moved by dragging.

Screen Bar: If not autohide, bar reserves screen space like Windows Taskbar. There are two types of screen edge bars: current size and full screen.

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 List 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.

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: On bar sits on Windows Taskbar.

Fixed: Bar will stay in same position independent of screen resolution.

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 set use the bar setup dialog to require that ctrl be down for dragging a bar to move it.

Creating an Autohide Bar

1. Ctrl-right click bar and select configure.
2. Select command lists tab.
3. Select bar name from drop down.
4. Press Properties button.
5. Select autohide delay from "Hide After" drop down.
6. Select screen edge to bump from "Show if Bump" drop down.
7. Press OK to save properties and OK again to close configure.

Tip: If you cannot make a bar appear, use ctrl-alt-del to shut down PowerPro and restart. Bars will appear at start up. Or execute Pproconf.exe program to reconfigure.

Tip: You can control the amount of time the cursor must be held at edge of bar with Key/Mouse tab, Setup button, Screen edge delay setting.

Tip: If you prefer another technique for showing the bar, configure a hot key or button or menu (from another bar) with the *Bar Show command.

Creating a New Bar

1. Ctrl-right click bar and select configure.
2. Select command lists tab.
3. Press New List and enter name for bar's command list then press OK.
4. Add items buttons for bar using Add New After or Add New Before.

5. Press Properties and set look for bar: icons, maximum text label, tool tips, etc.
6. Check "Show as Bar on Command Lists tab to automatically show the bar at PowerPro start. Or use the *Bar Show command to show the bar when desired.
7. Press Apply to preview (if Show as Bar checked).
8. Press OK to save.

Removing a Bar

1. Ctrl-right click bar and select configure.
2. Select command lists tab and select bar to be removed from drop down.
3. To stop showing bar while keeping its configuration: uncheck Show as bar. Ctrl-right click bar and select Close bar
4. To remove bar and its configuration information: Press Delete List and then press OK to confirm.
5. Press OK to save.

Bar Look

Flat: If checked, buttons are shown flush with bar unless mouse cursor is over them. To keep buttons flat even with mouse cursor is over them, gray-check flag on Properties for bar's command list.

Topmost: Bar is shown on top of all other windows.

3D/Sizing Frame: a 3D border is drawn around the bar. If the Position is floating and "Bar size is sum of all buttons" is unchecked, then left clicking on this border and dragging will change the shape and size of the bar. The label "Sizing Frame" will appeal only if you can use the frame for sizing; otherwise it will be called 3D Frame.

Border: a black border is drawn around the bar.

Text under icons: text is drawn under icons.

Buttons same size: button width is set by width of first button.

Vertical: Buttons are aligned beneath each other.

Right icons Iocns are shown to the right of button text.

Center text: center text label on button.

Bar size from sum of buttons: If checked, the bar size is set by PowerPro and the bar cannot be resized with the sizing border. The size will be changed if there are active buttons as the number of active tasks changes. It will be one row or one column unless *Format NewBarRow is used to force new rows on non-vertical bars.

Positioning PowerPro Bars in or beside the Foreground Window

You can position Windows PowerPro bars at the left, middle, or right of the foreground window caption or to the left, right, below or above the foreground window by selecting such a position as the position in the Bar. You can also specify an offset for the bar from the base position with this dialog.

Use Default Menu and Bar to avoid putting Windows PowerPro bars in captions of or beside dialog boxes.

When there is no active window to use for the position 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 the appropriate check box on the Use Default Menu and Bar dialog.

If you want to make the window visible only if certain windows are active, use *Format Context at the start of the bar command list. You can have several such bars, each positioned in the caption of or beside windows but only visible if the specified *Format Context windows are active.

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

Clipboard Manipulation, Tracking and Copying

Purpose

The *Clip command is used to work with the clipboard. Windows PowerPro has commands to copy text or file to the clipboard, to copy the clipboard to a file, and to clear the clipboard.

Furthermore, Windows PowerPro has a clipboard history function: it can track text as you put it to the clipboard and can subsequently display a list of such text items on a menu. If you select an item from this menu, the selected item is recopied to the clipboard and optionally pasted.

*Clip Actions

Menu	Show menu ; select an entry to put on clipboard.
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.
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	Appendclipboard to text file.
ClearClip	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 entered text on clipboard.
TextAppend	Append following text to clipboard. If no text, newline appended.
TextPaste	Put entered text on clipboard and then automatically paste with Ctrl-V.
Copy	Sends Ctrl-C to foreground window for copy.
Cut	Sends Ctrl-X to foreground window for cut.
Paste	Sends Ctrl-V to foreground window for paste.

Capture Turn capture *off, on*, or *reverse* current state You can follow *on* by a folder name, say c:\path, to start capturing clips in folder c:\path. You can replace foldername by word *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(ie use *Clip capture on refresh).

TextPaste will often be faster then *Keys for long text.

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

Windows 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 with a manual copy attached to one hot key and a filepaste and the same file name assigned to the other hot key.

Clipboard History Tracking

To enable automatic tracking of plain text as you paste it to the clipboard, you must check "Keep clips" on the GUI Control dialog. When this is done, Windows PowerPro will automatically track the most recent items pasted to the clipboard. Gray check to capture rich text format as well as plain text.

You can also ask Windows PowerPro to automatically copy selected items to subfolders of your clip folder with filter strings.

If you set a non-zero value for "Milliseconds mouse hovers over *Menu Folder" on advanced setup dialog, PowerPro will show the first few lines of the stored clip as a tool tip for the *Clip Menu display.

You can play a sound file each time an item is captured by setting the PowerPro Clip sound on the sound tab. You can run a command list called "ClipCaptured" by checking the "Run ClipCaptured" checkbox.

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.

To conserve memory, PowerPro normally only captures clips less than 63K in size. You can increase this to 250K by checking "Large Clips" and to 1000K by gray checking "Large Clips" on the GUI Control dialog.

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 check "Show tool tips for *Men Folder and *Clip Menu" on Command List setup, PowerPro will show the first few lines of the stored clip as a tool tip for the *Clip Menu display.

Further Information

Clipboard plain text is actually stored in a .txt file in the clip subfolder of your main Windows 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 Windows PowerPro can read.

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.

Clip Filters

You have Windows 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

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 Windows 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".

If the captured item is longer than 250 characters, only the first and last 125 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.

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	Show desktop icons.
HideShowIcons	Hides icons if visible; shows them if hidden.
HideTaskBar	Hides all TaskBar on desktop
ShowTaskBar	Show desktop TaskBar.
ShowTaskBarautohide	Shows taskbar; re-hides when mouse is moved off taskbar and taskbar is not foreground window. Assign this command to screen bump hot key to show autohide taskbar but prevent inadvertent shows by movements near screen bottom when taskbar is hidden. You may also need *Desktop hidetaskbar as startup scheduled event.
HideShowTaskBar	Hides TaskBar if visible; shows them if hidden.
HideShowWindows	Hides all desktop windows; show them when next executed.
MinShowWindows	Minimizes all desktop windows; show them when next executed.
SaveIcons	Save desktop icon positions
RestoreIcons	Restore desktop icon positions
SaveIconsGrid	Align desktop icon positions to a grid and then save them.
Resolution	Change resolution
TransIconText	Makes transparent background for desktop icon text; use keyword auto to reset transparency when background changes
IconTextColor	Set color of text under desktop icons; use keyword auto with TransIconText function to reset color when background changes
SetWorkArea	Sets size of work area on desktop screen: that is the size of maximized window. Specify four numbers left, top, bottom, right. Relies on system routine built into Windows which controls effect of command.

Saving and Restoring Desktop Icon Positions

Use the Built-in commands *Desktop SaveIcons, *Desktop RestoreIcons, and *Desktop SaveIconsGrid save/restore the relative positions of desktop icons and to align icons according to a grid. Assign the commands to a button or menu, and execute them to save/restore your desktop icons positions.

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 Icon commands, 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. For example:

Command: *Desktop SaveIconsGrid

Parameter 30 20

aligns icons so that horizontal pixel position is a multiple of 30 and vertical is a multiple of 20.

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 Windows 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.

Changing Screen Display Resolution

You can change the display resolution, color depth, and refresh frequency (NT only) with the built-in *Desktop Resolution command.

If you use this command with nothing in the parameters edit box, Windows 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 Win 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 x 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.

Using *Exec

Use the *Exec command to access various functions; the *Exec keywords are:

BrowseRun:	Shows a file open dialog; the select file is immediately executed.
CommandLine	Shows a tiny command line to enter a command to run.
HideWindow	Shows a dialog allowing you to pick a window to hide.
ContextMenu	Show right click menu of window under mouse; usually followed by *Keys {to menu}… to select entry from 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
Autoscroll	Starts automatic scrolling.
ScrollwithMouse	Starts manual scrolling.ScrollWindow Scroll window under mouse.
ClearRecent	Clears recent command folder on Start Menu.
AutoPress	Used to learn new types of windows for mouse stop/press
WindowInfo	Shows/hides a small window showing mouse position and window size.
Disable	Disables PowerPro until mouse moved over a bar or hot key used.
Mute	Mutes sounds; run again to reverse.
RestoreLastMin	Restores last minimized window.
Alarms	Can suspend or re-activate checks for scheduled programs.
ClearRecentExcplorer	Clears list of recent explorer folders shown my *Menu Explorer.
ToFile	Writes a line of text to a log file.
Setenv	Set environment variable.
Prompt	Sets a flag according to a prompt
Dos	Starts Dos, runs a command line, restarts Windows (not for NT).
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.
RefreshEnvironment	Refreshes all environment variables from registry (NT only).
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. You can use positive or negative numbers for differences. Use mouse or arrow keys to change dates. Or click on appropriate field of date and type new year, day number, or month number. Unavailable on early Win95 versions unless IE3 or later has been installed.
Monitor	Can suspend or re-activate repeated running of monitor command list as set on Command List Setup.
HotKey	Can 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 numbers after on/off/reverse to act only on those hot keys. The hot key number can be found by exporting the hot keys to a file with setup|advanced|export text; it is the first number.

EmptyRecycleBin Empty 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.

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.

LogKeys Log keystrokes to file.

NewFolder Creates a new file folder.

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.

Print Print file using associated program.

CD use command play, next, previous, eject to play audio CD.

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 be 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. For example *Exec SchedulerAdd 20021225 0900 Merry Christmas. For example, *Exec SchedulerAdd 0 +100 "c:\program files\myprog\mprogra.exe" runs myprog one hour from now.

ChangeConfiguration changes to configuration stored in new pcf file; the new file path can be entered in the command or it will be prompted for if no path is provided.

Suspending Alarms

You can suspend ringing of alarms by executing the following command (eg 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, ie 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 Setup configuration dialog.

Tiny Type and Run Dialog

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 25 commands entered.

Put the *Exec CommandLine command as a Windows PowerPro start up alarm if you want the run box to appear when Windows PowerPro starts.

If your command file name contains blanks, you must surround it by double quotation marks.

With NT4/Win98, or if you have installed IE3 or later (even if you no longer use it), you may be able to type World Wide Web URLs directly into a run box and have them executed. Try this to see if you have this feature. If not, then right click on the run box and select "Check for URL"; in this case, Windows 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 this .bat file is executed. Use explorer to set the properties of this .bat file to change its configuration (eg full screen versus window). Precend the Dos command by *hide to run it in a hidden window. For *Dos commands, if you use *Script Assign to set the variable CurrentDirectory to a drive:folder, PowerPro will do a cd /d to the directory 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 desired dimensions. Windows 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)

or specify that all commands expect those starting with "win " should be prefixed by the ksh shell prefix *dos ksh -L -c; you can change the shell prefix with the shellprefix internal option

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 Windows PowerPro should try to interpret the command as a URL to send to a running browser

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.

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 the Tiny Run Box command).

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 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 (.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, eg {alt} or {pgdn}. In addition, {alt}, {ctrl}, {shift}, and {apps} 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 logkey to display an X on a button label if logging is active or the keyword logkeyfile to view the file name of a logging file. You can also access this information in *Script variables with *logkey and *logkeyfile.

Date/Calendar Calculations and Display

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 More Commands of 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.

Use the dialog as follows:

To display 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 (eg 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 calendar, you can also use *Exec Calendar.

Prompting for Yes/No Information

Use *Exec Prompt to prompt for a Yes/No answer an 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.

You can also prompt for a yes/no/cancel result by using a variable instead of a number:

Command *Exec
Action: Prompt
Parameter c Any text

Displays a yes/no/cancel dialog and sets c to 0 for no, 1 for yes, 2 for cancel.

Sound Volume

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, where n is a number between 0 and 255. Use +n or -n to adjust volume relative to current setting.

CD Functions:

Use *Exec CD to control your audio CD player. Enter one of the following commands in the parameters box.

Play n	Plays tracks start an number n. Omit n to play starting with track 1.
Next	Plan next track.
Previous	Play previous track.
Eject	Ejects (opens door for) default audio CD.
Close	closes door for default audio CD.

Writing Entries to a File

You can use the following command to write a line of text to a file:

Command *Exec
Action ToFile
Parameter "filepath" text
writes the 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 phase date include (assuming & is variable insertion character) to c:\logs pth\log1.txt

PowerPro *File Commands

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.

Click here for more information on copying and running commands with random files.

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.

Working with a Randomly Selected File

Purpose

The *File RunRandom/CopyRandom/CommandRandom actions can be used to select a file at random using a file path with wild cards that you provide. The selected file is then copied to a specified target file; or is run directly; or is used in a command line to execute a program or batch file that you specify.

One use of this command is to set up your own randomization routines for system 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.

Configuration

CopyRandom: Parameters: filepath outfile
selects a file at random from the filepath (which must contain wildcards like *.bmp) and copies it over outfile.

RunRandom: Parameters: 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.

CommandRandom Parameters: 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.

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 RandomCommand
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.

Sending Keys to Other Windows

Purpose

Use the *Keys command to send keystrokes to other windows. The keystrokes entered in the parameters edit box are sent to the currently active window. You can send keys to enter text, such as canned phrases or passwords, or you can send keys to automate functions by sending Alt-key or Ctrl-key pairs recognized by programs to execute their functions. 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.

Configuration

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 {enter} or {back 3}.You can use the find button on the configuration dialog to select a special key or to record keys.

To specify an Alt-prefixed key, prefix it by **%**; similarly use **^** for Ctrl key, **+** for Shift, and combine as needed (eg %^ for both Ctrl and Alt). Note that you can often simulate menu selections by sending % followed by a set of characters; eg %fn sends Alt-fn which does a File|New menu selection in many programs.

Beware of these characters which have a special meaning for Windows PowerPro:

% use {pe} or {%} (% alone signals Alt)

^ use {ca} or {^}(^ alone signals Ctrl)

\+ use {pl} or {+} (+ alone signals Shift)

Example

Command *Keys

Parameter hello, world

Sends hello, world to active window.

Normally keys are sent to the currently active window. But you can switch to another windows first by preceding the sequence of keys with {to xxx} where xxx selects the new target window.

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.

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. 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. 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 first line in the file, then any number of lines of comment text, then ** again at the end of a line.

PowerPro has two methods of sending keys, the "fast" method (journal hooks) and a slower method, which is the default. Use Configuresetup|advanced|others to change. The default method works for most keys, but the fast method is needed to send shift in XP and W2K, 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}

To send mouse clicks, use *mouse.

You can control the delay between sent keys with the advanced options.

Specifying the Window to Receive the Keys

For the overview, see sending keys.

The *Keys command normally sends keys to the foreground (active) window. You can reset the foreground window before sending the keys 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:

:

*	sends keys to current active window
=File Path	sends keys to program run from that "File Path"
Title	sends keys to window with caption "Title"
PartTitle*	sends keys to window with caption starting with "Part Title" (Note asterisk at end)
*PartTitle	sends keys to window with caption end with "Part Title" (Note asterisk at start)
PartTitle	sends keys to window with caption containing "Part Title" (Note asterisks at start and end)
autorun	sends keys to window of last window matched by autorun menu
activebar	sends keys to window of last window referenced by active bar button

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, eg

{to ^*notepad}

avoids the error message if no Notepad window is open.

Specifying the Keys to be Sent using *Keys

For the overview, see *Keys.

Send letters, numbers, and other keyboard characters by typing them as you want them to be sent.

To specify an Alt-prefixed key, prefix it by %; similarly use ^ for Ctrl key, + for Shift, and combine as needed (eg %^ for both Ctrl and Alt). Alternatively, you can use {alt} to toggle Alt up/down which allows multiple keys to be sent with Alt down: eg {alt}ab{alt} sends Alt-Down, a, b, Alt-Up. Similarly for {ctrl} and {shift}.

Use {datelong}, {dateshort}, {time} for sending the current date and time. To send the time without seconds, use {time}{back}{back}{back}.

You can change the either the { or the } or both to any non-alphanumeric using the Advanced dialog. They can be set to the same character.

Use the following character pairs enclosed in {} for special characters. You have a choice between the long form and a two-letter abbreviation. To send the opening brace, use {{}. To send the closing brace, just use }.

You can repeat the character up to 29 times by placing the repeat count following a single space just before the closing brace, eg {left 5} for five left arrows:

{param}, {pp}	Insert the parameter prompt character (default ?).
{clip}, {cc}	Insert the clipboard character.
{var}, {sv}	Insert the script expression character.
{plus}, {pl}	Plus (also can use {+})
{percent}, {pe}	Percent sign (also can use {%})
{caret}, {ca}	Caret (also can use {^})
{{}	Inserts the brace.
{brace}, {br}	Curly Brace (
{enter}, {en}	Enter
{space}, {sp}	Space
{quote}, {qu}	double quote
{question}, {qn}	question mark
{greater}, {gt}	greater than sign >
{less}, {lt}	less than sign <
{tab}, {ta}	Tab
{esc}, {es}	Escape
{apps}, {ap}	Apps key
{up}, {au}	Up arrow
{down}, {ad}	Down arrow
{left}, {al}	Left Arrow
{right}, {ar}	Right Arrow
{ins}, {in}	Insert Key
{del}, {de}	Delete Key
{back}, {ba}	Backspace Key
{home}, {ho}	Home Key
{end}, {ed}	End Key
{pgup}, {pu}	Page Up
{pgdn}, {pd}	Page Down

{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 unchecked
(capslock},{cl}	Caps lock ("use fast send keys" on advanced|setup must be unchecked
dateshort},{ds}	date in Windows short format
{prevshort},{ps}	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
{fn}	Function Key "n" (eg {f1} for function key 1; do not use the letter n)
{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/up; use {at}ab{at} to send Alt-down, a, b, Alt-up.
{shift}, {sh}	Toggle Shift down/up; use {sh}def{sh} to send DEF.
{ctrl}, {co}	Toggle Ctrl down/up; use {co}{ta}{ta}{co} to send Ctrl-Down, tab, tab, Ctrl-up.
{win},{wi}	Toggle Win down/up; use {wi}p{al} to open accessories.

{filemenu c:\path\items.txt}You can select keys to be sent from a menu

Example: "%fnhello^v%{f4}" sends Alt-F, then n, then hello, then ctrl-v, then alt-f4.

If you have only one key to send, the surrounding double quotes are not needed. You can send at most 200 keys.

To send Alt+xxx keys (eg alt+0181 =µ), use {alt}{pad0}{pad1}{pad8}{pad1}{alt}.

Examples of Keys Commands

For the overview, *Keys.

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.

Sending Keys to Programs When They Are Started

For the overview, see sending keys.

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 Commands	*wait ready
	*keys "%gn"

The sequence *wait ready causes Windows 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 5 seconds until a window with a specified caption appears by preceding the caption with a **+** and using the {to xxx} option:

Command: *Explorer
More: *keys {to+*Exploring*}"%vl"
Start Explorer, then waits for up to 3 seconds for the window with a caption containing **Exploring** to appear, then sends Alt-v followed by **l** to the window with caption starting with **Exploring.** This could set the list view for Explorer. This is especially useful with Explorer, where the **!** may not work (since Explorer is always running). You must use the *Keys command for this approach.

Selecting some 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 the menu item text and the corresponding characters to be sent.

Use
*Keys {filemenu c:\path\items.txt}
for this feature. Each line in c:\path\items.txt is of this form
name=keys

There may be many lines in the file. Each line will display a menu item with the menu text 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)
Title=result
Endcontext
The title=result line is only included if the expression is true. You cannot nest contextif.

You can have other keys to be sent beside 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.

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

Creating Menus or Bars of Favorite Folders using *Keys

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 advanced setup 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 (eg) a hot key that runs 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 or beside the active window (use a vertical bar). Don't forget to check "auto-show as bar".

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 a command parameter. Normally, the selected file is placed in double quotes after a space.. 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 |. Use || to get the folder excluding the file name.

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

Formatting Menus and Bars with *Format

Purpose

Use the *Format command to control the look of menus and bars. You can use *Format Context and *Format Item with menus and bars. Other *Format command can only be used in command lists displayed as menus.

Configuration

Use the following actions with *Format:

Separator	Inserts a horizontal separator in a menu;
NewBarRow	starts a new row in a bar.
NewBarRowLine	starts a new row in a bar and draw separator line.
BarVerticalLine	draws a vertical line in a bar.
NewColumn	Start a new column in a bar or a menu.
NewColumnLine	Start a new column with a separating vertical line.
StartSubMenu	The following items in the list appear in a submenu.
EndSubMenu	Ends the submenu. You can nest submenus up to 4 deep.
Context/ContextIf	Starts a menu section or bar which depends on active window. or an if condition
EndContext	Ends portion of menu depending on active window or if condition.
Item	Changes colors and text associated with menu or bar item.
Drag	Assign to a bar button and then click-drag that button to move bar.

Changing the Look of an Item with *Format Item

Execute the *Format Item command to change the look or text of an item on a command list used for a bar or menu.

When you configure this command, use the button 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. You can specify an item number of 0 to indicate the last button pressed on any bar, in which case the list name is not used.

There are restrictions on changing the text associated with special *Info labels. You cannot change the text from an ordinary item label to a *Info special label or the reverse. You cannot change tool tips using *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 later manually use the configuration dialog and save the new configuration. If this is not what you want, you could put the *Format Item commands to reset the desired values in the Reconfigure command list, as set by the advanced setup dialog.

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 (eg leftclick) or two letter short forms (ll):

leftclick or lc	left click (both left down and left up)
leftdown or ld	left down
leftup or lu	left up
leftdouble or ll	double click left (note: lc lc will not work)
middleclick or mc	middle click (both middle down and middle up)
middledown or md	middle down
middleup or mu	middle up
middledouble or mm	double click middle (note: mc mc will not work)
rightclick or rc	right click (both right down and right up)
rightdown or rd	right down
rightup or ru	right up
rightdouble or rr	double click right (note: rc rc will not work)
save or sa	save current position position
alt or al	reverse alt key (ie press if up, release if down)
shift or sh	reverse shift key (ie press if up, release if down)
win or wi	reverse win key (ie press if up, release if down)
ctrl or ct	reverse ctrl key (ie press if up, release if down)
save	save current mouse position
restore	restore saved mouse position
mo x y	move mouse x pixels right, y down (x or y can be negative)
screen or ab x y	set mouse to absolute position x y (absolute means 0 0 is top left of screen)
relative or re x y	set mouse to relative position x y (relative means 0 0 is top left of active window)

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 (eg will draw a line in MS Paint).

Command *mouse
Parameter ab 40 60 lc
More: *keys abc
moves mouse to absolute position 40 60, sends a left click, then sends letters abc

Displaying Menus with *Menu

Purpose

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 RecentCommands), captured Explorer folders (*Menu Explorer), or the Windows Start menu.

Configuration

Use the following actions with *Menu:

Show	Show a command list. 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 List setup.
Explorer	Show folders recently accessed with Explorer. You must check "Track explorer" on Setup dialog.
StartMenu	Show Windows start menu at mouse cursor. Alternatively, you can assign *Menu Folder StartMenu to a hotkey.
Tray	Show tray icons as menu.
UnderMouse	Show menu bar of the window under the mouse. Only works in Win95/98. Must be assigned to hot key or as part of menu shown by hot key.

Note that there are two steps to showing a menu: defining the command list, say xxx, and then executing a *Menu show xxx command. For example, to show a menu by pressing a button, assign *Menu Show 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 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.

You can use *Menu Show or ShowAtButton 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 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)
button	shows menu aligned under/above last button pressed on horizontal bar

horbuttoncenter	shows menu centered under/above last button pressed on vertical bar
verbutton	shows menu aligned right/left with last button pressed on vertical bar

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 submenus.

*Menu Folder

Purpose

Using the built-in *Menu Folder command, you can show a menu listing the files from a folder with subfolders shown in submenus. Left clicking an entry runs the file; right clicking an entry shows the explorer context menu for that entry.

*Menu Folder can display all three types of folders:

ordinary file folders

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. Then 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 menu separator. You can use the word "ColSep" to start a new column in the menu. For example

c:\textfile,colsep,desktop

shows the files and folders in c:\textfile, starts a new column, then shows 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 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. See below for details on this dialog.

You can use *Menu Folder to explore a large tree of files and folders, but if there are more than 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; see below for details on this approach

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 keyword 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 (eg |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.

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.

The 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.

Format of *Menu Folder

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, (see here), its simpler and safer to always use the dialog accessed by pressing the find button.

Controlling Look of Menu

Exclude icons: check to avoid icons on menu. You may also want to check "Add … to folders to easily distinguish folders in this case.

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 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.

Add explorer entry: Add an entry to start of each menu and submenu

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 screen bar 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.

List 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 list|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 etnry: 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 lcick 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 *Menu Folder command. Normally, the entries in the folders listed in the *Menu Folder command are placed in 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 Folder to Include

You can limit 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 dash, eg

-.exe -.dll

omits exe and dll files.

You can omit certain folders by listing names separate by commas in the edit box.

Specifying command to execute

Windows PowerPro normally runs the file 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 read into notepad. PowerPro will create a command line consisting of the command you specify followed by the selected file. 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 single quotes:

'c:\program files\myprog.exe'

Normally, the selected file is placed in double quotes after a space and command. 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. I Note that you must include the quotation marks if appropriate when using |. Use || to get the folder excluding the file name.

*keys {to folder}||

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) by instead of explorer by specifying it, eg

'c:\program files\2x\2xExplorer' '|*'

Note single quotes which will be replaced by double when run.

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 with the you have checked "Add Explorer Entry…"

Special Folders for *Menu Folder

Using the built-in *Menu Folder command, you can display a menu of the special folders used by Win95/NT 4. To access special folders, the parameters edit box for this command can contain one or more of the following (separated by commas).

start menu	start menu entries
desktop	shortcuts on your desktop
recent	recently accessed documents
templates	standard document templates
personal	personal favorites folder
programs	menu of all program folders (current user for NT4)
programs xxx	menu of programs folder xxx (eg Accessories)
Allprograms	menu of all program folders
Allprograms xxx	menu of programs folder xxx for All Users profile (NT4 only)

AllStart Menu	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)

Entering Format Information for Folder Contents Command

See Folder Contents Menu for an introduction. 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	Use **autocol n** to automatically start a new column every n entries; this gives the menu a toolbar look (applies to top level menu only, not submenus). Use **autosoftcol n** to automatically start a new column every n entries without including a bar between the columns (applies to top level menu only, not submenus). Use **autocolall n** to automatically start a new column every n entries; this gives the menu a toolbar look (applies to top level menu and submenus). Use **autosoftcolall n** to automatically start a new column every n entries without including a bar between the columns (applies to top level menu and submenus).
Text labels	Using **maxtext n** limits text labels to n characters. Using **omit** deletes the phrases in the "omit strings..." edit box on the special config tab; **omit** is applied before **maxtext**. Using **mne** in the menu box means Windows 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. Using **noext** in the edit box means file extensions will be removed from menu item names.
Position	Placing **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	Use **nosort** in the edit box so that the items will not be sorted. Using **sortext** in the edit box means items to sort by file extension. Put **sorttime** to sort most recently change files first.
Subfolders	Use **folderdots** in the edit box means "..." is added to folder names; this is useful with NoSubDir if you do not use icons in menus. Use **folderstart** in the edit box sorts menu entries with folders at start. Put **folderback** in edit box to add Back (previous folder) entry when NoSubDir specified. Use **nofolders** to omit all subfolders. Use **foldershortcut** to expand all folder shortcuts in the menu and **foldershortcuts2** to expand only those folder shortcuts with names ending in _x. Use **nosubmenu** in the edit box means all files from subdirectories will be listed in the main menu. Use **empty** in the edit box means empty folders will be included in the menu (normally, they are excluded). Use **nosubdir** in the edit box means no subdirectories will be included. The names of subdirectories are still shown; if selected, a *Folder Contents Menu is shown for that subdirectory.

Explorer

Use **nofiles** in the edit box means only folder will be shown and not files; useful with the **explorer** option to traverse large folder trees.

Place **explorer** in the edit box to add a menu entry "Explore" to all submenus; left clicking on it will open a single-pane Explorer window on the selected directory and right clicking will show an *Folder Contents menu for the folder (useful with **nofiles**). Uncheck "Switch to if active" to allow 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, eg

cmd "'c:\program files\2x\2xExplorer' '|*'"

Note single quotes which will be replaced by double when cmd run.

Place **explorer2** in the edit box to add a menu entry "Explore2" to all submenus; left clicking on it will open a double-pane Explorer window on the selected directory and right clicking will show an *Folder Contents menu for the folder (useful with **nofiles**).

Icons

Place **noicons** in the edit box to omit menu icons (only works if the Folder Contents menu is not embedded in another menu).

Place **back** in edit box to use default background from command list|setup.

Execution

Place ***all** in edit box to execute all commands, rather than displaying a menu.

Place ***allclose** in edit box to close all commands, rather than displaying a menu.

Place ***allclosefoce** in edit box to force closed all commands.

Embed

Place **embed** in the edit box is used if the *Folder Contents command appears in a menu: it causes the menu entries to be embedded within that menu rather than appearing when the *Folder Contents command is selected (embed must be in lower case).

Position

Place **center** in edit box to center menu on screen.

Place **under** in edit box to center menu under mouse.

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.

Placing **sortext** in the edit box means items will be sorted by file extension.

Fileman

Follow this keyword by a string which replaces "Explorer" in menus when explorer keyword used.

Exclude

Follow this keyword 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.

Command

Follow the keyword **cmd** by a the command to be executed; enclosed in double quotes. Use a | to indicate whether the selected file is to be placed in the command line. 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

Follow the keyword **zero** 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.

Extension

To include 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.

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 "*script runfile DoFiles =|" zero "FCount"
runs script file DoFiles for each file with x9 set to each file name in turn. No menu is displayed. Variable FCount is initialized to zero.

Using *Folder Contents Menu 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 Folder
Parameter: c:\
Format nosubdir autocol 16 folderback

The nosubdir keyword is also selected by checking "Show subdirectory only when parent entry clicked" on *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.

Another approach for large directory tree:

Command: *Menu Folder

Parameter: c:\
Format explorer nofiles

Shows a menu of all folders for drive C with single explore entry in menu for each folder. Left click on this entry to show files for that folder in Explorer window. Right click to show files in *FolderContents Menu.

The first technique shows the menu faster, but requires clicks to go up or down the folder tree.

Window-Specific Bar and Menu Contents

Purpose

You can specify that portions of a menu or a whole bar should only appear if specified window or program is active. Use the *Format Context and *Format EndContext built-in commands to do this.

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 keys to activate program features; *Format Context would be used to display the *Keys commands which were appropriate for the active program.

For menus only, you can also use any if command condition in the parameter edit box. For example,

Command *Format ContextIf
Parameter modem

Shows the menu portion only if a DUN modem connection is active.

The *Format Context the function can also used on a button bar to show or hide the bar depending on the active program. Such a bar could be attached to active window; different bars would then appear depending on which program is active. Each could contain commands relevant to the active program.

Configuration

To create a program-specific portion of a menu, you 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 for all windows from the program with .exe file exename (no path). Follow this command by the program-specific menu entries. End with the *Format EndContext command.

To create program-specific bars, put the *Format Context command as the command for left clicking the first button and enter the list of window captions to control when the bar is visible with this command. Do not use a *Format EndContext.

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

Item Name: Images
Command *Keys
Parameter ^i

Item Name: Back
Command *Keys
Parameter %{al}

Menu Item Name: End
Command *Format EndContext
Parameter

You cannot use these commands in menus attached to clicking on the desktop as the desktop will be the active program in this case.

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 View|Options.

If you then check "Track Explorer" on the Setup configuration dialog, Windows PowerPro will remember the last 32 file folders that you open with Explorer, PowerDesk, or 2XExplorer. Activating the command

Command *Menu Explorer
Parameter

displays a menu these folders sorted by path, last accessed, or drive.

To clear the list of explorer windows, use *Exec ClearRecentExplorer.

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.

Right clicking on the menu with shift or ctrl down will remove the selected entry from the recent explorer list.

To clear the list of explorer windows, use *Exec ClearRecentExplorer.

Windows PowerPro will remember whether you used a single or double pane Explorer window and use this configuration. If you wish, you can force a **s**ingle pan window by holding down **s**hift when you select a folder name from the menu, or you can force a doub**l**e pane window by holding down ctr**l**.

To create a menu of favorite folders, create a command list with a set of commands like this

Command: *Exec Explorer

Parameter c:\the\folder\path

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 Explorer, you may prefer

Command: c:\windows\explorer

Parameter /e,/select, c:\the\folder\path

` Omit the /e for a single pane window. This will produce the same result as *exec explorer.

Displaying a Message with *Message

Use the *Message 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.

If you find that the *message if not taking the focus when you want it to, try using the On Top checkbox or putting *takefocus at the end of the message.

Accessing the Screen Saver with *ScreenSaver

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 (see below)
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	changes saver to specified file
ChangeTimeout	changes saver timeout to specified value (in minutes)
ChangeRestart	set, clear, or reverse setting of restart saver when changed setting on GUI Control dialog

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.

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 an 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
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

Command Scripts

There is a tutorial on scripting in the file PPST 1.01.chm in the PowerPro folder.

Purpose

You can run all the commands on a command list or a file with a single *Script command. You can program scripts with variables, if, and jump commands to wait for some condition or to create loops and conditionally execute commands.

You can store scripts in command lists or in files, but you will usually find it easier to work with scripts stored in files

The commands in a script can be

Windows commands, like c:\windows\explorer.exe
PowerPro built-in commands, like *Wallpaper
Script programming commands, like *script if()do

The actions associated with the *Script command as follows:

*Script Assign v expression	Assigns a string or number to a variable
v=expression	Assigns a string or number to a variable
*Script Run commandlist	Runs a script from command list (only left commands are used). You can also use the syntax run.script(args) to run command list script with argument args. You can use this syntax alone on the command line or as part of an expression.
*Script RunFile	Runs a script from a file. You can also use the syntax runfile.script(args) or .script(args) to run file script.txt or script.powerpro in the scripts subfolder. You can use this syntax alone on the command line or as part of an expression. You can use runfile.script@lab(args) or .script@lab(args) to start

	execution of the script at label lab (no spaces around @)
*Script If	Test an expression to control whether next command is executed
*Script If()do	Test an expression to control whether a block of commands is run
*Script Else	Used with *Script If()do
*Script EndIf	Used with *Script If()do
*Scrip For	Starts a for loop
*Script Endfor	Ends a for loop
*Script Break	Break out of a for loop.
*Script Jump	Jumps to a label in a script.
*Script Quit	Ends execution of a script.
*Script Flag	Assign a value to a flag.
*Script Debug	Writes following text to a debug window
*Script Close commandlist	Ends all programs listed in command list
*Script CloseForce	Ends all programs listed in command list, forces them closed, possibly losing data.

You can use the syntax

run.script(args) to run a command list with up to 8 comma-separated arguments. You can use this form either alone on a command line or in an expression. For example,

getmax=run.max(alpha,beta,gamma)

would set getmax to result of calling command list max. Similarly, use

runfile.script(args) or .script(args)

to run file script.txt (or script.powerpro). This syntax is the simplest way to pass arguments to your scripts; you access them in the script with the arg(n) function. You can use runfile.script@lab(args) or .script@lab(args) to start execution of the script at label lab (no spaces around @). You can use runfile.dir\script or .dir\script to run file script from subdirectory dir of script folder.

You will usually find it more convenient to work with files when creating scripts of any complexity. Files are easier to type. You can use indenting to show program logic. You can omit the *Script when entering *Script commands and you can omit the leading * from all commands. You can add comments.

You can use variables outside of scripts to insert text into commands by defining the expression insertion character using Setup|Advanced|Chars. For example, if the insertion character is & and you executed

*Assign fpath "c:\thefolder\"

then running

*Notepad &(fpath)file1.txt

would cause Notepad to open c:\thefolder\file1.txt.

Once you define the insertion character, you can temporarily suspend its special meaning by preceding it with a quote, e.g.

*Message put '& in message

displays

Put & in message.

You do not use the expression insertion characters in assignment, if, and for statements. If you do, it will be ignored, unless you have checked "process expression character" on setup|advanced|characters.

Windows PowerPro normally starts executing the script with the first command on the command list. But if you put the command list item label of a command list item after the command list name (preceded by @), PowerPro will start at that entry:

Command *Script run

Command List mymenu @cmd2

runs all commands on mymenu starting at the one labeled cmd2. Put the command list name in double quotes if it contains spaces.

For convenience, you can preset variable x9 for the script by putting the value for x9 after an equal sign (=);

Command *Script run

Command List mymenu =value for x9

This is often used in a cmd option for *Menu Folder or with an explorer context menu; for example

*Menu

Folder

c:\dir

cmd "*Script runfile aScript ="

which runs scriptfile aScript with whatever file is selected.

You can use the wait command in a script started with *Script call to wait for some condition. For example, you could run a dialer, wait for the modem, then run a program which accesses the modem; see below for a sample of a script which does this.

Variables

PowerPro lets you create variables to hold text or numbers and be used in expressions. Variable names must start with a letter and consist of up to 23 letters and digits and underscores. The case of letters is ignored. You cannot use any of the following as variable names:

mci, not, andom,vdeskempty,mounted, length, anywindow, visiblewindow, activewindow, validpath, input, inputcancel, timer, timerrunning, eval, run, 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 the *assign statement and an expression or var=expression:

For example

var1= "abc"

*assign 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). Note that the *Script has been omitted from the *assign samples as you can do in script files (but not with command lists).

If you do not assign a value to a variable before you use it, the variable is initialized to the empty string ("") which is treated as zero in arithmetic or logical expressions.

You can use variables outside of scripts to insert text into commands by defining the expression insertion character using Setup|Advanced|Chars . For example, if used the insertion character & and you executed

Path = "c:\thefolder"

then running

Notepad &(path)\file1.txt

would cause Notepad to open c:\thefolder\file1.txt.

You can also use the insertion character to insert an expression directly into a command:

Notepad &(date select -6).txt

opens a file with the same name as the last 6 digits of the current date.

You do not use the expression insertion characters in assignment, if, and for statements. If you do, it will be ignored, unless you have checked "process expression character" on setup|advanced|characters.

Conditional Statement If and If()do/Else/ElseIf/Endif

Use conditional statements if and if()do to control which part of a script gets executed. For

*If (expression)

the statement following the if is skipped if the expression evaluates to 0 or the empty string "". Or if the *Script if is followed by More Commands, these commands are skipped rather than the next statement if the if is false. Note that the expression must in parentheses. You can only use *if()do inside of a file or command list script; it will not work in More Commands. You can also use *Script with no Do outside of a script to control following More Commands.

To control more than one statement, use

*If (expression) do
statements
*elseif (expression)
statements
*elseif (expression)
statements
*else
statements
*endif

Note keyword do following the if. 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/else are executed instead. Otherwise, the statements following the if()do are executed then the statements between the else/elseif and 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
        *assign 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
        *debug a is 1
*elseif (a eq 2)
        *debug a is 2
*elseif (a eq 3)
        *debug a is 3
*else
        *debug a gt 3
*endif
```

Selects one of the *debug statements depending on the value of a.

For, Endfor, Break Statements

Use the For statement to create a loop. The statements

```
for (var=initexpr; testexpr; var=increxpr)
statements
endfor
```

are exactly equivalent to

```
var=initexpr
@loop
if (testexpr) do
  statements
  var = increxpr
  jump loop
endif
```

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 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.

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.

Quitting Scripts

Windows PowerPro normally executes all commands until the end of the script, but you can stop execution by the command:

Command	*Script
Action	quit

Follow by the keyword all to quit any calling script 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

Command	*Script
Action	jump
Parameters	xxx

to go to label xxx of the currently executing script for the next command. For scripts in command lists, put the label in the item name field. For scripts in files, create a label xxx by starting a line with @xxx. The rest of the line can be empty or can contain a command.

Flags

To help with Script programming, Windows PowerPro has a set of 32 flags which you can manipulate and test. To set a 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 toggle n1 n2**. Use **0 31** for **n1 n2** to access all flags.

You can test the flag with the **if** command; use

Command	*Script
Action	if
Parameters	(pproflag(4))

to check to see if flag number 4 is set; use (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.

Expressions

Purpose

Use expressions in PowerPro to compute a number or a string.

You can use an expression to assign a value to a variable with var=expression or *Script assign. You can also use an expression as the condition in a *Script If()do statement or a *Format

ContextIf. And you can insert an expression directly into a command with the variable insertion characters (say &). For example,

x = 5 + 2 *length "abc"

assigns 11 to variable x (i.e. 5 + 2*3).

str= select ("abcdef" ,3) ++ "123"

assigns "abc123" to variable str.

*script if (alpha >= 5 or select(beta, 2) == "ab")) do

executes the statements following the if-do if the expression is true (value 1).

Normally you must use an expression as part of another statement, such as an assignment or an if statement. But PowerPro will allow the following expressions to be used alone as statements:

expressions starting with do, eval, messagebox, mci, inputdialog;

expressions starting with a plugin call.

Structure of Expressions

As can be seen in the examples, an expression is a series of values and operators. The expression is evaluated 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).

PowerPro can work with either strings or numbers in expressions and will try to use whatever is appropriate to the operator. For example, "-12" + 9 yields the number 3. "-12" join "ab" yields the string "-12ab".

Values

In PowerPro, 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; only whole numbers (integers) can be used

a variable, like a or x4; see programming for more on variables

a special keyword, like date; see below for keyword details

a function call, like length(string) or window("caption", ="notepad")

a plugin call, like date.add(date, 7)

a call to run a script, like run.script(args) or runfile.script(args) or .script(args)

You use the character ' as an escape character in strings in expressions. First, check "Use quote ' for escape in expression strings" on Setup|Advanced |Chars. Then you can use the following sequences in PowerPro strings.

'n	newline
'r	carriage return
't	tab
'"	double quote
''	single quote
&	expression follows character (replace & by whatever character you use)

Keyword Values

Powerpro lets you use any of the following keywords in an expression to get at system values:

date	Todays date as 8 digit string yyyymmdd (e.g. 2000 12 31)
shortdate	Todays date in control panel|regional short date format
longdate	Todays date in control panel|regional long date format
xtime	Time in control panel|regional time format
time	Time as 6 digit string hhmmss, 24 hour clock, eg. 140515.
timesec	Number of seconds since midnight January 1, 1970
dayofyear	Day of year (eg 35 is Feb 4).
dayofweek	Day of week as single digit (Sunday is 0, Monday 1, etc.)
timezone	Name of current time zone.
uptime	Number of seconds Windows has been running
ctrl	Value 1 if ctrl key id down, 0 otherwise.
shift	Value 1 if shift key is down, 0 otherwise.
alt	Value1 if alt key is down, 0 otherwise.
win	Value 1 if apps key is down, 0 otherwise.
mouseleft	Value 1 if left mouse button is down, 0 otherwise.
mousemiddle	Value 1 if middle mouse button is down, 0 otherwise.
mouseright	Value 1 if right mouse button is down, 0 otherwise.
pprofolder	Folder of PowerPro configuration (including terminating \)
pproversion	Version of powerpro as four digit integer, e.g. 3310 or 3400.)
disk	Letter of disk where PowerPro as run from (e.g. c)
deskname	Name of current virtual desktop
desknum	Number 1-9 or current virtual desktop.
deskempty	1 if current vdesk empty; 0 otherwise
currentdir	Path to working folder of current foreground window
Subbarname	Name of current subbar
caption	Caption of current foreground window
captionunder	Caption of window under the mouse
exefullpath	Full path to exe name for current foreground window
exefilename	File name (no path, no .exe) to exe file for current foreground window
dunidle (95/98)	Number of seconds since last characterreceived over dial-up
dunrate (95/98)	Dial up rate in characters per second
pmem	Percent free memory
cpu	Cpu usage (98 only)
gdi	Gdi resources (95/98 only)
user	User resources (95/98 only)
threadcount	Number of active threads
processcount	Number of active processes
cdcurtrack	Current CD track
cdlasttrack	Last CD track
defaultprinter	Name of default printer

keylog	"X" if logging; zero otherwise
keylogfile	Name of currently open key log file; empty string "" otherwise
lastidletime	Length of last idle period in which an alarm occurred
clip	First line of clipboard; only first 263 charaters of first line are returned. Use clip plugin for unlimited clipboard access.
lastclippath	Full path to last captured clipboard item
lastclipname	File name of last captured clipboard item
lastactivehandle	Window handle to last window selected by active button
lastautorunhandle	Window handle to last window selected by autorun item
cliptrackon	Set 1 if clip tracking is on; 0 otherwise
xmouse	Horizontal mouse position in pixels (0 for extreme left of screen)
ymouse	Vertical mouse position in pixels (0 for extreme left of screen)
xscreen	Screen width in pixels.
yscreen	Screen height in pixels.
paper	Wallpaper file name
saver	Screen saver file name
saveractive	Value 1 if screen saver active; 0 otherwise.
batterypercent	Percent battery power left
volume	Current master volume (0 - 255); can be set by *Exec VolumeAll
muted	Set 1 if muted sound; set 0 otherwise.
acdc	"A" if running on ac power; "D" otherwise
recycleSize	total size of recycle bin on all drives in KB
recycleItems	total number of items in recycle bin on all drives
modem	1 if dial-up connection active; 0 otherwise.
browserURL	URL in current browser window
browserDomain	Domain in current browser window
browserSubdomain	Domain and subdomain in current browser window
inputtext	Display input dialog; same as function input("text") except you cannot set the title.
inputpath	Displays file browse dialog and returns results. Clear flag 0 if cancel pressed; sets it otherwise.
inputfolder	Displays folder browse dialog and returns results. Clear flag 0 if cancel pressed; sets it otherwise.
inputcolor	Displays color dialog and returns results as single number (eg for *Desktop IconTextColor). Clear flag 0 if cancel pressed; sets it otherwise.
inputdate	Displays calendar calculator dialog and returns selected date as 8 digits yyyymmdd. Clear flag 0 if cancel pressed; sets it otherwise.
inputdatetime	Displays calendar calculator dialog and returns selected date and time as 14 digits yyyymmddhhmmss where 24 hour clock is used for time. Clear flag 0 if cancel pressed; sets it otherwise.

For example,

x = ReplaceChars(browserDomain, "._")

assigns x the Browser Domain with dots replaced by underscores.

minutes = (time /100)%100
assigns minutes the current number of minutes in the time.

Operators and Functions

Powerpro has arithmetic, comparison, string, and special operators and functions. You can also call plugins from expressions.

The following functions which take one following operand, like mci("string") or input("title"). The parentheses are optional for functions with one operand, so you can write input "title" or mci "string" as well.

mci	Execute mci command; see section on mci below for details
not	Result is 1 if following expression is zero or empty string; 0 otherwise.
arg	arg(n) returns nth argument in call to script using runfile.script(arg1,arg2,…) or run.script(arg1,arg2,…). arg(1) is first argument.
eval	Evaluates following string as an expression and returns results. Eval allows you to build expressions as strings and then evaluate them t.
random	Random n returns a random interger between 1 and n.
vdeskempty	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.
mounted	Result is 1 is following string's first letter is removable disk and is mounted; 0 otherwise.
length	Compute length of following string; see section on strings below for details.
filemenu	Displays a file as a menu, returning the selected item's text. See section on filemenu in *Keys for details of file layout.
anywindow	Result is 1 if following string is a caption list which matches any open window (visible or not); 0 otherwise.
visiblewindow	Result is 1 if following string caption list matches any open visible window; 0 otherwise.
activewindow	Result is 1 if following string caption list matches the foreground window; 0 otherwise.
validpath	Value is 1 if following string is validpath to file and 0 otherwise. String may contain wildcards (* and ?).
env	Value of environment variable named by following string.
input	Prompts for input using title for input dialog given by following string. 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	Same as input, except that if cancel is pressed then a running script and any calling script are immediately ended.
pproflag	Usage pproflag(n) returns value of nth flag.
timer	Value of timer in seconds; first letter of following string determines timer, e.g. timer "c" for timer c. You can also use number 1 - 26, e.g. timer 14.

timerrunning	Usage: TimerRunning(t). Returns 1 if timer running; 0 otherwise. First letter of following string determines timer, e.g. timerrunning ("c") for timer c. You can also use number 1 - 26, e.g. timerrunning(14).

PowerPro also has operators like addition and comparison, as listed in the following table.

*	Multiple two numbers. 5*2 is 10.
/	Divide; result is truncated to nearest integer. 12/5 is 2.
%	Remainder after division; 15 % 4 is 3.
+	Addition. 2 + 2 is 4.
-	Subtraction. 15 - 8 is 7.
++	Join two strings. "abc" ++ "defg" is "abcdefg"
== eq	Result is 1 if values are equal; 0 otherwise. Case of letters strings is ignored in all comparisons, e.g. "abc" is equal to "AbC".
!= ne	Result is 1 if values are not equal; 0 otherwise.
< lt	Result is 1 if first value less than second value; 0 otherwise.
<= le	Result is 1 if value less than or second equal to value 2; 0 otherwise.
> gt	Result is 1 if first value greater than second value ; 0 otherwise.
>= ge	Result is 1 if first value greater than or equal to second value; 0 otherwise.
& and	Result is 1 if both values are 1; result is 0 otherwise
| or	Result is 1 if either value is 1; result is 0 otherwise

The grouping in the above table shows the priority of the operators. Operators nearer the top of the table have higher priority. For example, * priority is higher than + and -; in turn + and - have higher priority than the string operators which have higher priority than 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 expression follows character.

Operators consisting solely of alphabetics, like *and* or *not*, must be preceded and followed by a blank when used.

Finally, PowerPro has functions of two operands, such as select("string", 5) and case("upper, "string"). Functions with two operands must always be followed by a left parenthesis; the two operands are separated by a comma. You can nest function calls, such as

select (case("upper, "string"), 2)

The following table shows the functions with two operands:

max	Maximum of two values eg. max(x0, 7)
min	Minimum of two values eg min("abcd", vstring)
do	Uses three arguments: do(cmd, param, dir) and runs the command stored in cmd with parameters stored in param and work directory stored in dir. You can used with file commands (.exe) or with PowerPro * commands, like *script. For * commands, use the cmd to store the * command name, the param for the action and any parameters, and the dir for any keywords. Example: do("notepad.exe", "c:\file", ""). Example: do("*menu", "show test", "center"). The do function provides an alternative to using the expression follows character to create commands, e.g. use do("notepad.exe", thefile,"") instead of notepad.exe &(thefile). The function do can be used alone on a command line. If used in an expression, it always returns 1
assign	Assigns second string to variable name give 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).
word	Usage: word ("string" , number). Scans the string and returns the word given by number, starting at 1. A word is any sequence of non blank characters. If number is negative, starts from end of string. word("aaa b$ ccc" , 2) is "b$" and word("aaa b$ ccc", –3) is "aaa".
Join	Join two strings. join ("abc", "defg") is "abcdefg". Same as ++ operator.
inputdefault	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 "".
inputdialog	Shows a dialog to input up to 6 variables; see below for details.
messagebox	Shows a message box; see below for details.
formatdate	Formats a date; see below for details.
formattime	Formats a time; see below for details.
diskspace	Diskspace ("kfree", "c:") returns free kilobytes on drive c. You can 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..
ifelse	The first string should contain a comma; result is everything up to this comma if second string is not zero or empty and everything after comma otherwise; e.g. ifelse (10<20,"alpha,omega",) is "alpha". You can also use with three operand: ifelse(expr, s1, s2) returns s1 if expr is not 0 or "" and s2 it if is.
if	The result if the first string if second string is not zero or empty; result is "" otherwise. .
remove	Remove characters from string; see section on strings below for details.
select	Select characters from string; see below for details.
index	Index of second string within first; see below for details.
revindex	Index of last occurrence of second string within first; see below.
fill	Combines strings; see below for details.
case	Changes string case; see below for details.
replacechars	Replaces character(s) by specified replacement character.

repeat	Returns string formed by repeating first argument number of times given by second.
window	Determine window coordinates; see below for details.
readline	Usage: readline("filepath", n) reads the nth line from text file filepath and returns this string; Line numbers start at 1. e.g. x1=3 line=readline ("c:\path\path.txt",x1) assigns third line of file to line. Flag 0 is cleared by readline if a line is read and is set if end of file is reached before desired line number.
vdeskhaswindow	Usage: vdeskhaswindow ("vdeskname","captionlist") returns 1 if named desktop has window matching captionlist.. Vdeskname can be a single digit n for the nth desktop, a desktop name, "" for current desktop, 0 for list of locked windows. For example, vdeskhaswindow("misc" , "*internet explorer") returns 1 if internet explorer window open on desktop misc.

String Operators

PowerPro has operators to help you work with strings of characters. PowerPro supports strings of any length.

Length returns a number representing the length of a string: length ("abc") is 3.

Join or ++ is used to join one string to the end of another: "ab" ++ " 1" is "ab1"

Remove is used to remove characters from a string. Specify remove(string,number). For example,
remove ("abcd" , 2)
is "cd". Use a negative number of remove characters from the end of a string. For example, remove("abcd", -3)
is "a".

Select is used to select characters from the start, end, or middle of a string. Specify as
select(string,number). 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'. Finally, 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".

Index is used to find one string in another, ignoring case of letters. The operator index returns 0 if the second string is not found. If it is found starting at the nth position, index returns n. For example,
index("to be or not to be", "be")
returns 4.

RevIndex is used to find last occurrence one string in another, ignoring case of letters. The operator index returns 0 if the second string is not found. If it is last found starting at the nth position, revindex returns n. For example,

revindex("c:\path\to\filname.txt", "\")

returns 11 (index of last backslash).

Fill is used to pad a number or string using a padding string. The expression

fill(string, number)

creates a string of length no less than the first string with the ending characters set by the second number or string. For example,

fill ("0000",12)

yields "0012" and

file("***///", "a")

yields "***//a".

Case is used to change the case of a string. The first 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".

The string "acronym" creates an acronym from the first character of each word:

timezone = "Easter Standard Time".

case ("acronym", timezone)

would yield "EST.

ReplaceChars is used to replace character(s) by a given 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 characters. All occurrences of any other characters in the second string which occur in the first string are replaced by this final character. For example, the result of

replacechars ("xx.x.abc=*", ".=_")

is "xx_x_abc_*". The _ has replaced all occurrences of . or =. Unlike other string operators, ReplaceChars is case sensitive.

MessageBox

The messagebox operator shows a message box with up to three buttons and returns a result indicating which button was pressed. The form is:

messagebox ("layout", "text")

where "text" is the text for the messagebox and "layout" contains keywords specifying the buttons and icons for message box. 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

For example

messagebox("yesno question" , "continue with script?")

displays a yesno messagebox with a question mark icon. The result depends on the button pressed.

Cancel=0, OK=1, Abort=3, Retry=4, Ignore=5,Yes=6, No=7.

InputDialog

The InputDialog operator displays a dialog with up to six edit boxes, check boxes, or combo boxes to set the values of up to six variables. You can also set the variable to a folder name or a file name. The form is:

inputdialog"("var1=title1, var2=title2, var3=t3, var1=t4, var2=t5, var3=t6", "caption for dialog")

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", is initialized to the variable x1, and is used to reset variable x1 when OK is pressed. The second box has title "Value here" and uses variable beta.

To display a check box rather than an edit box, add two question marks (??) to the end of the title; that is use var=check 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 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.

The result of the operator is 1 unless cancel is pressed in which case it is 0.

FormatDate and FormatTime

Formatdate returns a formatted date. Usage is
formatdate ("format string",yyyymmdd)
where yyyymmdd is 8 digit date. The 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 special characters like / or : in the string. For example,
formatdate("dddd, YYYY/MMM/dd", "20020822")
returns "Thursday, 2002/Aug/22" (for English locale)

Formattime returns a formatted time. Usage is
formattime("format string",hhmmss)
where hhmmss is time using 24 hour clock The 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

For example,
formattime("h:mm:ss tt" , "140102") returns "2:01:02 PM".

VisibleWindow, AnyWindow, ActiveWindow Operators

These three operators must be followed by a string which is interpreted as a caption list. For visiblewindow, the result is 1 only if a matching, visible window exists. For anywindow, the matching window can be hidden. For activewindow, the matching window must be the foreground window. For example,
*if (activewindow "=explorer")
checks to see if explorer is the active window. Another example:

v = not (visiblewindow "*notepad*,*internet explorer")
assigns v 1 only if neither notepad nor internet explorer is running in a visible window.

Window Operator

The window operator lets you get at the position, size, min/max/visible/topmost state, caption, exename, exe full path, window class, and window handle of any window on the screen. Use
window ("pos", "caption")
where pos is one of "left", "top", "bottom", "right", "height", "width", "minimized", "maximized", "visible", "topmost", "caption", "exename", "exefullpath", "class", "anywindow", "visiblewindow", "class", , "anywindow", "visiblewindow", "firstwindow" and caption is a caption list string that matches the caption of 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.

See section on window handles for information on "anywindow", "visiblewindow", "firstwindow".

For example,

window ("right","*explorer*")

returns the coordinate of the right side of the first window PowerPro finds with explorer in its caption. For example,

window("caption", "=notepad")

returns the caption of the first window found from notepad.

Help on Expressions

Note: The PowerPro folder contains a file expressions.txt which you can use to help remember the format and definitions of keywords and operators. Define a hot key to execute the command

*Keys

{filemenu expressions.txt}

This will display a menu of keywords and operators; select one to add it to currently open editing window. You can edit expression.txt if you like to change the layout of the menu.

(Note: older versions of PowerPro used an operator syntax for functions of two variables, such as "filepath readline n. This will still work but the function syntax is preferable).

Sample Script

Here is a sample of a script which uses *wait to wait on the status of both the modem and a program and uses *Windowto terminate a program.

Starting the script uses the Dundial program, described in dundial.txt in the Windows 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 Microsoft 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 them menu is created and saved, the script can be run with the command:

Command	*Script
Parameter	run internet

Here are the entries for menu **internet**.

Item Name	Dial connection
Item Command	c:/program files/PowerPro/dundial.exe
Item Parameters	DunName UserName Password

Item Name	Wait for connection
Item Command	*wait
Item Parameters	modem

Item Name	Start SpeedUp program
Item Command	c:/program files/speedup/speedup.exe
Item Parameters	

Item Name	Start explorer
Item Command	c:/program files/Internet Explorer/IExplorer.exe
Item Parameters	

Item Name	Wait for explorer to be terminated
Item Command	*wait
Item Parameters	nopath c:/program files/Internet Explorer/IExplorer.exe

Item Name	Hangup connection
Item Command	c:/program files/PowerPro/dunhang.exe
Item Parameters	*

Item Name	End SpeedUp program
Item Command	*Window
Item Parameters	close =c:/program files/speedup/speedup.exe

In a file, the script would look like this:

```
"c c:/program files/PowerPro/dundial.exe" DunName UserName Password
*wait modem
"c:/program files/speedup/speedup.exe"
"c:/program files/Internet Explorer/IExplorer.exe"
*wait nopath c:/program files/Internet Explorer/IExplorer.exe
"c:/program files/PowerPro/dunhang.exe"
*window close =c:/program files/speedup/speedup.exe
```

A problem with this script is that, since a *wait executes in the background, you will not be able to run another script file (due to PowerPro restriction that script files cannot be run while another script file is doing a *wait). Another approach which avoids this problems is to remove the *wait. Replace it by *Timer Start a, which starts a timer (say) a with a reset of (say) 2 seconds; the reset command should be

```
Command:    *script
Action:     if
Parameter:  (not visiblewindow "*internet explorer*")
More:       *Timer stop a
          "c:/program files/PowerPro/dunhang.exe"
            *window close =c:/program files/speedup/speedup.exe
```

This uses the ability of the *Script if to control the More Commands.

Running a Script from a File

You can create a command script and store it in a file and the run the file with

```
*Script
RunFile
c:\path\filename.txt
```

If you specify a file name without a path, then the file is assumed to be in a subfolder **scripts** of the folder of the PowerPro configuration file. If the file name has no extension, then .txt is used or .powerpro is used; PowerPro looks for both.

You can also use

runfile.filename(arg1,arg2) or .filename(arg1, arg2)

to run filename.txt in the scripts folder; you access the arguments in the script with the function arg(n).

Each line in filename.txt is a PowerPro command or Windows program/document to run. For example:

```
c:\windows\notepad.exe
wait visiblewindow *notepad*
keys abc
```

would run notepad, wait for a notepad window to be visible, and then send the keys abc.

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.

For example,

```
if (arg(1)<=1)
   quit(1)
quit (.factorial(arg(1)-1)*arg(1))
```

return the factorial of its argument.

Any line starting with a semi-colon is ignored and can be used as a comment. Blanks at the start of a line are ignored and can be used to indent to show structure.

If the file name of the program you want to run from a script contains blanks, put it in double quotes:

"c:\program files\editor\editme.exe" param

You can put a label on a line by preceding the label with an @

```
c:\windows\notepad.exe
wait visiblewindow =notepad
@loop *keys a{enter}
      *script if (not ctrl)
jump loop
keys b
```

This file 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.

You can tell PowerPro to process two lines as one by ending the first with a single quote then a plus ('+).

You can include work directory information for a command by preceding it by the characters !`

C:\windows\notepad.exe !`c:\mytext.

starts notepad with work directory c:\mytext. Also use this approach to enter format keywords for *Menu commands.

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)

For example

c:\windows\notepad.exe edme.txt !`d:\mydir *min

starts notepad minimized on file edme.txt with starting folder d:\mydir.

You can run other scripts by Runfile within a script file.

Except for *wait sleep, you can only use *wait commands in the outermost script; if a script is called from another script, then the only type of *wait it can contain is a *wait sleep.

You cannot put multiple commands on one line in a script file.

In general, how do you put PowerPro commands into Script files? There are two cases: running Windows files/programs and running PowerPro asterisk commands (like*Menu or *Keys).

For files and programs, the PowerPro command entry controls generally appear as

Command:	c:\program file\cccc\cccc.exe
Show at Start	Normal
Parameters	xxxx /switch
Initial Dir	c:\thedir

You enter this on one text line in a script file as

"c:\program file\cccc\cccc.exe" xxxx /switch !`c:\thedir

Explanation: put the command in double quotes, then a space, then the command parameters. If you need to specify an initial directory, put !` then the directory information. You cannot specify show at start information (it is forced to be normal). (The double quotes around the command are unneeded if the path does not contain spaces).

For PowerPro asterisk (*) commands, the command entry controls will be something like

Command	*Menu
Action:	Show
Edit	mymenu
Second Edit	centerScreen

This is entered in a script file as the line

*Menu Show mymenu !`centerScreen

A few commands (eg Keys) have no action; for them, just put the command followed by the edit info (e.g *Keys abc). Most commands do _not_ have the second edit in which case the !` and the following text are omitted.

PowerPro has an an approach to importing and exporting command lists which works well with scripts. When exporting, if you check "Export left command only" Powerpro will export a file with one line for command list item which can be easily edited. You can also import scripts using this format; to do so you must precede the command list items by

[**name]

where name is the command list name. Note that two asterisks must precede the command list name.

Note: if labels or commands contains blanks, they must be enclosed in double quotes. Also, PowerPro will automatically remove *Script when exporting these files and insert it when importing these files.

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 program. You can also use this command to hide these tray icons and still access the commands by simulated mouse clicks.

This lets you decide how to access tray icon functions and which tray icons should appear in the tray window on your taskbar.

Configuration
Before you can access a tray icon, you must train Windows PowerPro on how to access the icon. . You have to train Windows PowerPro once for each icon you want to access.

Once you have trained Windows PowerPro, you send mouse clicks to the icon with the following command:
Command *TrayIcon
Acton click
Parameters icon_name keystrokes
click is one of leftclick (left click), leftdouble (left double click), etc.
icon_name is the name you assigned to the icon when you trained Windows 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 icon.

Example
Command *TrayIcon
Action leftclick
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

Training PowerPro to Recognize Tray Icons from Other Programs

To access tray icons from other programs, you must first train Windows PowerPro to recognize the hidden window and internal codes that this icon uses. Follow these steps:

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 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 Windows 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 Windows PowerPro will simulate a right click on the icon as a test.
6. If the right click test succeeds, Windows PowerPro will ask you to enter a name for the icon information. This is the icon_name field used in the *Tray Icon command or selected from the drop down in the command wizard.

If Windows 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.

Changing the Wallpaper with *Wallpaper

Use *Wallpaper to change the wallpaper. You can also change the saver from the media dialog.

Windows PowerPro allows you to use .jpeg and .jpg files as wallpaper.

These are the *Wallpaper actions:

Show	changes wallpaper to indicated file but does not store the file name
Change	changes paper to file in same folder; you can specify random or sequential change
ChangeTo	changes paper to indicated file
Style	Enter center, tile, or stretch as parameter to set wallpaper style. Stretch may not work on your system

Virtual Desktops

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 command. Only windows on the active virtual desktop are visible.

When you shutdown Windows 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 "NewFromList" or "ReplaceByList" actions or you can specify the initial contents of a desktop using the configuration Desktop tab.

Configuration

You define and switch-to a virtual desktop in two ways: by activating the built-in command *Vdesk or through a menu that you access by Shift+right-clicking anywhere on a Windows PowerPro button bar.

Use the menu to switch desktops, create new desktops, lock/unlock windows on desktops, move windows between desktops, close and rename desktops.

In addition to the menu, you can also use the *VDesk command to work with desktops, by associating 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.
Menu	Displays the virtual desktop menu.
Next	Activates the next virtual desktop.
Previous	Activates the previous virtual desktop.
MoveActive	Moves active window to named desktop (which must already exist).
MoveAutorun	Moves last autorun match to named desktop (which must already exist)
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.

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).

Further Information

You can have up to 9 active desktops.

You can show the name of the current desktop as a button label.

It is possible to show a different Windows PowerPro bar for each desktop. Create new bars and start them with the desktop you want them to be associated with (use *Bar Show to show a bar). Make sure "All Vdesks" on the Bar 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.

Explanation of 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. The 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. The 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 Virutal Desktop Setup dialog.

Remove From Desktop
Shows a list of windows. Selecting one removes it from the current desktop.

Move/Copy from this
Shows list of of windows. Selecting one causes 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 windows 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 close 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. 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 new name to 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, eg **(mydesk)**; other desktop names are shown in angle brackets, eg **<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, eg **(mydesk)**; other desktop names are shown in angle brackets, eg **<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.

Virtual Desktop Setup

Use Virtual Desktop Setup dialog which you access from the Setup dialog to specify characteristics of virtual desktops.

Normally, Windows 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" Shutdown and restart PowerPro after changing this option.

If you activate a window which is on a hidden desktop (eg via tray icon), Windows PowerPro can be configured to show and switch to the hidden desktop. If you want this feature, check "Show Virtual Desktop if any of its windows is activated".

You can specify that PowerPro should move a window activated 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.

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, Windows PowerPro will execute the script of the same name for the desktop.

Check Press bar button with *VDesk SwitchTo xxx as command when xxx is active to have PowerPro show a button corresponding to the active desktop as pressed. Assign the command *Vdesk SwitchTo xxx *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.

You can specify the name of a command list to be run after each time you switch to a new desktop. The command list can access the new desktop name, if desired, with *Script if (vdeskname=="string") or *Script assign v desktop. 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 locked on all desktops.

Initializing Desktops Using the Configuration 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" is not checked, and make sure "Show on all Vdesks" of 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.

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.

Wait Command

Use the wait command in multiple commands or in when executing all commands on a menu in order to wait for some condition before executing some of the commands.

Except for *wait sleep, PowerPro can still be used while a wait is underway. For *wait sleep, PowerPro will be unresponsive until the wait ends. However, *wait sleep has two advantages: it can be used with short waits, and can be in any script (other waits are restricted to the outermost script and cannot be used in a script called by another script).

Wait 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. Unlike all other *wait commands, *wait sleep can be used an any script.

Wait for n seconds:

Command: *wait

Parameter: n

where n is any number waits for that number of seconds.

Wait until command is ready for input:

Command: *wait

Parameter: ready caption_list

The caption_list is optional. If omitted, PowerPro waits until the last file launched by PowerPro is ready for input up to a maximum of 10 seconds. If the caption_list is present, PowerPro waits until any window selected by the caption list is ready to accept input up to a maximum of 10 seconds.

Wait until command exits

Command: *wait

Parameter: done captionlist

The caption_list is optional. If omitted, PowerPro waits until the last file launched by PowerPro exits. If the caption_list is present, then a windows which matches the list must be visible, and PowerPro waits until any program with a window which matches the list exits.

Wait for modem to be connected (Dial-Up Networking RAS connection only):

Command: *wait

Parameter: modem

You can also put a number ahead of the word modem; PowerPro will wait for either that number of seconds, or until the modem is connected, whichever is smaller. For example, "8 modem" waits for up to 8 seconds or until the modem is connected.

Wait for modem to be disconnected (Dial-Up Networking RAS connection only):

Command: *wait

Parameter: nomodem

Wait until window with specified caption is active (foreground):

Command: *wait

Parameter: active xxx

waits until any program with caption xxx is active (foreground). Put caption in double quotes if it contains blanks. Use xxx* for captions starting with xxx, *yyy with captions ending with yyy, and *zzz* for captions containing zzz anywhere. You can use multiple captions separated by commas. Use =progname for any window from program with exe file name progname (no path, no .exe). You can put a number n ahead of the caption to limit wait to n seconds. .

Wait until window with specified caption is not active (foreground):

Command: *wait

Parameter: noactive xxx

waits until any program with caption xxx is not active. Put caption in double quotes if it contains blanks. Use xxx* for captions starting with xxx, and *yyy with captions ending with yyy and *zzz* for captions containing zzz anywhere. You can use multiple captions separated by commas. Use =progname for any window from program with exe file name progname (no path, no .exe).You can put a number n ahead of the caption to limit wait to n seconds.

Wait until window with specified caption is running:

Command: *wait

Parameter: window xxx

waits until any program with caption xxx is running. Put caption in double quotes if it contains blanks. Use xxx* for captions starting with xxx, *yyy with captions ending with yyy, and *zzz* for captions containing zzz anywhere. You can use multiple captions separated by commas. Use =progname for any window from program with exe file name progname (no path, no .exe). You can put a number n ahead of the caption to limit wait to n seconds. . (You can use **caption** instead of **window**). Use **visiblewindow** to ensure the window is visible.

Wait until window with specified caption exits:

Command: *wait

Parameter: nowindow xxx

waits until any program with caption xxx exits. Put caption in double quotes if it contains blanks. Use xxx* for captions starting with xxx, and *yyy with captions ending with yyy and *zzz* for captions containing zzz anywhere. You can use multiple captions separated by commas. Use =progname for any window from program with exe file name progname (no path, no .exe).You can put a number n ahead of the caption to limit wait to n seconds. (You can use **nocaption** instead of **nowindow**). Use **novisiblewindow** to omit invisible windows.

Wait until command with specified exe path is running:

Command: *wait

Parameter: path c:\path\prog.exe

waits until any program executed from c:\path\prog.exe is running. Put path in double quotes if it contains blanks. You can put a number n ahead of the path to limit wait to n seconds.

Wait until command with specified exe path exits:

Command: *wait

Parameter: nopath c:\path\prog.exe

waits until any program executed from c:\path\prog.exe exits. Put path in double quotes if it contains blanks. You can put a number n ahead of the path to limit wait to n seconds.

Wait with a message box and a count down timer:

Command: *wait

Parameter: message n text

displays a message box containing **text** and a countdown timer which starts at n seconds. 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 time setup dialog.

Wait 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.

Wait for alt, ctrl, or shift key

Use ctrl, alt, or shift as command parameter in *wait command to wait until this key is pressed. Use noshift, noalt, noctrl to wait until the key is not pressed. You can optionally follow any of these by a number n of seconds to limit the wait to that time.

If you reconfigure Windows PowerPro, all outstanding waits will be ended.

You can have at most eight outstanding waits.

You can terminate all outstanding waits by running the command:

Command: *wait

Parameter: quit

Manipulating Windows of Running Programs

Purpose

Use the *Window command to ask Windows 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 Windows PowerPro shows, a list of window captions, or all windows on your system.

Configuration

The command has this format:

Command	*Window
Action	action
Paramter	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 Posotion 10 30 100 200 autorun
positions lasts window selected on autorun menu.

Command *Window
Parameter minmemory "*Netscape,*Internet Explorer"
swaps Netscape or Internet Explorer out to disk (NT only).

Specifying the Action for *Window Command

Following are the possible values for the action of the *Window command:

close	closes window
closeforce	forces the window to close; you may lose unsaved information
min	minimizes the window
max	maximizes the window
normal	displays as non-minimized, non-maximized
move	move the window by moving mouse; click any mouse button to stop move
size	size the window by moving mouse; click any mouse button to stop size
hide	makes window invisible
hideshow	makes window invisible if visible, shows 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 (nottop 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 within full screen
rollup	rolls up the window to just caption; shows if it is already rolled-up
maxnormal	maximizes normal window; makes maximized window normal
minrestore	restores minimized window; minimizes otherwise
traymin	minimizes window to tray
automin	minimizes window to tray if window matches autotraymin on Window Control tab; ordinary minimize otherwise
minmemory	setting memory working set (NT, W2K, XP only).
SetPriority	set process prioirity of selected window; precede caption by idle (lowest), below, normal, above, high (highest)
Position x y w h	sets a window position
Trans x	make window transparent (W2K, XP only)
PostMessage m w l	Does PostMessage(h, m, w, l) where h is selected window
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

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 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.

For the Trans command, precede the window target by an integer -255 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, eg 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 minimumum 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.

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 menu.
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. If the *Window menu command is included in a Windows 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 in the work directory of the *Window command .
all	Selects all visible windows, including minimized windows.
window_list	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). Put ~ at the start of the window list to avoid an error message if no matching window is found. Put the window_list in double quotation marks if it contains blanks. Example: "*Notepad,*Internet Explorer, =calc" selects notepad windows, Internet Explorer windows, and Calculator windows. You can also use a window handles or list of window handles separated by commas in the window list.

Window Handles

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.

Following are ways to retrieve window handles:

window("firstwindow",caption_list)	returns handle of the first window found which matches caption list. For example, "firstwindow" window "active"

	returns the handle of the active window.

window("visiblewindow", caption_list)	returns string of window handles of visible windows matching caption list. The window handles are separate by blanks. You can use the *word* operator to extract them, as in
	list = "visiblewindow" window "=exename"
	for (j=1;;j=j+1)
	handle =list word j
	if (handle == "")
	break
	; Process the handle
	endfor

window("anywindow",caption_list)	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* operator to extract them.
LastActiveHandle	Variable which PowerPro sets to the handle of the last window selected by an active button.
LastAutorunHandle	Variable which PowerPro sets to the handle of the last window matched by an autorun command list.

Once you have a window handle, you can use it in any command or context which requires captions list: in a *window command, in a 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 MyHandle.

For example,

*Format Context ~&(hWnd)

would include the following menu items if any window but the one in variable hWnd was active.

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 Windows 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. Use only letters, digits, and spaces in the item name. 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 and type in the command name *Macro into the command entry edit box. Put the name of the command list with the macros in the parameter edit box of the command.

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	.
Hot Key Command	*Macro
Hot Key Parameters	mymacros

When you type **.me**, Windows 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 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: Windows 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 also 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 (eg as a button command).

Favorite Folders and 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 manually set favorite folders and recently used favorite 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 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 menu, 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 precede file folder paths by myname= to have "myname" to appear in the menu to represent the folder path. You can edit either the _any.txt file or the .txt file for a specific program, or both.

To use a menu of only manually entered files, make sure the "Combined Menu" and "Separate Menu" checkboxes are unchecked on the setup tab. Edit the files for manual entry. Omit the sep or colsep at the start.

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 Folder on a Bar

To track folders as you use them for display on a bar, start by checking "Shortcuts" on the setup tab. This causes PowerPro to create a shortcut in c:\program files\powerpro\favfolder_anyshort folder for each folder as you access it. The shortcut command will cause PowerPro to run *Keys {to folder}c:\path which is used to set the folder.

Then you need to create a bar to display these shortcuts.

To do so, create a new command list and make sure you check "Autoshow as bar". Set Properties as follows: check tool tips, set max text to 32, set icons to none, check vertical bar (**not** vertical text), check bar size from sum of buttons, set position to "to right of active window". Set the edit "Use this folder …" at the bottom of properties to c:\program files\powerpro\favfolder_anyshort (or change as appropriate if you installed PowerPro in another folder). Check "Show text" and "Auto-refresh". Then create a single entry in command list for the bar

*Format

Context

filedialog

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

To add manually set folders to the bar, create 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 "Shortcuts" on setup tab or leave the "Use this folder…" edit box on Properties blank.

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

Windows PowerPro Command Line

Windows PowerPro normally uses the configuration file pproconf.pcf found in the same folder as the Windows PowerPro .exe file.

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

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

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

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"

As well as pcf files, Windows PowerPro puts all files which it can change into this 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 Windows PowerPro folder, you must move all **.pcf** files, all **.iconpos** files, all **.timerlog** files, all **.alarmlog** files and the **explorer.windows** file.

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

You can also run built-in (*) commands from a command line. This could be used to execute PowerPro commands from shortcuts or batch command files.

PowerPro must already be running. Type the built-in command, action, and parameters on the command line. For example

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

shows menu MyMenu from the running PowerPro program.

Information for Stiletto Users

PowerPro is not backwards compatible with Stiletto. You will need to manually convert your configuration.

Purchasing PowerPro

The 32 bit version of Windows PowerPro is freeware.

Frequently Asked Questions (with Answers)

Where is my configuration stored? How do I back it up? How do I keep my configuration when upgrading?

The configuration is stored in file Pproconf.pcf. Take a backup copy of this file to save your configuration. Installation zips of Windows PowerPro do **not** include a Pproconf.pcf file so they do not overwrite any existing configuration when installed.

What are all the files in the Windows PowerPro folder? Which can I delete?

See filelist.txt in the folder for an explanation. In addition, Windows PowerPro creates

Where is my registration code stored? Do I have to re-enter it for each upgrade?

The registration code is stored in the registry. Windows PowerPro automatically reads it from there. There is no need to re-enter when upgrading.

How can I start many Explorer windows at the same time? How can I set the folder that Explorer starts with?

To start many windows from Explorer (or any other program), you much uncheck "Switch to If Active" at the bottom of the command entry controls for each button or menu item which is to start the command. To learn how to use Explorer to start at any folder, see the file tips.txt that Microsoft includes in your Windows directory. Put the command parameters described there into the Windows PowerPro Parameters edit box.

For win95 and NT4, what is the best way to show a menu by right-clicking the desktop?

If you use the menu setup tab to set a desktop menu, Windows PowerPro will attempt to show both the Windows PowerPro menu you set and the Windows desktop or desktop icon context menu. This may not always work well; eg in NT 4, one of the menus may not close properly.

Instead of using the menu setup tab, create a right-desk hot key which executes a *Show Menu for your desktop menu. Include the following command in your menu:

Item Name:	Context
Command	*Mouse
Parameter	RightClick

If you click your mouse anywhere on the desktop, only the Windows PowerPro menu will be shown. To access the Windows context menu for the item under the mouse, select the Context command.

You may also want to experiment with right-hold hot keys, chord left+right hot keys, and middle mouse hot keys.

How do I use middle mouse button to send left double click? What else can I do with the middle mouse button?

To send middle double click, attachew the command *Mouse leftdouble to the middle anywhere hotkey.

The middle mouse button can provide many other functions with Windows PowerPro**:**

You can attach hot keys to it: for example, a mouse-all hot key and a mouse-hold hot key. Use these hot keys for direct commands, like sending a left double click with *Mouse, or for menus,

such as menu of send key commands to send common shortcut keys or simulate picking menu entries.

In addition to the hot key, you can also use the middle mouse for either scrolling or for moving a window by setting the option on the special config tab.

How can I activate programs which are not files, such as printers or control panel applets?

Use Explorer to create shortcuts to these special programs and then run the shortcuts from Windows PowerPro. You can create a folder of shortcuts to all your printers or other special programs, and display them all as a menu using *Menu Folder.

How do I create a bar in the caption so it looks like the icons of the bar are part of the caption?

On the Properties dialgo for the commandlist of the bar, set the colors to the caption colors and uncheck the Border and Sizing checkboxes.

Power and Flexibility of PowerPro

Windows PowerPro gives you the power to control your system and how you access programs because it allows you to choose the combination of how to activate and what to activate.

The following table lists all the techniques for activation and all the things you can activate. You can combine any entry from the first column with any entry from the second column.

How to Activate

Button Bars
- left, middle, right click
- keyboard access to bar

Menu
- pick item and dismiss menu
- show all files in folder and run one

Tray Icon
- left, middle, right click

Hot Key
- alt, ctrl, win plus any key
- tap ctrl, alt, shift, win, caps lock
- specified key (like ;) then any key
- can depend on active program

Keyboard Macros
- any string of characters

Mouse Actions (any mouse button)
- click window
- click desktop
- click caption
- click left or right of caption
- click system, min, or close box
- press and hold mouse button
- short horizontal or vertical drag
- horizontal or vertical stroke
- move mouse to screen corner
- bump edge of screen
- chord two mouse buttons
- can depend on active program

When A Specified Window First Appears
- based on caption or exe name

At a Scheduled Time
- any time/date with repeat interval
- after system idle for specified time
- at PowerPro start up

Based on a Timer
- timer start or stop
- after a countdown

What to Activate

Run any program, shortcut, or document
- specify parameters
- specify keys to send at start
- specify window configuration at start
- browse files and select file to run

Virtual desktop
- create new desktop
- switch to existing desktop
- drag/drop windows between desktops

Control any Window on your Desktop
- select by caption, or under mouse, or all windows, or from menu of active windows
- select main window or MDI window
- activate, close
- topmost, not topmost
- hide, show, minimize, maximize, normal
- minimize to the tray
- rollup so only caption is visible
- send to back (underneath all windows)

Show a Menu
- pre-built with optional submenus
- portions shown can depend on active program
- show Start Menu at any location

Run a Script of Many Commands
- script can contain any command in this column
- program logic with flags, variables, if, jump, wait

Work with Tray Icons of Other Programs
- simulate left, middle, or right click
- hide icon

Show Contents of Folder as a Menu
- select a file and run it or show its properties
- access special folders like desktop, start menu

Send Keystrokes to a Running Program

- at a repeating interval
- insert text
- control the program by sending Alt+ or Ctrl+ keys

Send Mouse Actions to Running Program

Control Look of Your Desktop

- change wallpaper/saver
- change any sound
- all changes can be random, sequential, or to specific file
- hide/show desktop icons or task bar
- save and restore desktop icon positions

Shut Down or Restart Windows

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 "Change configuration" menu item, and then select "demo" from the resulting submenu.

The demo bar will appear in the top left of your screen. Click on one of the following topics for more information:

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.

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.

Demonstration of Menus and Context Menus

Left click the menu button to see the command list MenuSample displayed as a menu. This menu illustrates submenus and context menus. You will see on the command list MenuSample (ctrl-right click bar) 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. The *Keys and *Message commands between these two commands will only be displayed if Notepad is the active window. You can start notepad with the NotePad button on the bar. Press the menu button when notepad is running and active and note that these entries appear. Try again with notepad running but not active (eg click on desktop) and you will see a different menu.

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 (ie the command list) is separate from how it a menu is shown (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; click here for more information.

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, on top, or not on top the windows. Access the configuration dialog with ctrl-right click on the bar, and note command list WindowMenu. See how the *Window commands are placed within *Format StartSubmenu/*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.

Demonstration of Subbars and Manually Shown Bars

Ctrl-right click the bar and view the command list Bar. It shows three subbars: edit, util, and none. You will see these names on the *Format StartSubbar Commands. You can switch among bars by right clicking the menu button and selecting 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!

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, the Properties for the ManualBar command list 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.

Demonstration of *Window Commands

*Window commands let you manipulate active windows. Ctrl+right click the bar 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 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 ctrl+alt+k 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 can also display menus of active tasks; selecting one performs the specified action. Middle click (shift+left) the menu button to see window WindowMenu 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 menu command directly from a button or a hot key. Right clicking 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).

Demonstration of Sending Keys with *Keys

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

The snippets menu 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 through the use of 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.

Demonstration of Hot Keys and Mouse Actions

The sample bar illustrates many different ways to use hot keys/mouse actions. Ctrl-right click the bar 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 and vertically for about 10 pixels for the name sort and 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, *Menu Show, and menus of *Keys.

Demonstration of *Menu Folder

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'll 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 market 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, ensure the bar entry is selected on the command list 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 show 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.

Demonstration of Keyboard Macros

Keyboard macros let you use abbreviations for text you commonly type. You create a command list where the 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 command is faster for longer text but it will overwrite the clipboard.

Ctrl-right click the bar 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.

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 use the Macros command list, you must define a macro signal character with hot keys. 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 macro. Then try =xx and =min.

Demonstration of Running Commands when a Window First Opens

You can configure PowerPro to check each new window that is opened on your system and run a PowerPro command for specified windows. 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 and view command list NewWindow.

To activate the feature, you must first press the command list|setup button and select the command list "NewWindow" in the drop down "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 list NewWindow.

The command item names are used to matched 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 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 is opens; try selecting about notepad from the notepad help to see the effect. You can press the default button of dialogs by sending the {enter} key stroke and you can press other buttons by sending alt-x (%x), where x is the underlined character on a button name. You can use &() expressions in the item name; the expression is re-evaluated each time a new window is opened and checked.

When you are finished with the demo, be sure to go back to command list|setup and set the drop down box back to (none) so that this NewWindow sample does not interfere with other samples.

Virtual Desktop Sample

To run the virtual desktop sample, 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 to 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 desktop|setup configuration. Note how desktop Manual is started with the CreateOrSwitchTo command; this command runs the 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 desktop|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 and view the configuration of the bar, 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 be part of all desktops: try all of the desktop buttons to see this. You could have also prespecifed that calculator be locked by entering the caption Calculator in the "Windows to be shown on all desktops" edit box on the Desktop|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.

Notes

Purpose

Use notes to capture any type of text: reminders, scripts, text for copying into other documents, and so on. 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 from notes to other windows from windows to notes.

Configuration

The *Note command is used to work with notes. Create new notes or open existing notes with *Note Open. Hide or show open notes with *Note ShowHideOpen, open all notes in a category with *Note OpenCategory, open one note in a category with OpenOneFromMenu, close and save all notes in a category with *Note CloseCategory. You can see a menu of all open notes and show and bring the selected one to the top with *Note OpenMenu. You can show notes with date categories today or before with *Note ShowToday. You can delete open notes from a given category with *Note DeleteOpenCategory.

When you create a new note by executing *Note Open, you can optionally set its colors, text, source file, and category. To access these features, use the "find" button to the right of the edit box when configuring the *Note command.

Right click on a note text or use the Apps key (beside right ctrl) to see a configuration menu which lets you set the note category, set note color, close the note, run the entire note as a PowerPro script, and perform other functions.

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).

Resize a note by dragging its border and move a note by dragging the caption.

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.

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.

Select all text in a note by clicking the A. Left click and drag from the A to drag-and-drop all note text.

Rollup a note to its caption by clicking R or using Rollup selection from right-click menu. 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 (-).

You can control the default size and position, color, and font of newly opened notes with the Configuration tabbed dialog, setup tab, advanced button, note tab. 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 desktops and whether shown notes get the keyboard focus.

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 Cat is the note category and "first line of text*" is the start of the first line of the note's text.

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 Configuration|Setup|Advanced|Note) or by running *Note ShowToday (eg in a StartUp alarm).

Notes can hold up to 64K of text.

Summary of Usage

The following table summarises the user interface to notes and how to use it:

Desired Action	How to Perform Action
Create new note	Execute *Note Open from bar, menu, hot key, tray icon, etc
Create notes in different categories each with own	Configure a *Note Open for each category, using "find" button to assign color and category.

color	
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 button at upper left of note. Drag to move all text. Hold down ctrl to copy.
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 note in its category folder and delete.
Set category of note	Specify initial category in *Note Show, 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 ".

Skins

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 specify that a bar uses a background bmp 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 Kaos1SkinSample, zlkSkinSample, NewbieSkinSample, and KaosSkinSample 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 at as basis for creating PowerPro skins.

Click here 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 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** or a bmp file name 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. You can find more information on this type of subbar usage here.

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.

Creating Skins

General Structure of Skins

A skin consists of a skin .txt file and associated bmp, cursor (.cur or .ani), 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 *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.bmp"
```

sets the bar with keyword height, value 200, keyword width value 150, and shape to background for panache.bmp.

Blank lines and any lines in the .txt file starting with a semi colon (;) are ignored.

The case of letters in keywords is ignored: eg, *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: eg, "244 0 0" is bright red. You can use the Paint program, 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: bmp files, wav files, 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 (eg .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

this 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 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.

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 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 a *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 using 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 is used to create a series of section buttons; pressing any section button displays a set of command launch buttons. Subbars are used in the pcf file for the button in each section. 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 .bmp 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.

PowerPro shape bmps are compatible with the bmps created for the LaunchKaos program; you can find many such bmps at www.skinz.org.

Bars using "shape" are never resizable. Use the "background" keyword instead of shape to allow resizing (depending of course of the pcf settings).

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 bmp which determines the shape, size, and look of the skin.	none
background	Followed by file name for .bmp 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 bmp 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 is shown after autohide.	none
soundhide	sound file to play when bar is hidden.	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
menuwidth	Fixed menu column width in pixels.	none
menuheight	Fixed menu item height in pixels.	none
menusepcolor top	Color of top line of separator.	system color

menusepcolor bottom	Color of bottom line of separator.	system color
menuindent	Fixed menu item indent in pixels.	none
maxmenuwidth	Maximum menu column width in pixels.	none

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**

The *font line can be used to create up to four 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, 3 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 between 0 and 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 between 0 and 3600. Specifies the angle, in tenths of degrees, between each character's base line and the x-axis of the device	0
charset	Needed for some non-English fonts; followed by a number between 3 and 255 (see below).	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**

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. For 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**

*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 Value
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 first item in the command list which has an item name beginning with the same characters as the string	next
left	Position of left of button, relative to top left of bar, in pixels. Use 0 for top left of bar.	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 or the button or when the button is pressed (not followed by a value). You would normally specify this	N/A

	keyword if you specified the pressbmp or hoverbmp keywords.	
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)	none
iconover	Icon is only shown if mouse over button (no following value)	none
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 3).	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 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 most than once to specify both horizontal and vertical justification, eg 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.	
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 all cases	none
facebmp	Background bmp file when mouse is not on button.	none

hoverbmp	Background bmp file when mouse is over button	none
pressbmp	Background bmp file when button is pressed	none
allbmp	Sets face, hover, press to same value	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

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

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 start with the text characters. If ia 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 bmp for background, hover, or press, the bmp 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 usag

Sharing PowerPro Configurations

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 say 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, by using *Exec ChangeConfiguration, or 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 tool. 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 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.

If you want to use command lists or hot keys from a shared configuration in your working configuration, use the export as text feature to export the command lists or hot keys of interest. Then import these into your configuration. You may also need to set features from the shared configuration command list|setup dialog.

Contact for Questions or Support

Find out about the latest Windows PowerPro version at

www.WindowsPowerPro.com

Plugins

General Information

PowerPro allows programmers to add new features by writing plug-ins. A plug-in is a .dll file that you must place in the same folder as your PowerPro .exe file or subfolder plugins (usually c:\program files\powerpro\plugins). The plugin will offer one or more services.

To use a plugin, you use an expression. Use the syntax
dllname.service(args)
to run a plugin. The case of the dllname and the service are ignored.

You can use plugins standalone in a statement or in expressions. Standalone plugin calls are a form of expression and you can use variables and other expressions in the arguments.

If the plugin service as no arguments, you can call the service by writing either dllname.service() or dllname.service.

Examples

file.close(handle)
call file plugin, service close, with argument handle. Note that you can use a plugin call alone as a script command or in the command entry controls.

res = float.add(a,b,c)
calls float plugin, service add, arguments a, b, and c.

file.CloseAll
calls the file plugin, service closeall (case of service name ignored) with no arguments.

days = Date.Diff(date.today(), test_date) + 5
calls Date plugin with service diff. The first argument in turn calls date plgin again with service today.

PowerPro comes with three plugins -- file, date, and float. These are found in the file plugins.zip which you must unzip into the powerpro folder. See the corresponding .txt file for documentation.

If you wish, you can unload plugins from memory when you are finished with them with the unload service, eg 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.

How to Program Plugins

The remainder of this help section is intended only for programmers of plug-ins.

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 showmenu(NULL,NULL,
        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 263 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 9 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.

You can return a result from the plugin from setting *szargs to the result if it is less than 263 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 s1, LPSTR s2,LPSTR s3, LPSTR s4, LPSTR s5, LPSTR s6);
        LPSTR (*AllocTemp)(UINT leng);
        void (*ReturnString)(LPSTR sz, LPSTR* szargs);
        LPSTR (*GetVarAddr)(LPSTR var);
        LPSTR (*SetVar)(LPSTR var, LPSTR val);

}
PPROSERVICES;
```

Use the call (ppsv->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.

Use (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.

Use (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.

Use (ppsv->IsRolled)(hw) to check to see if the specifyiedwindow is rolled-up by PowerPro; function returns 1 if it is and 0 otherwise.

Use (ppsv->IsTrayMinned)(hw) to check to see if the specified window is trayminned by PowerPro; function returns 1 if it is and 0 otherwise.

Use (ppsv->GetExeFullPath)(hw, sz) to get full path to exe of window hw)

Use (ppsv->RollUp)(hw) to rollup window hw if it is not already rolled up and to show it otherwise.

Use (ppsv->TrayMin)(hw) to tray minimize window hw if it is not tray minimized and to show it otherwise.

Use (ppsv->SendKeys)(sz) to send keys using sz; this is equivalent to using *keys command on sz. The string sz cannot contain more than 260 characters.

Use (ppsv->EvalExpr)(szExpr, szResult) to evaluate the expression stored in szExpr and return the result in szResult. The result is always truncated to 263 characters.

Use (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. You do not need to specify all six strings; e.g (ppsv->Debug)(s1,s2)) is valid.

Use (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.

Use (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.

Use (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) to avoid memory leaks.

Use (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.

Sending Commands to PowerPro

You can send commands back to PowerPro using two methods: 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 using 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 a *menu Show command on the menu name stored in variable MenuName.

```
  #define VAR_SIZE 264
char szCommand[VAR_SIZE+24];
char szName[VAR_SIZE+1];
COPYDATASTRUCT cd;

strcpy(szCommand, "*Menu Show  ");
(*GetVar)("MenuName", szName);
strcat(szCommand, szName);
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);
```

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".

AU PLUGIN (Version 2002 12 22)

OVERVIEW

The au plugin provides access to the services provided by the AutoIt dll as documented in AutoItDll.chm. This dll must be in the same folder as au.dll.

Many of these services are also available from the win keyword or from the PowerPro *keys commands. However, the au plugin uses a different approach for implementing these features which may work when a PowerPro feature is not working. For example, try au.send as an alternative way of sending keystrokes.

The documentation for the features of au.dll is found in AutoItDll.chm. When calling a function, use au. instead of AUTOIT_. For example, send a left click with au.LeftClick or test if a window is active with au.IfWinActive(caption, text).

Use any PowerPro variable or expression for the service arguments. The au dll will use integers or strings as required by AutoItDll.dll.

DATE PLUGIN (Version 2002 10 29)

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.

LIST OF SERVICES

```
Date.Today      todays date, same as PowerPro keyword date
Date.AddDays    adds a given number of days to a date
Date.Sub        subtracts two days, given the number of days 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 digit dates, and n1 is a number of days In all cases, n1 may be positive or negative.

TODAY
Date.Today
Returns todays date; same as internal keyword date.

ADDDAYS
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)

SUB
d3 = Date.Sub(d1, d2)
Gives the number of days between d1 and d2.
d3 = Date.Sub(20021102, 20021031) sets d3 to 2.

WEEKDAY
Gets the day of the week for a the date; Sunday is day 0.
d1 = Date.WeekDay(20021029) sets d1 to 2 (Tuesday)

YEARDAY
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

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

GET
d1 = Date.Get
Displays a calendar and returns selected date. Returns 0 if Cancel pressed.

EVENT PLUGIN (Version 2003 01 01)

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.

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.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.
```

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.

In commands and expressions, you must precede the expression follows character (eg &) and the double quotation by a single quote '. For example, the following creates an event which shows the event count if notepad is active.

event.create(1,100,"debug '&(event.countthis)", "visiblewindow('"*notepad*'")"

Note how the & and the "s in the expression are preceded by ' so that this characters are not processed until the event occurs.

You can use multiple commands in the command by separating each command with 'r. For example,
event.create(3,0,"debug '&(var1)'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.

CREATE

e=event.create(s,c,cmd,expr)
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.

For example,

e=event.create(3,0,"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 20 seconds elapse. Note how '+ is used to extend the command over two lines (for scripts in files).

CREATEMS
e=event.createms(m,c,cmd,expr)
Same as create, except that the interval is specified in milliseconds. The minimum is 20 milliseconds.

DESTROY
e=event.destroy(e)
Removes event e and returns zero which can be assigned to a variable to set it to an invalid event handle.

DESTROYTHIS
event.destroythis
Must be used in the command assigned to an event when it is created. If executed, removes the command.

DESTROYALL
event.destroyall
Removes all active events.

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.

EXISTS
event.exists(e)
Returns 1 if e is a valid event; 0 otherwise.

COUNT
event.count(e)
Returns the number of intervals remaining for an event scheduled with a count.

COUNTTHIS
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,"debug &(event.countthis)")
creates an event which displays a countdown 9,...0 in the debug window.

FILE PLUGIN (Version 2003 01 19)

OVERVIEW

The file plugin lets you read and write text to files. 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.

The plugin can only handle text files with lines terminated by a newline (NL) or Carriage Return (CR)-NL pair. Neither CR nor NLs are returned by the plugin.

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.

LIST OF SERVICES

For this list, the following standard variable names are used:

fh is used to identify the File Handle: a integer assigned by the plugin service open when a file is opened.

str is used to identify a string or a variable holding a string.

The following is a list of services

```
open        opens a file and returns a File Handle used subsequently to
access the file.
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
eof         returns 1 if last read encountered eof; 0 otherwise
writeline   writes a string and following CR-NL
writestring writes the string but does not output a CR-NL
setmaxline  set maximum line length for subsequent calls to
file.readline
```

Following is a more detailed explanation of each service.

OPEN
fh = File.Open(str1, str2)

The Open service opens the file with path given in str1. 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

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.

CLOSE
File.Close(fh)
Closes the file given by fh.

CLOSEALL
File.CloseAll
Closes all open files.

RESTART
File.Restart(fh)
Next operation will occur at start of file.

READLINE
str = File.Readline(fh)
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.

SETMAXLINE
File.SetMaxLine(n)
Sets maximum line length for subsequent file.readline calls. If n <264, 264 is used. If n>64K, 64K is used.

EOF
File.Eof(fh)
Returns 1 if last read encountered an eof; 0 otherwise. The standard way to read a file line-by-line is:

```
fh = File.Open(strPath, "r")
if (fh > 0) Do
  for (lineNum=1;1;LineNum=LineNum+1)
    str = File.ReadLine(fh)
    if (File.Eof(fh))
      break
    ; process the file line in variable str
  endfor
else
  MessageBox ("ok", "Error opening file "++strPath)
endif
```

WRITELINE
File.WriteLine(fh, str)
Writes the line given in str to the file, adding a CR-NL to make a complete line. YOu can also use File.WriteLine(fh, "")
to write just a CR-NL and terminate any lines partially written with File.WriteString.

WRITESTRING
File.WriteString(fh, str)
Writes the string str to the file with no terminating CR-NL. Subsequents WriteLine or WriteString will add text to same line.

EXAMPLE
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 = file.readLine(fp)
      if (file.eof(fp))
        break
      file.writeString(fpout, "***")
      file.writeLine(fpout, line)
    endfor
    file.close(fpout)

  else
    Debug cannot open out &(fpout)
  endif
  file.close(fp)

else
  debug cannot open in &(fp)
endif
```

FLOAT PLUGIN (Version 2002 10 28)

OVERVIEW

The float plugin lets you do arithmetic with floating point numbers, like 14.567 or 2.3e-6.

Floating point numbers are not directly supported by PowerPro, so you must put any numbers with a decimal points in quotes when you use the float plugin. Quotes are optional for numbers without decimal places. For example, float.add("1.718", 2) yields "3.718". You can use both decimal points and the exponent notation with numbers, for example

"0.00021" is the same number as "2.1e-3"

Since floating point 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

```
Float.Add        Adds two to nine nine numbers and returns result in
string.
Float.Sub        Subtracts two numbers.
Float.Mul        Multiplies two to nine numbers.
Float.Divide     Divides two numbers.
Float.Compare    Compares two numbers, returning -1, 0, or 1.
Float.Ceiling    Returns smallest whole number which is greater than
input
Float.Truncate   Returns the smallest whole number which is less than
input
Float.Display    Returns float number rounded and without "e" notation;
specify total digits
Float.DisplayDec Returns float number rounded and without "e" notation;
specify digits after decimal point
```

DESCRIPTION OF SERVICES

In the following descriptions, r1, r2, r3, r9 are floating point numbers.

ADD
r3 = Float.Add(r1, r2, ..., r9)
Adds up to nine numbers and returns result.
For example
r1 = Float.Add("1.1", "2.2", "3.3") sets r1 to "6.6".
r2 = Float.Add(8, r1) sets r2 to "14.6".

SUB
r3 = Float.Sub(r1,r2)
sets r3 = r1-r2.

MUL
r3 = Float.Mul(r1, r2, ..., r9)
Multiplies up to nine numbers.
For example
r3 = Float.Mul("0.5", "0.25")

sets r3 to "0.125"

DIVIDE
r3 = Float.Divide(r1,r2)
Sets r3 = r1/r2.

COMPARE
n1 = Float.Compare(r1,r2)
Sets n1 = -1 if r1<r2, 0 if r1==r2, 1 otherwise.

CEILING
n1 = Float.Ceiling(r1)
Sets n1 to next whole number larger than r1.

TRUNCATE
n1 = Float.Truncate(r1)
Sets n1 to next whole number smaller than r1.
To round a number, you can use Float.Truncate(Float.Add(r1,"0.5"))

DISPLAY
r2 = Float.Display(r1,ndig)
Returns r2 with n significant digits. The e notation will not be used. ndig must <=25.

DISPLAYDEC

r2 = Float.DisplayDec(r1,ndig)
Returns r2 with n digits after the decimal point. The e notation will not be used. ndig must <=25.

Sample PLUGIN (Version 2002 12 22)

OVERVIEW

The sample plugin provides a sample of how to write plugin services.

DESCRIPTION OF SERVICES

```
showmenu(str)          shows a menu with name in str
echo(a1,a2)            echoes up to 8 arguments
okcancel(msg,title)    shows an ok/cancel messagebox and returns the
result.
```

VEC PLUGIN (Version 2002 12 28)

OVERVIEW

The vec plugin lets you work with single-dimension arrays, ie vectors.

You create an array with the vec.create function which returns a handle to use in subsequent calls to vec. Handles are always greater than 0. You can then set elements with vec.set, retrieve them with vec.get.

Vectors have a size, called a capacity. This capacity is the number of elements the vector can contain. You specify the starting capacity of the vector when it is created. You can subsequently grow the vector, either automatically when you attempt to set element one larger than the capacity, or manually, with the vec.grow call.

Vectors are indexed starting at 0 to their capacity-1. If you retrieve elements which have not been assigned a value, vec returns "" for the value.

You can insert new elements in the array with vec.insert, which moves all existing elements up to to create an empty space for the inserted element. You can delete an element and move existing elements down.

You can retrieve the current capacity of an array with vec.capacity. You can retrieve the current used length with vec.length. The length is the index+1 of the largest set element. There can be unassigned elements with index less then the length.

When you overwrite an vector element with a new string value, the vec plugin will attempt to reuse the existing memory of the overwritten element if there is enough room for the new string. This helps reduce memory fragmentation. You can specify a minimum number of bytes for each element in the vec.create call to help reduce fragementation this way if you plan to make many changes to array elements.

The vec plugin can support at most 25 simultaneously active vectors.

LIST OF SERVICES

```
vec.create       Creates a new vector and returns its handle.
vec.destroy      De-allocates all the vectors memory and frees its
handle.
vec.exists       Returns 1 if vector exists; 0 otherwise.
vec.set          Set an element, overwriting anything value at specified
index.
vec.get          Retrieves string at specified index.
vec.insert       Inserts new element in vector by moving elements up
one.
vec.delete       Removes element from vector and moves elements done
one.
vec.grow         Grows capacity of vector by specified amount.
vec.capacity     Returns capacity of vector.
vec.length       Returns length of vector
vec.sort         Sorts vector ascending.
```

DESCRIPTION OF SERVICES

In the following descriptions, v is s vector, i an integer>=0, s a string.

CREATE
v= vec.create(capacity, growth, minsize)
Creates a vector with specified capacity. 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 you specify minsize, all elements will be allocated at least this much memory, which 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.

The vec plugin can support at most 25 simultaneously active vectors.

DESTROY
v=vec.destroy(v)
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.

EXISTS
vec.exists(v)
Returns 1 if v is a valid vector; 0 otherwise.

SET
vec.set(v,i,s)
Assigns the string s to element i of vec v. Indexes start at zero. If i>=vec.length, then the vector length is assigned i+1. If i==vec.capacity, and growth is bigger than 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.

GET
s=vec.get(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.

INSERT
vec.insert(v,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.

DELETE
vec.delete(v,i)
Moves all elements starting at i down one, deleting the element at i. Vec.length decreases by 1.

GROW
Vec.grow(v,growth)
Adds growth elements to the capacity of v.

CAPACITY

n=vec.capacity(v)
Sets n to current capacity of vector v.

LENGTH
len=vec.length(v). Sets len to current length of v. The length is one greater than the index of the highest set element.

SORT
vec.sort(v)
Sorts v in ascending order.

EXAMPLES

The following example reads in file c:\sortme.txt, sorts it, and writes it to c:\sorted.txt.

```
han = file.open("c:\sortme.txt","r")
v = vec.create(200,100)
for (i=0;1;i=i+1)
  line = file.readline(han)
  if (file.eof(han))
    break
  vec.set(v,i,line)
endfor
file.close(han)

vec.sort(v)

han=file.open("c:\sorted.txt", "w")
for (i=0; i<vec.length(v); i=i+1)
  file.writeline(han,vec.get(v,i))
endfor
file.close(han)
vec.destroy(v)
```

WIN PLUGIN (Version 2002 12 22)

OVERVIEW

The win plugin retrieves information about a window or its child windows. It can also manipulate windows.

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.

```
left(cl)       returns left window coordinate of first visible window
matching cl
right(cl)      returns right window coordinate of first visible window
matching cl
top(cl)        returns top window coordinate of first visible window
matching cl
bottom(cl)     returns bottom window coordinate of first visible window
matching cl

width(cl)      returns width first visible window matching cl
height(cl)     returns height of first visible window matching cl

handle(cl)     returns window handle of first visible window matching cl
handleatpoint(x,y)     returns window handle ofwindow 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.

caption(cl)    returns window caption of first visible window matching
cl
class(cl)      returns window class of first visible window matching cl
exepath(cl)    returns full path to exe of first visible window matching
cl

maxxed(cl)     returns 1 if first visible window matching cl is
maximized, 0 otherwise
minned(cl)     returns 1 if first visible window matching cl is
minimized, 0 otherwise
topmost(cl)    returns 1 if first visible window matching cl is topmost,
0 otherwise
rolled(cl)     returns 1 if first visible window matching cl is rolled-
up, 0 otherwise
trayminned(cl) returns 1 if first visible window matching cl is tray-
minned, 0 otherwise

resizable(cl)  returns 1 if first visible window matching cl is
resizable, 0 otherwise
maxable(cl)    returns 1 if first visible window matching cl has
maximize box, 0 otherwise
```

```
minable(cl)    returns 1 if first visible window matching cl has
minimize box, 0 otherwise
toolwindow(cl) returns 1 if first visible window matching cl is
toolwindow, 0 otherwise

close(cl)      closes first visible window matching cl
closeforce(cl) closes first visible window matching cl, unsaved
information is lost
rollup(cl)     rolls up visible window matching cl; unrolls if already
is rolled up
traymin(cl)    tray minimizes first visible window matching cl; un tray
mins if already is
ontop(cl)      makes first visible window matching cl topmost; removes
topmost if it already is

show(cl)       shows first window matching cl
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

move(cl,x,y)   moves first visible window matching cl to position x, y;
size(cl,x,y)   sizes first visible window mathcing cl to size x,y

sendkeys(sz)   sends keys in string sz to active window

debug(sz1, sz2) up to 6 arguments can be specified; they are joined and
shown in a debug window.

sendmessage(han, msg, wp, lp)  sends message msg to window with handle
han
postmessage(han, msg, wp, lp)  posts message msg to window with handle
han



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.
```

RegEx Plugin (12/26/02 Version)
Coded by: Julien Pierrehumbert

This package includes a GNU PCRE-compatible regex library (pcre.dll) and a PowerPro plugin (regex.dll) which works as an interface between PP and that library.
The source is of course included. Simply change the first #include to compile it with another POSIX-like regex library (like GNU's rx).

I hope that distributing GNU stuff this way is not a war crime. Anyway, you'll find all the licenses, documentation, and stuff alongside the lastest version of the library at:
http://gnuwin32.sourceforge.net/

Usage:

To execute the plugin, put it in PowerPro's folder alongside pcre.dll and use:
regex-match(string,regular_expression,replacement_string)
or
regex.replace(string,regular_expression,replacement_string)
or
regex.matchg(string,regular_expression,replacement_string)
or
regex.replaceg(string,regular_expression,replacement_string)

For case insensitive treatment, put (?i) in front of the regular_expression.

In addition to the return code, the plugin will create a string, which will be placed in the PP variable "rx_output".

Return Codes:

0	Full Match
1	Partial Match
2	No Match
3	Error: Output Too Long
8	Error: Invalid Pattern
9	Error: Invalid Input

Known issues:

Using lookbacks (a PCRE-specific feature) with the global services might lead to unexpected results.

Contacting the author:

If you don't have my email address, use the list.

Legalese:

Do whatever you want with the program and the source. I decline any responsibility.

Following is documentation of the previous version of the regex plugin (which used a different syntax)

REGEX plug-in syntax (PowerPro 3.4)
Table of Contents

Quick start
Description
Installation
Usage
Input
Services
Output
Return codes - simple examples
Detailed examples: MATCH/MATCHG
Detailed examples: REPLACE/REPLACEG
REGEX plug-in test script
Getting started with Regular Expressions
Final words

- **Quick start:**

If you're familiar with PowerPro and its plug-ins, here is a quick start with everything you need to know. But you're still advised to read the entire document and get acquainted with important details. If you're not familiar at all with PowerPro and its plug-ins and/or want the full explanation, then you should definitely skip this first table and read all the rest:

REGEX plug-in syntax (PowerPro 3.4):

```
*Exec Plugin regex service
or:
some_variable = regex ("regex", "service")

Examples:
*Exec Plugin regex match
x0 = plugin ("regex", "replace")
```

REGEX plug-in syntax (PowerPro 3.5):
This syntax has NOT been implemented yet by the plug-in.

```
some_variable = regex ("regex", "service")
or:
dll.service("arguments")

Examples:
x0 = plugin ("regex", "matchg")
```

regex.replaceg("rx_string", "rx_pattern", "rx_replace")

Services:
match = match first occurrence
matchg = match all occurrences
replace = replace first occurrence
replaceg = replace all occurrences

Input variables:
rx_string = the string to be parsed
rx_pattern = the pattern to be applied on the string
rx_replace = the replacement string

Output variables:
rx_return = numeric return code (click here)
rx_output = transformed (or not) string

- **Description:**

The REGEX plug-in makes it possible to parse short strings with PowerPro, as a standalone operation (to test a regular expression, for example) or in the context of a PowerPro script. PowerPro already provides native functions to select, remove or replace characters in a string, but these native functions are somewhat limited because they rely on predictive characteristics, not available in every situation. Only regular expressions, provided with this handy plug-in, can truly find, match and replace any possible pattern. Another excellent thing about this plug-in is that it supports PCRE (Perl-Compatible Regular Expressions), thus allowing even more flexible and accurate patterns.

This special functionality is added to PowerPro with two DLLs: **pcre.dll**, a GNU library written by **Philip Hazel**, and **regex.dll**, written by **Julien Pierrehumbert**. The latter should be called with PowerPro every time a single match and/or replace operation is desired, using the correct syntax provided below. PowerPro can work with probably any custom-made DLL, making its full range of functionalities virtually unlimited. Note that it is possible to change the first #include in the source code of **pcre.dll** to compile it with another POSIX-like regex library, like GNU **rx**. Further information regarding licenses, documentation and sources of GNU libraries can be found at: http://gnuwin32.sourceforge.net/.

- **Installation:**

Just copy **pcre.dll** and **regex.dll** to PowerPro's base directory, as explained in the chapter about plug-ins of PowerPro's documentation.

- **Usage:**

WARNING!

All syntax, usage and behavior described here applies to PowerPro 3.4. A lot of it is likely to change in version 3.5.

Any given PowerPro plug-in can offer several different and even unrelated functionalities, called "**services**". The REGEX plug-in has four services: **match**, **replace**, **matchg** and **replaceg**. As explained in PowerPro's documentation, any plug-in is called this way:

Exec Plugin** ***plug-in service

So there are four possible ways to call the REGEX plug-in, one for each service:

```
*Exec Plugin regex match

*Exec Plugin regex replace

*Exec Plugin regex matchg

*Exec Plugin regex replaceg
```

Actually, after PowerPro version 3.4.0, the syntax above became deprecated. You may therefore feel tempted to use the new syntax:

```
x0 = plugin ("regex", "match")

x0 = plugin ("regex", "replace")

x0 = plugin ("regex", "matchg")

x0 = plugin ("regex", "replaceg")
```

The new syntax introduces **plugin** as a function that returns a value and assigns it to the special **x0 variable**. If you're familiar with PowerPro plug-ins and the instructions provided in the manual, you would think the return value will be assigned to **x0** anyway so we don't need to assign the function to **x0**. Such assumption makes sense, but:

- PowerPro's syntax does not allow the use of a function without a **variable** assignment. If you try, for example,

```
plugin ("regex, "match")
```

PowerPro will think that "**plugin**" is a command, which does not exist, and will generate an error message. We have to assign the **plugin** function to a **variable**. Assigning it to **x0** may seem redundant, but it works;

- if you're familiar with PowerPro plug-ins and the instructions provided in the manual, however, it may not work quite as expected. The manual says that the function's output will be assigned to the **x0 variable** automatically, and that is not entirely true because the plug-in must comply with such standard for this behavior to occur. The REGEX plug-in does not comply. It sends output to the **rx_return** and **rx_output** variables instead, so the **x0 variable** will be empty and useless even after the assignment. So, although it is not necessary, you may want to avoid using the **x0 variable** and create a new **variable** like, say, "**void_var**":

```
void_var = plugin ("regex", "match")

void_var = plugin ("regex", "replace")

void_var = plugin ("regex", "matchg")

void_var = plugin ("regex", "replaceg")
```

Feel free to use any other name instead of **void_var** to remind you that this is an empty **variable**, or just keep using **x0**. So long as you assign the **plugin** function to a variable.

- **Input:**

Of course, the REGEX plug-in cannot do anything until it is given some input. This is provided by means of three PowerPro **variables**:

- **rx_string**
- **rx_pattern**
- **rx_replace**

You must create these **variables** in PowerPro's namespace and assign them values before the plug-in is called. Only **rx_replace** can be neglected in some cases (if you're only interested in the return code's value, for example). Here is the description of each **variable**'s role:

rx_string should contain the string that you want to analyze, i.e. the string on which the regular expression will be applied;

rx_pattern should contain the regular expression that will be applied on the subject string;

rx_replace should contain the replacement string, i.e. the string that is going to replace whatever portion of the subject string (**rx_string**) that is matched by the regular expression (**rx_pattern**).

Here is a very simple but practical example:

rx_string = "Mary had a little lamb"
rx_pattern = "lamb"
rx_replace = "dog"
void_var = **plugin** (**"regex"**, **"replace"**)

Output => "Mary had a little dog"

Plain English: take the phrase "Mary had a little lamb", take the first occurrence of the word "lamb" and replace it with the word "dog".

Yet another example:

rx_string = "Mary had 52 little lambs"
rx_pattern = "[0-9]+"
rx_replace = "many"

Output => "Mary had many little lambs"

Plain English: take the phrase "Mary had 52 little lambs", take the first occurrence of a number with one or more digits ("52") and replace it with the word "many".

Much more complex match and/or replace operations can be made, but these require good knowledge of Regular Expressions syntax, which will not be explained in this document. It is fairly easy to find information about it all over the Web. There are a few variations of RegEx, so you're advised to look for "extended Regular Expressions" or "Perl-Compatible Regular Expressions". See the links provided at the end of this document (click here). PCRE have more features and are a bit more complex, but provide greater flexibility and accuracy. If you're familiar with PCRE and are eager to test it, go ahead! It works! If you're not familiar with PCRE, you can safely use regular expressions that do not include PCRE without the risk of incurring any incompatibility issue.

- **Services:**

The REGEX plug-in has four services: **match**, **matchg**, **replace** and **replaceg**.

match	"**MATCH**" operation. Checks whether **rx_string** contains any portion that matches **rx_pattern**. **The check is performed only once**, so **the first match only is found**. The result of the operation is stored automatically in two **output variables***.
matchg	"**MATCH GLOBAL**" operation. Checks whether **rx_string** contains any portion that matches **rx_pattern**. **The check is performed several times** if necessary, until **all possible matches are found**. The result of the operation is stored automatically in two **output variables***.
replace	"**REPLACE**" operation. Checks whether **rx_string** contains any portion that matches **rx_pattern**.

	If any match is found, **the first occurrence of that match is replaced** with the text stored in **rx_replace**. The result of the operation is stored automatically in two **output variables***.
replaceg	"**REPLACE GLOBAL**" operation. Checks whether **rx_string** contains any portion that matches **rx_pattern**. If any match is found, **every occurrence of that match is replaced** with the text stored in **rx_replace**. The result of the operation is stored automatically in two **output variables***.
* Detailed information on the **output variables** is provided below, in the "**Output**" section.	

- **Output:**

After the input **variables** are provided and the REGEX plug-in is run, it returns two useful values in two automatic **variables**:

- **rx_return**
- **rx_output**

rx_return contains the result of the expression's evaluation, in the form of a return code. Here is the meaning of each possible code:

0 = Full Match
1 = Partial Match
2 = No Match
3 = Error: Output Too Long
8 = Error: Invalid Pattern
9 = Error: No Input

A better explanation of each return code is given in the table "**Return Codes - Simple Examples**" below.

rx_output contains the output of the match or replace operation, i.e. an actual string that may be an exact copy of the original subject string, only part of it or a whole new string created by whatever values have been assigned to **rx_string**, **rx_pattern** and **rx_replace**.

In other words...

rx_return is the variable that will tell you what happened after the plug-in was run. It informs you (and the running script) whether or not there was a successful match in the operation.

- If you are replacing things in a string (**rx_string**), **rx_output** is the variable that will contain the original text, modified by the plug-in after looking for a given pattern (**rx_pattern**) in it and replacing everything that matches that **pattern** with some other text (**rx_replace**). Depending on what you feed the **variables**, many or no modifications at all may apply, turning the output string **rx_output** into something slightly different from the original string **rx_string**, completely different from the original string **rx_string** or leaving it just the way it was.
- If you are not replacing anything in a string, i.e. if you're just matching, **rx_output** is the variable that will contain what was matched by **rx_pattern**. This statement is in fact not true, but pretend it is for the time being and move on. This issue and the actual mechanism required to find a match will be explained later.

Note that neither do you need nor should you create the **rx_return** and **rx_output** variables. These are created by the REGEX plug-in **automatically** as soon as it is run.

Return Codes - Simple Examples

Return code	Meaning	Description
0	Full Match	• The pattern assigned to the **rx_pattern** variable matches the string assigned to the **rx_string** variable completely, i.e. they are exactly the same. **Example**: **rx_string** = "223557864" **rx_pattern** = "[0-9]+" **rx_return** = 0 The **pattern** looks for a sequence with one or more numbers and nothing else, so the entire **string** matches the **pattern**.
1	Partial Match	• Part of the string assigned to the **rx_string** variable matches the pattern assigned to the **rx_pattern** variable. **Example**: **rx_string** = "Today is 24/09/2002 03:28" **rx_pattern** = "[0-9]+" **rx_return** = 1 The **pattern** looks for a sequence with one or more numbers and nothing else. "Today is ", the slashes, the colon and the spaces do not match, so only a few portions of **string** match the **pattern**.
2	No Match	• No part whatsoever of the string assigned to the **rx_string** variable matches the pattern assigned to the **rx_pattern** variable. **Example**:

		rx_string = "Today is 24/09/2002 03:28" **rx_pattern** = ".* anytime!" **rx_return** = 2 The **pattern** looks for any sequence of characters (.*) followed by a space, the word "anytime" and an exclamation point. The exclamation point alone is enough to ruin all matching possibilities. The word "anytime" is not present in **rx_string** either, so there is no match at all.
3	Error: Output Too Long	• The output, i.e. the resulting string cannot be obtained because it exceeds the 263-character limit. **Example**: **rx_string** = "Today is 24/09/2002 03:28" **rx_pattern** = "[0-9]" **rx_replace** = "doo-be-doo-be-doo, it's the crazy Asian Y2Khai" **rx_return** = 3 According to the PowerPro manual, in the "Expressions" page, "**PowerPro supports strings of up to 263 characters; longer strings are truncated to this length**". The **pattern** looks for single occurrences of any number and nothing else. **rx_replace** in the third line causes every single occurrence of a number to be replaced with a rather long phrase. There are twelve occurrences of a number in the original **string** (24, 09, 2002, 03 and 28), so the long phrase will occur twelve times in the **output** string. The **output** string is therefore too long and cannot be displayed due to limitations in PowerPro's design.
8	Error: Invalid Pattern	• The pattern assigned to **rx_pattern** is invalid according to Regular Expressions syntax rules. **Example**: **rx_string** = "Today is 24/09/2002 03:28" **rx_pattern** = "(open parentheses... [or brackets..." **rx_return** = 8 Not closing or escaping opened parentheses, for example, is a mistake according to Regular Expressions syntax rules. The REGEX plug-in cannot process an incorrect regular expression, so it aborts the operation and returns the corresponding error code.
9	Error: No Input	• There is no input to handle. **Example**: **rx_string** = "" **rx_pattern** = "" **rx_return** = 9 Neither **rx_string** nor **rx_pattern** can be left empty. If either is omitted, the REGEX plug-in does not have enough data to evaluate and do its job.

- **Detailed examples:**

Let's see practical examples of each service, their behavior, return code and output in detail. You can always try your own examples with the REGEX plug-in test script, provided at the end of this document. Click here to take a look at it now, then press the browser's "**Back**" button to come back to this point. You may want to copy the test script, then run it as you read each example and try it yourself.

MATCH/MATCHG:

We use the **match** and **match global** services to check whether some text contains any sequence of characters, like words, phrases, numbers or symbols. You'd better keep two things in mind when using these services:

1. Return code **0** denotes a full match, **1** denotes a partial match and **2** denotes no match. Any number higher than these denote errors;
2. the **match** and **match global** services also have some replace-like behavior.

The first point is relevant if you just want to test the presence of some pattern in a string and don't need to know exactly what was matched.

The second point is relevant if you want to use the portion of the string that matches **rx_pattern**. That's due to the perhaps unexpected behavior of the REGEX plug-in, placing in the **output** the replacement string **rx_replace**, which the user himself has just inserted, whenever a match is found. For example:

Plain English: take the phrase "**Today is 24/09/2002, 03:28**", check if it contains the pattern "**[0-9]{4}**", i.e. a sequence of exactly four digits, and obtain the matched portion so as to verify exactly what four-digit number was found.

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{4}"
rx_replace = ""
void_var = plugin ("regex", "match")

rx_return => 1
rx_output =>
```

Hmmm... something is not right. Part of the **string** matches the **pattern**, so **rx_return** is **1**. But the **output** is empty. Why? Because the replacement string **rx_replace** is empty! Let's try again:

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{4}"
rx_replace = "XXX"
void_var = plugin ("regex", "match")

rx_return => 1
```

```
rx_output => XXX
```

OK... so **rx_replace** becomes **rx_output**. But we're just matching. We're not replacing. Return code is **1** so we have a match, but what is the four-digit number that the plug-in found? **How do we obtain the portion of the string that was matched?**

There are two ways to achieve that, and both require that we use the **match** service as if it were a replacement service:

- First, an obvious workaround: we can **(group)** the whole **pattern** and get the match with the first back reference:

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "([0-9]{4})"
rx_replace = "\1"
void_var = plugin ("regex", "match")

rx_return => 1
rx_output => 2002
```

OK, now we found the four-digit number matched by "**([0-9]{4})**"! It's "**2002**"!

- the second way is not a workaround, but an actual mechanism provided by the REGEX plug-in: **grouping or not** the whole **pattern**, the match can always be obtained with **the zeroth back reference**:

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{4}"
rx_replace = "\0"
void_var = plugin ("regex", "match")

rx_return => 1
rx_output => 2002
```

Not very intuitive, but no rocket science either. Mystery solved. That's what documentation is for!

Note that although we use the **match** service as if it were a replacement service, **no actual replacement has taken place** yet. We use the **rx_replace** variable, but **we're not changing** the original **string** at all, we're in fact just replacing whatever we place in **rx_replace** with the matched portion, thus extracting the matched portion. **Replacing involves changing** part or all of the original **string** and **getting back the entire string with the modifications**. The **match** service will not output anything besides the matched portion only, its output never includes the rest of the **string** if it does not match.

But, wait! There is more! We still haven't tried the **matchg** service. What if we want to find all occurrences of a pattern that may occur more than once? Let's see:

Plain English: take the phrase "**Today is 24/09/2002, 03:28**", check if it contains the pattern "**[0-9]{2}**", i.e. a sequence of exactly **two** digits, and obtain the matched portion so as to verify

```
exactly what two-digit number was found.

rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{2}"
rx_replace = "\0"
void_var = plugin ("regex", "matchg")

rx_return => 1
rx_output => 240920020328
```

The REGEX plug-in finds **24**, **09**, **20**, **02**, **03** and **28** and displays them all in a sequence, in the order they are found. If you want to separate the matches, just add a space to the **replacement** string:

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{2}"
rx_replace = "\0 "
void_var = plugin ("regex", "matchg")

rx_return => 1
rx_output => 24 09 20 02 03 28
```

If you want to unload the plug-in at the end of the script, here is how it's done:

```
void_var = plugin ("regex", "*")
```

REPLACE/REPLACEG:

We use the **replace** and **replace global** services to check whether some text contains any sequence of characters, like words, phrases, numbers or symbols, and replace the matched pattern with some other text. You may also want to keep two things in mind when using the replacement services:

1. The **replace** and **replace global** services also return the match status return codes. So they also have some match functionality.
2. The **replace** services actually replace text, i.e. they actually modify the original **string** and **output** a new string.

The first point means that we can also use the replace services to test matches. Usually, the replace services are used when some match is already expected and intended to be replaced right away. But it is possible to use replace operations to test a match and, if it is found, replace it with something else, all in a single operation.

The second point is related to something that has been said before in this document and

is repeated now:
"Note that although we use the **match** service as if it were a replacement service, no actual replacement has taken place yet. We use the **rx_replace** variable, but we're not changing the original **string** at all. **Replacing involves** changing part or all of the original **string** and **getting back the entire string with the modifications**. The **match** service will not output anything besides the matched portion only, its output never includes the rest of the **string** if it does not match."

Contrasting with the behavior described above, the **replace** and **replace global** services will take everything: they will take the whole **string**, replace the **matched** portion with whatever is provided in **rx_replace** and **output** not just the matches, but rather a combination of not matched and matched/replaced text. While the **match** service required that we use the **rx_replace** variable just to extract matches, this time the **rx_replace** variable actually does the job of replacing something and modifying the original **string**.

OK, let's see some examples:

Plain English: take the phrase "**Today is 24/09/2002, 03:28**", check if it contains the pattern "**[0-9]{4}**", i.e. a sequence of exactly four digits^. If it is found, we're going to replace it with "**XXXX**".

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{4}"
rx_replace = "XXXX"
void_var = plugin ("regex", "replace")

rx_return => 1
rx_output => Today is 24/09/XXXX, 03:28
```

"**2002**" is the only sequence of four digits in **rx_string**, so the year **2002** is replaced with "**XXXX**".

Let's see another example:

Plain English: take the phrase "**Today is 24/09/2002, 03:28**", check if it contains the pattern "**[0-9]{2}**", i.e. a sequence of exactly **two** digits. If it is found, we're going to replace it with "**XX**".

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{2}"
rx_replace = "XX"
void_var = plugin ("regex", "replace")

rx_return => 1
rx_output => Today is XX/09/2002, 03:28
```

There are several occurrences of two-digit sequences: **24**, **09**, **20**, **02**, **03** and **28**. But **24** is the first one to be found, so that's the one replaced with "**XX**". Because we used the **replace** service. If we want to replace **ALL** two-digit sequences with "**XX**", we use the **replace global** service:

Plain English: take the phrase "**Today is 24/09/2002, 03:28**", check if it contains the pattern "**[0-9]{2}**", i.e. a sequence of exactly two digits. If it is found, we're going to replace it with "**XX**".

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[0-9]{2}"
rx_replace = "XX"
void_var = plugin ("regex", "replaceg")

rx_return => 1
rx_output => Today is XX/XX/XXXX, XX:XX
```

OK, just one more example:

Plain English: take the phrase "**Today is 24/09/2002, 03:28**", check if it contains the pattern "**[A-Za-z]+**", i.e. a sequence of characters with undefined length which may include upper and lower case letters and spaces. If it is found, we're going to replace it with "", i.e. nothing.

```
rx_string = "Today is 24/09/2002, 03:28"
rx_pattern = "[A-Za-z ]+"
rx_replace = ""
void_var = plugin ("regex", "replaceg")

rx_return => 1
rx_output => 24/09/2002, 03:28
```

So, not the entire string matches the **pattern**. Only part of it, "**Today is** ". We told the REGEX plug-in to replace it with nothing, so "**Today is** " is just deleted. The rest is kept.

And this is it. Go ahead and do your own experiments! Use the script below to run any tests you feel like. The script is ready to be used. Just copy it, paste it in a text file (preferably with the ".powerpro" extension) and run it. The parts highlighted with a white background can and should be changed to suit your preferences.

REGEX PLUG-IN TEST SCRIPT

```
;rx_string = the string on which you want to run the expression
;rx_pattern = the regular expression
;rx_replace = replacement string
;myServices: match, matchg, replace, replaceg

rx_string = "Today is 24/09/2002 03:28"
rx_pattern = "[0-9]+"
rx_replace = "XX"
myService = "replace"
;# ------------------------------------------------------------
x0 = plugin ("regex", myService)
;# ------------------------------------------------------------
If (rx_return == 0) do
      myMsgPopup = rx_return ++ " - Full match!"
      ElseIf (rx_return == 1)
      myMsgPopup = rx_return ++ " - Partial_match"
      ElseIf (rx_return == 2)
      myMsgPopup = rx_return ++ " - No_match"
```

```
      ElseIf (rx_return == 3)
      myMsgPopup = rx_return ++ " - Output_too_long"
      ElseIf (rx_return == 8)
      myMsgPopup = rx_return ++ " - Invalid_regex. Pay attention!!!!"
      ElseIf (rx_return == 9)
      myMsgPopup = rx_return ++ " - No_input."
      Else
      myMsgPopup = rx_return ++ " - Yikes! Weird return code!"
      EndIf
;# ------------------------------------------------------------
Debug RETURN CODE: $(myMsgPopup), OUTPUT: $(rx_output)
;# ------------------------------------------------------------

;THE NEXT LINES ARE OPTIONAL. USE THEM IF YOU WANT
;A LOG FILE CALLED "DEBUG-REGEX.TXT" ON YOUR DESKTOP
myLogFile = "c:\windows\desktop\debug-regex.txt"
*Exec ToFile $(myLogFile) Testing myService: $(myService)
*Exec ToFile $(myLogFile)
*Exec ToFile $(myLogFile) rx_string = $(rx_string)
*Exec ToFile $(myLogFile) rx_pattern = $(rx_pattern), rx_replace = $(rx_replace)
*Exec ToFile $(myLogFile) rx_return = $(rx_return), rx_output = $(rx_output)
*Exec ToFile $(myLogFile)
*Exec ToFile $(myLogFile) ====================================
*Exec ToFile $(myLogFile)
```

- **Getting started with Regular Expressions:**

PCRE man page in txt format:
http://www.pcre.org/man.txt
PCRE man page in HTML format:
http://www.fifi.org/cgi-bin/man2html?pcre+7
PCRE introduction page at PHP.net:
http://www.php.net/manual/en/pcre.pattern.syntax.php
Perl Regular Expressions (not strictly PCRE) at Perldoc.com:
http://www.perldoc.com/perl5.6/pod/perlre.html

Visual Regexp is not PCRE-compatible, but it's a terrific tool for any user of Regular Expressions, especially newbies. It supports ARE (Advanced Regular Expressions), which are more sophisticated than plain POSIX regex but less sophisticated than PCRE:
http://laurent.riesterer.free.fr/regexp/

- **Good regex, bad regex:**

Many applications support Regular Expressions, but that's not always good. Good Regular Expressions are those that comply with common, widely-accepted standards. **Applications that support good regex** include:

- sed, super-sed, vi, Vim, Emacs, Perl, PHP, Tcl, Python, Ruby, VBS, SciTE, Crimson-editor, Visual Regexp.

The REGEX plug-in is fully compliant with POSIX "Extended" Regular Expressions and PCRE. If you can use Regular Expressions in the above applications successfully, you probably know actual Regular Expressions and should have no problem with RE syntax while using the PowerPro REGEX plug-in.

Bad Regular Expressions are those that do not comply with common, widely-accepted standards. Applications that use bad Regular Expressions typically have an incorrect and/or incomplete regex implementation and try to impose an arbitrary system, forcing their users to learn otherwise idiosyncratic syntax that has no use whatsoever in any other program or programming language. **Applications that make use of bad regex** include:

- MS Word 2000 or earlier, UltraEdit, Proxomitron.

The REGEX plug-in is fully compliant with POSIX "Extended" Regular Expressions and PCRE **only**. If you can only use successfully whatever the above applications call "Regular Expressions", you probably know little or nothing about Regular Expressions and will have to learn the real thing before you can use the PowerPro REGEX plug-in.

Of course, neither of the two lists above is comprehensive.

POWERPRO REGEX PLUG-IN

Designed by: Julien Pierrehumbert and Luciano Espirito Santo

Coded by: Julien Pierrehumbert

Documentation by: Luciano Espirito Santo

Contacting the authors:
If you don't have our email addresses, use the PowerPro mailing list.

Known issues:
Using lookbacks (a PCRE-specific feature) with the global services might lead to unexpected results.

PowerPro written by: Bruce Switzer

Regular Expressions GNU library by: Philip Hazel

Special thanks to:
Bruce Switzer