PowerTerm Help Contents

How to...

Click on a topic in the list below to view a description:

Show/Hide Buttons

Show/Hide the Power Pad

Start/Stop Keyboard Input Recording

Run the Recorded Macro

Execute a Script File

Edit/Create a Script File

Record Manual Operations to Automatically Create a Script

Execute Individual Script Commands

Set the Dimensions of the Window

Store the Session in a Log File

Input the Data from the Log File

Programming the Soft Buttons

Along the bottom of the PowerTerm screen are twelve programmable buttons, named by default from F1 to F12. These buttons can be renamed and/or programmed to carry out specific scripts. There is no connection between the soft buttons and the keyboard mapping or the Power Pad.

To program a soft button:

1. Click with the right mouse button on the soft button that you want to program. The function button window is displayed.
2. Enter the function description (the new name that will appear on the button) and click OK.
3. Enter the script command to be run by this button, for example, `send <f2>`, and click OK. The button is now displayed with its new name and clicking on the button with the left mouse button will execute the newly defined script.

{button ,AL(`Keyboard Mapping;Script Commands;Using the Power Pad',0,`',`sec')} Related Topics

Using the Power Pad

The Power Pad contains an adjustable number of programmable buttons that can be renamed and/or programmed to carry out specific scripts. There is no connection between the Power Pad and the keyboard mapping or the soft buttons.

To program the Power Pad:

1. Display the Power Pad by selecting Options | Show Power Pad.
2. Click with the right mouse button on the Power Pad button that you want to program. The Power Pad Button window is displayed.
3. Enter the Power Pad button description (the new name that will appear on the Power Pad button) and click OK.
4. Enter the script command to be run by this Power Pad button, for example, `send <f2>`, and click OK. The Power Pad button is now displayed with its new name and clicking on the button with the left mouse button will execute the newly defined script.

{button ,AL(`Changing the Power Pad Display;Keyboard Mapping;Programming the Soft Buttons;Script Commands',0,`',`sec')} Related Topics

Changing the Power Pad Display

You can adjust the number of buttons displayed in the Power Pad, by specifying the number of rows and columns to be displayed.

To adjust the number of buttons displayed in the Power Pad:

1. Select Options | Power Pad Setup to display the Power Pad setup window.
2. Enter the number of rows and columns of the buttons that you want to be displayed in the Power Pad.
3. Click OK. The Power Pad will now only display the buttons contained in the selected number of rows and columns.

{button ,AL(`Using the Power Pad',0,`',`sec')} Related Topics

Restore VT Terminal Defaults

To reset the VT terminal settings to the factory defaults:

- Select Terminal | Reset.

This command does not apply to PowerTerm's exclusive terminal parameters, such as color.

To restore the default factory parameters, including colors:

- Select File | New Terminal Setup

If you have changed terminal parameters since the last save, and before selecting File | New Terminal Setup, PowerTerm displays a warning message asking whether or not you want to save the latest changes. The message points to the terminal settings file currently loaded (PTDEF.PTS is the default).

Select one of the following options:

- Click Yes. PowerTerm saves the latest changes and then executes New Terminal Setup.
- Click No. PowerTerm cancels the latest changes and then executes New Terminal Setup.
- Click Cancel to cancel the operation.

Open an Existing Terminal Settings File

To open a existing terminal settings file:

- Select File | Open Terminal Setup.

Save Terminal Settings to Current Setup File

To save the terminal settings to the current setup file:

Select File | Save Terminal Setup to save the current terminal settings to the currently loaded terminal settings (.PTS) file.

{button ,AL(`Preferences;Setup for VT and General Types;Updating PTDEF.PTS',0,`',`sec')} Related Topics

Save Terminal Settings to a Specific File

To create a new terminal settings file:

1. Select File | Save Terminal Setup As.... PowerTerm displays the Save File As dialog box.
2. In the File Name text box type the file name that you want, with a .PTS extension (for example: `DG.PTS`).
3. Click the OK button.

{button ,AL(`Creating a New Parameter File;Setup for VT and General Types',0,`',`sec')} Related Topics

Print Selected Text or Contents of the Work Area

To print selected text or the contents of the work area:

- Select File | Print Screen.

{button ,AL(`Printer Parameters',0,`',`sec')} Related Topics

Select a Printer and Set Printing Parameters

To select a printer and set printing parameters:

- Select File | Print Setup to open the Print Setup dialog box. This dialog box contains and enables access to a set of printing parameters. The parameters change according to the printer you select. For details, consult your printer documentation.

  The Default Printer parameter enables you to send the output to the default printer selected under Windows. The Specific Printer parameter allows you to select one of the currently installed printers. For details about installing a new printer, consult your Windows documentation.

{button ,AL(`Printer Parameters',0,`',`sec')} Related Topics

Save All the Data Displayed in the Work Area and Print It

To start saving all the data displayed in the work area:

1. Select File | Start Auto Print. The Start Auto Print command starts accumulating incoming data (while it is displayed on the screen). After selecting this command, the menu entry changes to Stop Printing.
2. Select File | Stop Printing. The Stop Printing command prints all the data accumulated in the printing buffer of the slave printer or in the Auto print buffer. If data was buffered with a printing request and communication failed before the data was sent to the slave printer, select this command to print the accumulated information.

Execute a Form or Line Feed on the Printer

To execute a form feed on the printer:

- Select File | Form Feed.

To execute a line feed on the printer:

- Select File | Line Feed.

{button ,AL(`Printer Parameters',0,`',`sec')} Related Topics

Open a New Instance of the PowerTerm Window

To open a new PowerTerm window:

- Select File | New Terminal Window.

{button ,AL(`File Menu - New Terminal Window Command',0,`',`sec')} Related Topics

Exit PowerTerm

You can exit PowerTerm in one of the following ways:

- Select File | Exit (ALT+F4)
- Double-click the Control-menu box

If you have changed the terminal settings, PowerTerm displays a warning message asking you if you want to update the terminal settings file (.PTS). The message will point to the name of the file currently loaded (PTDEF.PTS if you used the default). Click OK to update the file, or Cancel to exit without saving the latest changes.

{button ,AL(`User-Initiated Fast Exit',0,`',`sec')} Related Topics

Select the Screen Contents

To select the contents of the entire screen:

- Select the Edit | Select Screen command (ALT+F10).

{button ,AL(`Selecting Text Items',0,`',`sec')} Related Topics

Clear the Work Area

To clear the work area:

- Select the Edit | Clear Screen command (ALT+F11).

Display a Mirror Image of the Work Area

To display a mirror image of the contents of the work area:

- Select the Edit | Reverse Screen command (ALT-F12).

Copy the Selected Information to the Clipboard

To copy selected information to the clipboard:

- Select the Edit | Copy command if the Automatic Copy option in the Edit menu is not active.

{button ,AL(`Automatic Copy',0,`',`sec')} Related Topics

Paste the Clipboard Contents into the Work Area

To paste the contents of the clipboard into the work area:

- Click the right mouse button or select the Edit | Paste command. This operation is equivalent to actually typing the contents of the clipboard on the host screen.

Copy the Selected Information to a Specific File

To copy selected text to a specific file:

- Select the Edit | Copy to File command. PowerTerm opens a dialog box in which you specify the file name.

Note: If no text is selected, the entire screen is written to the file.

Automatically Copy Selected Information to the Clipboard

To automatically copy text to the clipboard:

1. Make sure that the Automatic Copy option is active.
2. Select the text that you want to copy. The text will automatically be copied to the clipboard.

{button ,AL(`Copying the Selected Information to the Clipboard;Selecting Text Items',0,`',`sec')} Related Topics

Define Settings for Terminal Emulation

To define terminal emulation settings:

1. Select Terminal | Setup.
2. Click the Emulation tab.
3. Select the emulation type you require from the list of supported emulations. The number of Setup tabs change according to the emulation you clicked.
4. Click on each tab and define the settings in its corresponding page. For example, define the work area display by clicking the Display tab and entering information in the Display page.
5. When you have defined your settings, save them by selecting File | Save Terminal Setup.

{button ,AL(`Defining Parameters Per Session;Setup for 3270 and 5250;Setup for VT and General Types',0,`',`sec')} Related Topics

Set the System to be On Line or Off Line

To go on-line or off-line:

1. Select Terminal | Setup | On Line.

   When communication is established, this command is checked.

2. Select the checked command to go off line. When the system is off line, all keyboard input is directed to the screen (it is not sent to the host).

3. Select the command again to check it and resume normal operations.

The on line/off line state is indicated in the status bar.

{button ,AL(`Status Indicators',0,`',`sec')} Related Topics

Freeze or Unfreeze the Screen

To stop communications and freeze the screen:

1. Select the Terminal | Hold Screen command.
2. To resume normal operation, select the command again.

{button ,AL(`Status Indicators',0,``,`sec')} Related Topics

Define Connection Parameters and Establish a Connection

To define connection parameters and connect to the host computer:

1. Select Terminal | Setup.
2. Click the Emulation tab and select the terminal you want to emulate.
3. Click the General tab.
4. Define the parameters in the displayed dialog box.
5. Click the OK button to accept the new settings.
6. Select Communication | Connect. The Connect dialog box is displayed, in which you can define the connection settings.
7. In the Session Type list, select the communication type you require. Notice that for some communication types a different set of parameters is displayed in the corresponding group box, on the right.
8. Click Connect. This connects you to the host computer. The communication mode now appears beside the system name on the desktop's title bar. When communication ends, the mode name disappears.

Note: The parameters that you define will remain in force for the current session only, unless you save them.

{button ,AL(`DDE Commands;Defining Connection Parameters;Preferences',0,`',`sec')} Related Topics

Modify Connection Parameters for COM Type Communication

To modify the connection parameters for the COM session type only:

1. Select Communication | Modify Connection. The Modify Connection dialog box is displayed and the title bar displays the session type.
2. Change the session parameters (but not the Session Type), and the Login Command File.

{button ,AL(`Script Language - Session Command',0,`',`sec')} Related Topics

Disconnect a Connection

To disconnect the communication session:

- Select Communication | Disconnect.

Note: This option changes to Connect when the session has been disconnected.

Reset the Communication Port for COM Type Communication

To reset the communication port:

- Select Communication | Reset Communication.

Dial a Specific Phone Number for COM Type Communication

To dial a specific phone number:

1. Select Communication | Utilities | Dial.
2. Type the phone number in the displayed dialog box.
3. Click OK. PowerTerm starts dialing immediately.

Send a Break for COM Type Communication

To send a break:

- Select Communication | Utilities | Break or press CTRL+BREAK

Set/Clear DTR (Data Terminal Ready) Signals

To set or clear DTR signals:

1. Select Communication | Utilities.
2. Select Set DTR or Clear DTR.

Set/Clear RTS (Ready to Send) Signals

To set or clear RTS signals:

1. Select Communication | Utilities.
2. Select Set RTS or Clear RTS.

Define Host and PC Data Types for File Transfer

To define host and PC data types:

1. Select Communication | File Transfer Setup
2. Select the host and PC data types from the drop-down lists in the File Transfer Setup window.

Note: You must determine the data types before you perform any file transfers.

Send a File from the Host to the PC

To send a non- ASCII file from the host to the PC:

1. Establish communication.
2. In the work area, type the command the host uses to receive a file.
3. Select Communication | Receive File and click one of the tabs, except for the ASCII tab.
4. Specify a directory, if required.
5. If you want the PC receiving file to have a different name from the file on the host, type in a file name in the File Name field. Leave as *.* (or empty) if you want to receive the data into a file that carries the same name as the name used by the host.
6. Click the OK button. The received file will be saved on the PC disk under the name you specify (in the directory you selected). The PC starts to receive the file.

To send an ASCII file from the host to the PC:

1. Establish communication.
2. In the work area type in the command that displays the file (examples: for UNIX - cat filename; for VAX type filename). Do not press ENTER.
3. Select Communication | Receive File and click the ASCII tab.
4. In the File Name field, enter the name of the PC file that will store the data, and click OK.
5. Press ENTER to activate the command.
6. At the end of the capture process, select the Communication | Stop Receiving ASCII File command.

Send a File from the PC to the Host

To send a non- ASCII file from the PC to the Host:

1. Establish communication.
2. In the work area type the host's file reception command.
3. Select Communication | Send File.
4. Click a tab other than ASCII.
5. Select the file to be sent and click the OK button. The PC starts to send the file.

To send an ASCII file from the PC to the Host:

1. Activate the command that will accept the data on the host (examples: .for UNIX -cat > filename, or entering an editor and moving to data entry state within the editor).
2. Select Communication | Send File, and click the ASCII tab.
3. Enter the PC file name and click OK.

View the Default Keyboard Mapping

To see the active key mapping scheme:

1. Select Options | Keyboard Map.
2. Slide the mouse pointer over the different keys. The bottom line shows you the corresponding PC/Terminal keys. For example, if you point on the “t “ key of the Terminal keyboard, you will see that the corresponding PC key is “T”.

Map the Keyboard

To map the keyboard:

- Select Options | Keyboard Map. The Keyboard Mapping dialog box is displayed, in which you can map PC keys to host keys.

Note: Keyboard mapping definitions are stored in the terminal parameters (PTK) file.

To map a PC key to a host key:

- Drag a key from the upper terminal keyboard to a PC key on the lower keyboard. You can click on the Shift or Control keys to display additional key functions that you can then also drag into the PC keyboard, as described above.

To cancel a definition:

- Drag the PC key definition to the wastebasket. This restores the default function of the PC key.

To replace a PC key with another PC key:

- Drag the required key onto the key that it will replace. This cancels the function of the original key. To cancel the replacement, drag the replaced key back to its original position.

To restore the default mapping:

- Click the Defaults button.

{button ,AL(`Key Command (Script Language)',0,`',`sec')} Related Topics

Program the Power Pad

To program the Power Pad to carry out specific scripts:

1. Select Options | Show Power Pad to display the Power Pad.
2. Click with the right mouse button on the Power Pad button that you want to program. The Power Pad Button window is displayed.
3. Enter the Power Pad button description (the new name that will appear on the Power Pad button) and click OK.
4. Enter the script command to be run by this Power Pad button, for example, `send <f2>`, and click OK. The Power Pad button is now displayed with its new name and clicking on the button with the left mouse button will execute the newly defined script.

{button ,AL(`Changing the Power Pad Display;Script Commands',0,`',`sec')} Related Topics

Change the Power Pad Display

To adjust the number of buttons displayed in the Power Pad:

1. Select Options | Power Pad Setup to display the Power Pad setup window.
2. Enter the number of rows and columns of the buttons that you want to be displayed in the Power Pad.
3. Click OK. The Power Pad will now only display the buttons contained in the selected number of rows and columns.

Show/Hide the History Scroll Bar

To show/hide the history scroll bar:

1. Select Terminal | Setup.
2. Click the Display tab.
3. Click in the History Scroll Bar checkbox to select it. Click again to deselect.

When the option is selected, a vertical scroll bar is displayed along the right edge of the PowerTerm desktop window. This enables you to scroll the window to see data previously brought to display. If the host transmits during scrolling, the display automatically scrolls back to its current position. Notice that PowerTerm begins to store history after you display the scroll bar.

Show/Hide the Menu Bar

To hide the menu bar:

- Select Options | Hide Menu Bar. This removes the menu bar altogether.

To restore the menu bar:

- Click the Control-menu box and select Restore Menu.

Show/Hide the Tool Bar

To show/hide the tool bar:

- Select Options | Hide Tool Bar. The menu option becomes Show Tool Bar so that you can select it to redisplay the tool bar.

Show/Hide the Status Bar

To show/hide the status bar:

- Select Options | Hide Status Bar. The menu option becomes Show Status Bar so that you can select it to redisplay the status bar.

{button ,AL(`Status Indicators',0,``,`sec')} Related Topics

Show/Hide the Soft Buttons

To show/hide the soft buttons:

- Select Options | Hide Buttons. The menu option becomes Show Buttons so that you can select it to redisplay the buttons.

Show/Hide the Power Pad

To show/hide the Power Pad:

- Select Options | Show Power Pad to display the Power Pad. The menu option becomes Hide Power Pad so that you can select it to hide the Power Pad.

{button ,AL(`Altering the Power Pad Display;Programming the Power Pad',0,`',`sec')} Related Topics

Start/Stop Keyboard Input Recording

To start/stop recording:

1. Select Macro | Start Recording (CTRL+ALT+F9) to begin a macro recording session. PowerTerm activates the macro display area - the right section of the desktop's bottom line. As you type, the characters are shown in the macro display area. When the area is full, scrolling occurs.
2. Select Stop Recording (CTRL+ALT+F9) to stop recording your keyboard input.

Run the Recorded Macro

To run the recorded macro and transmit it to the host:

1. Select Macro | Activate Macro.
2. Press ALT+F9 to send the recorded characters to the host.

Execute a Script File

To run a selected script file:

1. Select Script | Run Script. The Run Script dialog box is displayed. It lists the files in the PowerTerm directory that carry the .PSL extension (however, you can assign any name and extension that comply with DOS file naming conventions).
2. To run a script, double-click the required script file in the list.

Edit/Create a Script File

To edit an existing script file:

1. Select Script | Edit Script. The Edit Script dialog box is displayed.
2. Select the required script file by double-clicking the file in the Files list. This activates Notepad and displays the script.
3. Edit the script as required.
4. Select the Save option in Notepad's File menu to save the edited file.

To create a new file:

1. Select Script | Edit Script. The Edit Script dialog box is displayed.
2. Type a name for the new script file in the File Name box. (You can type any name and extension that comply with DOS file naming conventions. However, we recommend that you use a .PSL extension since PowerTerm defaults to this extension to list the existing files.)
3. Click OK. A message box is displayed, asking you whether you want to create a new file.
4. Click Yes. The Notepad window is displayed.
5. Type in the script and select File | Save to save your new script file.

{button ,AL(`PowerTerm Script Language (PSL)',0,`',`sec')} Related Topics

Record Manual Operations to Automatically Write a Script

PowerTerm enables you to record the manual operations you perform in the emulation screen to create a new script file.

To start script recording:

1. Select Script | Start Script Recording. The Record Script dialog box opens.
2. In the File name box, type the name of the script file with the .PSL extension. The manual operations you perform will be stored in this file. Click OK.
3. Perform the manual operations that you want to record. If you do not want to include certain operations, select Script | Pause Script Recording. This will pause the script recording process.
4. To resume script recording, select the Script | Start Script Recording.
5. When you have performed all the operations you want to store, select Script | Stop Script Recording. You have now created a script file that can be run any time to repeat the operations that you recorded. This new file will appear in the list in the Edit Script dialog box, enabling you to edit it, if required.

Execute Individual Script Commands

To execute individual script commands:

1. Select Script | Script Command.
2. Type the name of the script command you want to execute in the displayed Script Command dialog box.
3. Click OK. PowerTerm will execute the specified script command.

{button ,AL(`PowerTerm Script Language (PSL)',0,`',`sec')} Related Topics

Specify the Work Area Display Settings

To set the number of columns in the window width (the number of characters per displayed line):

1. Select Terminal | Setup.
2. Click the Display tab.
3. In the Dimensions box, click either the 80 or 132 standard options or click the Other option and enter the required number of columns. Click OK.

To specify the number of lines per screen:

1. Follow steps 1 and 2 above.
2. In the Dimensions box, in the Lines per Screen field, specify the number of lines to be displayed in the work area. Click OK.

{button ,AL(`Display Tab in the Terminal Setup Window',0,`',`sec')} Related Topics

Store the Session in a Log File

To start storing data:

1. Select Options | Start Trace. Raw data is stored in the CAPTURE.LOG file, while formatted data with readable escape sequences is stored in the TRACE.LOG file.
2. Select the Stop Trace command to stop storing the data.

Input the Data from the Log File

To input the contents of the CAPTURE.LOG file to the PC window:

- Select Options | Input Trace

Getting Started

Click on a topic in the list below to view a description of that topic:

Please note the following list of PowerTerm versions showing the terminal types supported by each version:

PowerTerm 320	Supports VT terminal types up to VT320 and General terminal types.
PowerTerm 525	Supports all VT terminal types.
PowerTerm InterConnect	Supports all available terminal types, including VT, General and IBM.

Starting PowerTerm

To start PowerTerm:

In Windows 3.x

- In the Program Manager, double click on the PowerTerm icon in the PowerTerm 4.0 Program Group. The PowerTerm desktop is displayed.

In Windows 95

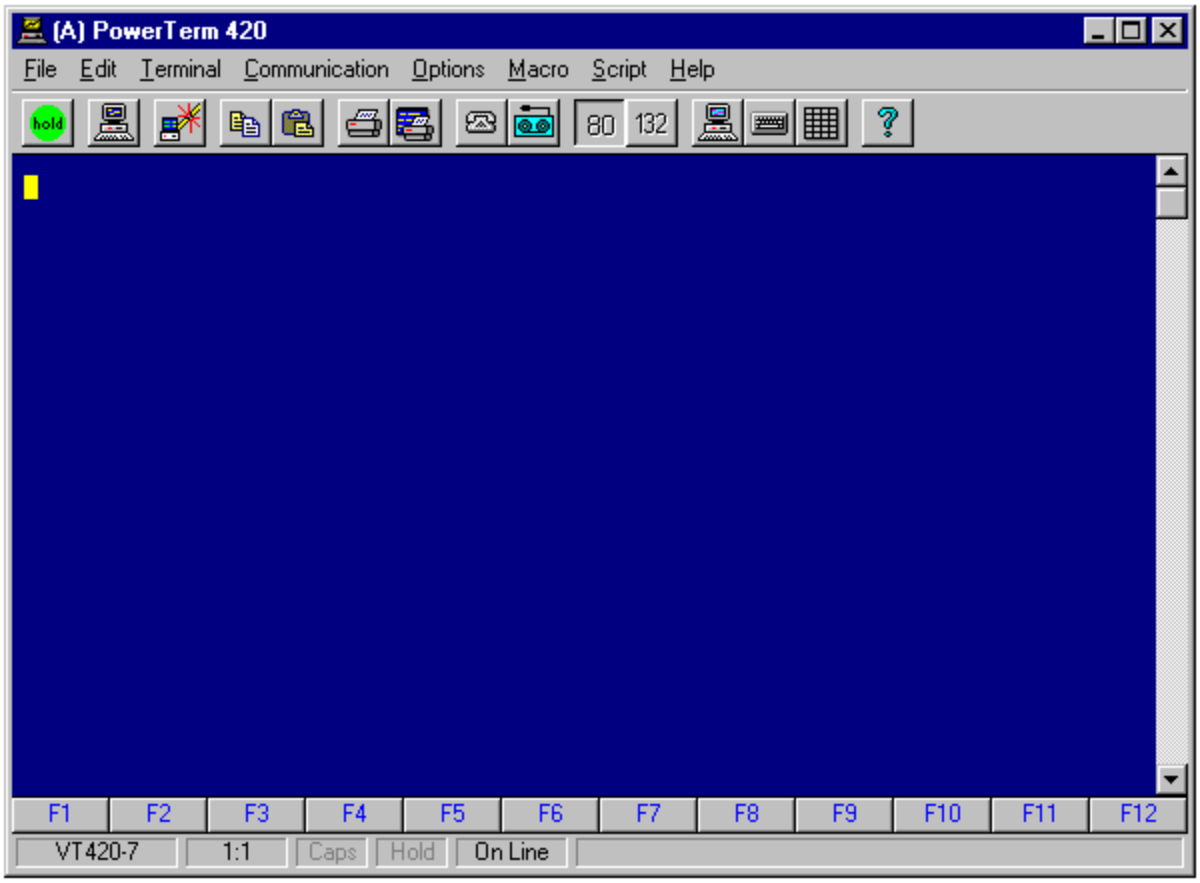
- Select PowerTerm from the Start | Programs menu. The PowerTerm desktop is displayed.

The PowerTerm Desktop

When you start PowerTerm, the desktop is displayed. The desktop is your PowerTerm work area.

Click on an desktop area in the list or in the picture below to see a description of that area:

Title Bar
Menu Bar
Minimize Button
Maximize Button
Close Button
Toolbar
Window Size
Work Area
History Scroll Bar
Terminal Indicator
Cursor Position Counter
Status Indicators
Soft Buttons Area
Macro/Message Display Area



The title bar displays the application name “PowerTerm”. During a communication session, the session type and/or the host name is displayed beside the application name. When communication ends, the session type disappears from the title bar.

The menu bar lists the PowerTerm drop-down menus.

The minimize button minimizes the PowerTerm desktop but leaves PowerTerm running. A minimized window becomes a button on the task bar. You can click that button to re-open the window. Use the minimize button to make room on your Windows desktop.

The maximize button enlarges a window so that it fills the entire screen.

The Close button closes the PowerTerm window and exits PowerTerm.

The toolbar contains buttons (icons) representing shortcuts to frequently used menu commands.

Drag the window borders and the window corners to change the size of the PowerTerm window. As you change the window size, the characters that appear in the work area are scaled up or down, so that all the information always remains in view.

The work area is where you type data and receive data from the host. During an emulation session, the work area looks like the emulated terminal.

The history scroll bar enables you to scroll the window and view previously displayed data. To display the scroll bar, select the Terminal | Setup command, click the Display tab, and check the History Scroll Bar box.

The terminal indicator shows the current emulation. This name changes when you select a different terminal.

The cursor position counter indicates the current line and column position of the text cursor.

The soft buttons area contains a series of buttons that can be programmed to execute commands with the mouse.

The macro/message display area shows system messages, or the macro sequence - as it is typed in the work area.

PowerTerm Workflow

The following is an example of a typical workflow for using PowerTerm. Click on a step below to display a full description of that step.

1. Start PowerTerm.
2. Define settings for terminal emulation (including terminal type, terminal ID, and connection options).
3. Map the PC keyboard to the terminal keyboard.
4. Save the settings file.
5. Define connection parameters and connect to the host.
6. Select the session type and appropriate host parameters.
7. Display any PowerTerm desktop components that you may require (for example, Power Pad, soft buttons, history scroll bar).
8. At the end of the session, disconnect and either exit PowerTerm or begin a new session.

Overview

PowerTerm is a full functional terminal emulator for MS-Windows. It is designed to emulate terminals of various types. PowerTerm includes the PowerTerm Script Language that enables you to automate tasks.

The following topics describe basic PowerTerm functions:

Keyboard Mapping

Click on a topic in the list below to view a description of that topic.

Communication Methods

For a PC-host interaction you need to determine two sets of parameters: the terminal parameters and the communication parameters. The terminal parameters are set via the Terminal menu, and the communication parameters are defined by means of Communication menu commands.

To conduct a PC-host interaction you can do one of the following:

- Define parameters per session
- Use stored settings

Before you start a PC-host interaction make sure that:

- The system is on line (if not, select Terminal | Online).
- The system is not on hold (if not, select Terminal | Hold Screen).

Defining Parameters Per Session

After you activate PowerTerm you can change the terminal and communication parameters. These parameters will remain in force for the current session, unless you save them.

1. Select Terminal | Setup.
2. Under the Emulation tab click the terminal you want to emulate.
3. Click the General tab.
4. Change any of the parameters shown in this dialog box.
5. Click the OK button to accept the new settings.
6. Select Communication | Connect.
7. In the Session Type list select the communication type you want. Notice that for some communication types a different set of parameters is displayed in the corresponding group box, on the right.
8. Click Connect. This connects you to the host computer. The communication mode now appears beside the system name on the desktop's title bar. When communication ends, the mode name disappears.

Using Stored Settings

You can store both the terminal parameters and the communication parameters for repeated use.

PowerTerm stores terminal parameters in a file that carries the .PTS extension. The default settings are stored in a file named PTDEF.PTS. This file is located in the PowerTerm directory. The communication parameters are stored in internal session lists.

If you frequently switch between hosts of different types, you can create different .PTS files and different communication sessions. If you frequently work with the same emulation, you can change the terminal parameters (if needed) once and save them to PTDEF.PTS.

{button ,AL(`Creating a New Parameter File;Starting a Session With Stored Parameters;Storing a Communication Session;Updating PTDEF.PTS',0,`',`sec')} Related Topics

Updating PTDEF.PTS

When you start PowerTerm, PTDEF.PTS is loaded automatically.

If you change the current terminal settings, you can save them by simply choosing the File | Save Terminal Setup command.

If you don't save the new settings, a warning message is displayed when you request to exit or to open another settings file.

Click Yes to save the modified terminal parameters to PTDEF.PTS, and exit PowerTerm. Select Yes if you frequently work with the same host type under the parameters you've just defined. The next time you load PowerTerm, it will automatically read these parameters.

Click No to restore the last saved parameters and exit PowerTerm.

Creating a New Parameter File

If you frequently switch between hosts that require different terminal settings, you can create different .PTS files. Then, when you start PowerTerm you can load the file you want, using the File | Open Terminal Setup command.

To save the current terminal settings to a file, follow these steps:

1. Select the File | Save Terminal Setup As... command. PowerTerm displays the Save File As dialog box.
2. In the File Name text box type the name you want, with the .PTS extension (for example: **DG.PTS**).
3. Click the OK button.

Storing a Communication Session

To store a communication session follow these steps:

1. Select Communication | Connect. The Connect dialog box is displayed.
2. Determine the session parameters.
3. Click the Save As... button. PowerTerm displays the Save Session dialog box.
4. In the Session Name text box, PowerTerm displays a default session name, derived from communication parameters. You can type a different name over the default name, if required.
5. Click the OK button.

Starting a Session With Stored Parameters

When you start PowerTerm, it automatically loads the PTDEF.PTS file and defaults to the communication parameters currently shown in the Connect (Communication | Connect) dialog box.

To start communication with stored parameters, follow these steps:

1. Select the File | Open Terminal Setup command. PowerTerm displays the Open File dialog box. The Files list shows the .PTS files contained in the PowerTerm directory.
2. Double-click the required file. The Open File dialog box closes automatically. The terminal mode is displayed in the lower-left corner of the screen.
3. Select Communication | Connect. PowerTerm displays the Connect dialog box. It lists the various sessions you saved.
4. Double-click the session you want. This starts communications.

Using a Script

You can easily create one of the scripts offered in the **session** command, and start a communication session automatically, as you activate PowerTerm.

In addition, PowerTerm includes several standard scripts designed for frequently used tasks.

{button ,AL(`PowerTerm Script Language (PSL);Script Commands',0,`',`sec')} Related Topics

Ending a Session

Communication may end in one of the following ways:

Automatic closing

Optional closing

User-initiated closing

User-initiated fast exit

When the PC and the host communicate, the communication mode appears beside the system name on the desktop's title bar. When communication ends, the mode name disappears.

Automatic Closing

Using the Preferences tab in Terminal | Setup you can instruct PowerTerm to close down when a session ends normally. If you modified the terminal parameters, the .PTS file save message appears before the automatic closing.

If communication is aborted, a message to this effect is displayed.

{button ,AL(`Optional Closing',0,`',`sec')} Related Topics

Optional Closing

If the automatic closing is not active, or if the communication session aborted, PowerTerm displays the following message: “Session closed (<exit code>). Hit ENTER to restart session.”

<exit code> may be one of the following:

- 0 (zero). Communication ended successfully.
- A number other than 0. Communication aborted. The exit code points to the error that caused the problem.

If you press ENTER communication is re-established based on the current terminal and communication parameters.

User-Initiated Closing

At any time you can close a session by simply choosing the Communication | Disconnect command.

User-Initiated Fast Exit

If you request fast exit (for instance, by pressing ALT+F4) while communication is in progress, PowerTerm's reaction depends on what you defined in the Communication setup dialog box.

If the parameter is checked, PowerTerm closes the session and exits. If not, a message is displayed enabling you to decide what to do next.

Exiting PowerTerm

You can exit PowerTerm in one of the following ways:

- Select File | Exit (ALT+F4)
- Double-click the Control-menu box

If you changed the terminal settings, PowerTerm displays a warning message asking you if you want to update the terminal settings file (.PTS). The message will point to the name of the file currently loaded (PTDEF.PTS if you used the default). Click OK to update the file, or Cancel to exit without saving the latest changes.

Default Key Mapping

To see the active key mapping scheme, open the Options menu and select the Keyboard Map command.

Slide the mouse pointer over the different keys. The bottom line shows you the corresponding PC/Terminal keys. For example, it you point on the “t “ key of the Terminal keyboard, you will see that the corresponding PC key is “T”.

Hot Keys

Hot keys are keys that you can press instead of choosing menu commands. These hot keys refer to your standard PC keyboard keys, before mapping to terminal keys has been performed. The following table lists the PowerTerm default hot keys.

Alt F4	Exit
Alt F6	open application
Alt F9	activate macro
Ctrl+Alt F9	start/stop recording macro
Alt F10	select screen
Alt F11	clear screen
Alt F12	reverse screen
Alt L	switch between English/Local
Scroll lock	hold screen
Pause	change the cursor shape
Ctrl Up Arrow	scroll up one line
Ctrl Down Arrow	scroll down one line
Ctrl Home	scroll to beginning of file
Ctrl End	scroll to end of file
Ctrl Page Up	scroll up one page
Ctrl Page Down	scroll down one page

PowerTerm Reference

General Topics

Text Selection, Copy and Paste

Status Indicators

Hot Keys

Menus

File

Edit

Terminal

Communication

Options

Macro

Script

Help

Help

Contents

About PowerTerm

Displays the contents page of the PowerTerm on-line help.

Displays PowerTerm copyright and version number information.

Text Selection, Copy and Paste

In the work area you can select text using the mouse.

Note: If the Automatic Copy option in the Edit menu is active (default), selecting text also copies the selection to the Clipboard (for details about the clipboard, consult your Windows documentation).

Selecting a word. Clicking a word selects the word. CTRL+ clicking a word selects the word and any punctuation marks or other symbols, up to the first space that follows them.

Selecting strings. Point to the first character you want to include in the selection. Drag the mouse to the last character you want to include in the selection and release the mouse button.

Selecting blocks. A block is any section of the work area.

- In VT: Point to one corner of the block, hold down CTRL and drag the mouse to the opposite corner of the block. Release both CTRL and the mouse button.
- In 3270 and 5250: Point to a line, hold down SHIFT and drag the mouse to the last line you want to include in the selection. Release both SHIFT and the mouse button.

Selecting full lines. Like block selection, full line selection differs according to the terminal type.

- In VT: Point to a line, hold down SHIFT and drag the mouse to the last line you want to include in the selection. Release both SHIFT and the mouse button.
- In 3270 and 5250: Point to one corner of the block, hold down CTRL and drag the mouse to the opposite corner of the block. Release both CTRL and the mouse button.

Selecting the entire screen. Select the Edit | Select Screen (ALT+F10) command instead of using the mouse.

Copy. If the Automatic Copy option in the Edit menu is not active, use the Edit | Copy command to copy the marked text to the Clipboard.

Paste. Right-click (or select Edit | Paste) to send to the host the data stored in the clipboard. This operation is equivalent to actually typing the contents of the clipboard on the host screen.

Selecting a menu entry. Double-clicking a word (except for 3270) sends the word to the host followed by an ENTER signal. You will use this feature to select a menu entry. For instance, if the emulation screen displays the menu of an application residing on the host, just click a menu entry to activate the program the menu entry represents.

In 3270 any double-click on the screen is equivalent to touching the screen with a light pen.

Status Indicators

The bottom line of the PowerTerm desktop contains a series of status indicators. A description of each follows.

Emulation type: The leftmost indicator shows the current terminal type selected under the Emulation tab (Terminal | Setup).

Row and column indicator: Indicates the current position of the cursor.

Caps: Indicates the status of Caps Lock.

Hold: Indicates whether the emulator is in Hold Screen state.

On Line, Off Line, Printer, Auto Prt: When communication is established, the status indicator reads On Line.

When data is transmitted with a printing request on the slave printer, this indicator reads Printer. The color is the same as the previous color; for example, if the system was On Line when the printing request arrived, Printer will appear in red.

When the terminal is in automatic printing mode, the data is sent both to the screen and the printer, and the indicator reads Auto Prt.

File

New Terminal Setup
Open Terminal Setup
Save Terminal Setup
Save Terminal Setup As...
Print Screen
Print Setup
Start Auto Print/Stop Printing
Form Feed
Line Feed
New Terminal Window

Restores the factory default parameters (including colors).

If you changed terminal parameters since the last save, and before activating File | New Terminal Setup, PowerTerm displays a warning message requiring you to decide whether or not you want to save the latest changes. The message points to the terminal settings file currently loaded (PTDEF.PTS is the default).

You can do one of the following:

- Click Yes. PowerTerm saves the latest changes and then executes New Terminal Setup.
- Click No. PowerTerm cancels the latest changes and then executes New Terminal Setup.
- Click Cancel to cancel the operation.

Opens an existing terminal settings file. If you changed terminal settings before activating this command, the system asks the same question as in New Terminal Setup.

Saves the current terminal settings to the currently loaded terminal settings (.PTS) file.

Enables you to create a new terminal settings file.

1. When you select File | Save Terminal Setup As... PowerTerm displays the Save File As dialog box.
2. In the File Name text box type the name you want, with the .PTS extension (for example: **DG.PTS**).
3. Click the OK button.

Prints the contents of the work area, or the selected text.

Allows you to select a printer. This command opens the Print Setup dialog box. This dialog box contains and leads to a set of printing parameters. The parameters change according to the printer you selected. For details, consult your printer documentation.

Notice that the Default Printer parameter enables you to send the output to the default printer selected under Windows. The Specific Printer parameter allows you to select one of the installed printers. For details about installing a new printer, consult your Windows documentation.

The Start Auto print command starts to accumulate incoming data (while it is displayed on the screen). After selecting this command, the menu entry changes to Stop Printing,

The Stop Printing command prints all the data accumulated in the printing buffer of the slave printer or in the Auto print buffer. If data was buffered with a printing request and communication failed before the data was sent to the slave printer, select this command to print the accumulated information.

Executes a form feed on the printer.

Executes a line feed on the printer.

New Terminal Window

Opens a new instance of PowerTerm (another PowerTerm window opens). After opening a PowerTerm window, select the terminal and communication parameters you want, and start a session. You can conduct several sessions concurrently and access a session by switching windows.

Each session is identified by a letter (A to Z) which appears in the session window's title bar. The first session is (A), the second (B), etc. A session is assigned the first available identifier. For example, if A, C and D are open, the next session is assigned B.

You can switch between sessions by pressing SHIFT+CTRL+session identifier (e.g. SHIFT+CTRL+C).

Edit

Select Screen
Clear Screen
Reverse Screen
Copy
Paste
Copy To File
Automatic Copy

Select Screen

Selects the contents of the entire work area. Use this command instead of selecting text with the mouse.

Clears the work area.

Displays a mirror image of the contents of the work area.

Writes the selected text to a file. If you don't select any text, the entire screen is written to the file.

After selecting this command, PowerTerm opens a dialog box where you specify the file name,

Automatic Copy

When the Automatic Copy option is active (default), selecting text also copies the selection to the Clipboard.

{button ,AL(`Selecting Text Items',0,`',`sec')} Related Topics

Terminal

The Terminal menu leads to the Setup command which allows you access to a large number of setup parameters. Other Terminal menu commands determine status.

Below the Terminal menu commands is a list of available languages, from which you can select a language for the PowerTerm interface. This is an optional feature.

Setup for VT and General Types

Setup for 3270 and 5270

Reset

On Line

Hold Screen

Setup for VT and General Types

The Setup command opens a dialog box divided into several tabs. The number of tabs depend on the terminal type you select under the Emulation tab. This topic reviews the parameters for VT Terminal Types and General Terminal Types.

All of the definitions made through Terminal | Setup can be stored in a .PTS file. Note the following:

- When you start PowerTerm, the default PTDEF.PTS file is automatically loaded. If you want, you can load a different .PTS file, using File | Open Terminal Setup.
- If you change any of the parameters accessed through a Terminal menu command, you can save them using File | Save Terminal Setup. If you don't save the changes, PowerTerm enables you to decide whether or not to save them when you request exit by pressing ALT+F4 or by double-clicking the Control-menu box. When this message appears, you can click OK to save and exit. The current settings are saved to the currently loaded .PTS file.
- You can save the current settings to a different .PTS file using the File | Save Terminal Setup As command.

VT and General Emulation Setup Parameters

Emulation

General for VT and general types

Display

Keyboard

Printer

Tabs

Colors

Preferences

Under the emulation tab PowerTerm displays the supported emulations. Click the emulation you require. The number of Setup tabs change according to the emulation you clicked.

General

This topic describes the parameters that appear when you click the General tab.

Terminal Id

Determines the Id returned by the emulation program to the host. Make sure you select an Id the host application recognizes.

NRC Set

Determines the communication and keyboard character set.

On Line

This parameter is equivalent to the On Line/Off Line button on the PowerTerm desktop.

Use 8-Bit Data Characters

If the communicated data is in 8-bit character format, check this parameter. For 7-bit characters, clear this parameter. If this parameter is cleared, the 8th bit is truncated.

If you receive 7-bit data, you can convert it to 8-bit data for printing on the slave printer.

Cursor Coupling

These parameter cause the cursor to remain visible during page scrolling.

Status Line

The None option displays an emulation screen without a status line.

Use Indicator to display the status line. Use Host Writable to display the status line sent by the host.

Display

This topic describes the parameters that appear when you click the Display tab in the Terminal Setup window.

Unscaled Screen

When this parameter is off (not checked), the characters appearing in the work area are scaled when you change the size of the desktop window. Check this parameter if you want to disable this feature.

Reverse Display Colors

Reverses the colors in the work area.

Autowrap Characters

If you check this box, word wrapping occurs at the end of a line and the cursor moves to the next line.

History Scroll Bar

Displays a vertical scrolling bar along the right edge of the PowerTerm desktop window. This enables you to scroll the window and see data previously brought to display. If the host transmits during scrolling, the display automatically scrolls back to its current position. Notice that PowerTerm begins to store history after you display the scroll bar.

Cursor Ruler

Use these parameters to display full-screen vertical or horizontal lines as cursor.

Cursor

The parameters in the Cursor group control the way the cursor is displayed. Experiment with each parameter.

Ctrl Characters

Click the Display box to actually display the control characters. The Interpret option displays normal text as affected by control characters.

Show Frame

Displays a window frame and causes the display to show fully on the emulation screen.

Dimensions

Determines the number of characters per displayed line, and the number of lines to be displayed in the work area. Characters are scaled according to the selected values. For example, if you select 132 Columns, PowerTerm displays 132 (small) characters across a single screen line.

Apart from the standard 80 and 132, you can also type a different value in the Columns box.

Scrolling

Determines the pace at which data is displayed in the work area, as it arrives.

If you select Jump, you should determine the Jump Scroll Speed. The higher the value the faster the scrolling. Select Unlimited to display data without delaying communication, or Page to scroll data by full screens.

The Smooth option is equivalent to a Jump Scroll Speed of 1.

Enable Soft Fonts

Enables you to work with VT soft fonts.

Keyboard

This topic describes the parameters that appear when you click the Keyboard tab in the Terminal Setup window.

Backspace Key Sends

Determines what the BACKSPACE key sends (a DELETE or an actual BACKSPACE).

Key Click

If you check this box, a click sound is issued when you press a key on the keyboard.

Use Emulator ALT Keys

This parameter is used to disable standard Windows ALT sequences like ALT+F4. If you check this parameter, a ALT sequence will perform the operation assigned to it by the emulated environment.

Use LK450 Keyboard

Check this box to transform your keyboard to Digital's LK450 keyboard.

Use Shift Lock

Check this parameter to simulate SHIFT LOCK. When this parameter is checked the entire keyboard moves to SHIFT LOCK status. For example, if you type "a", the keyboard issues "A".

Printer

This topic describes the parameters that appear when you click the Printer tab in the Terminal Setup window.

Print Device

This option allows you to select a printing output channel. The possibilities are:

Option	Sends the output
Print Manager	To the standard Print Manager of Windows, in text mode.
Device	To the DOS device you specify in the Device Name text box. For example, this can be a device such as prn, lpt1,com1. In the device name you can specify communication parameters. Example: COM1:9600,8
File	To a file. See next parameter.

Device

After choosing the File option in Print Device, specify the path and name of the file in the File Name parameter.

If a file of the same name exists, you can choose to add the new data to it, or to create a new file. This is done via the File Creation parameters. To add the data, click the Append option; to create a new file, click the Overwrite option.

Use Form Feed

Click this option to add a form feed (page eject) after each printing job.

Print Line Graphics As Text

Click this option to convert line graphics to text. It is designed to speed up printing on a slow dot-matrix printer.

Print Screen Data Conversion

You can determine to convert data to IBM or Digital standard. If you don't want to convert data, use the None option.

Slave Printer Data Conversion

Same as previous parameter for slave printer.

Slave Printer Job Delimiter

When printing in slave mode, the job delimiter character that you select here will divide the data into print jobs, instead of escape sequences arriving from the host application.

Tabs

This topic describes the parameters that appear when you click the Tabs tab.

Tabs parameters enable you to determine tab stops on the work area. Tabbed data received from the host will be laid out in the work area according to ruler settings you define here.

Set Every

Type a number and press ENTER. PowerTerm will set a tab stop every so many characters.

Alternatively, you can set tab stops manually by clicking anywhere you want within the ruler (Tab Stops) area.

Clear All

Click this button to clear all tab stops.

Colors

This topic describes the parameters that appear when you click the Colors tab.

Under the Colors tab you can define the color of displayed data. Color definitions are stored in the terminal parameters (.PTS) file.

Preview Box

The box above Select Attribute shows the result of your selections. Experiment with various attributes until you see the result you want, and then click OK.

Enable Underline

If data is transmitted with the underline attribute you can disable the underline by clearing this parameter. Check the parameter to enable underlined characters.

Enable Blink

If data is transmitted with the blink attribute you can disable blinking by clearing this parameter. Check the parameter to enable blinking.

Disable Host Colors

Check this parameter if you want to disable the host color definitions and to work with your own (PC) color scheme.

Select Attribute

Click the attribute for which you want to define foreground and background colors. Notice that the attributes change according to the emulation type you selected previously.

Text

Click the color that will apply to the text part (foreground) of the display.

Background

Click the color that will apply to the background of the text. Generally the attribute of the entire screen is Normal. The color for the Normal attribute determines the color of the entire work area.

Preferences

Following is a description of the parameters that appear when you click the Preferences tab in the Terminal Setup window:

Preferences are parameters that determine PowerTerm behavior and automate processes. They remain in force until you change them. For example, if you selected to connect automatically at PowerTerm startup, you will be always connected when you open PowerTerm, until you change this setting in Preferences.

On Terminal Setup File Open

These parameters tell PowerTerm what to do when you open the terminal setup file.

If you check	Then
Auto Connect	Connection is established immediately with the parameters saved in the terminal parameters file.
Show Connect Dialog Box	Connection is not established immediately. The Connect dialog opens, enabling you to select the connection parameters.
Do Not Connect	The PowerTerm window opens. You decide what to do.
Connect With	You will be connected with a set of parameters different from the one saved in the terminal parameters file. Select one of the sessions you saved in the Connect dialog box.
Login Command File	Execute a script after connect. Select the script file.

On Session Exit

These parameters determine what to do when to exit a session. Auto Reconnect re-establishes communication. Auto Exit PowerTerm closes PowerTerm altogether.

On PowerTerm Exit

These parameters determine what to do when you close PowerTerm.

If you check	Then
Save Terminal Setup	The new terminal parameters (if you changed them) are saved to the current terminal setup file.
Confirm Save	Terminal parameters are not saved automatically. PowerTerm displays a dialog box where you can decide whether or not to save.
Save Window Size & Position	Saves the size and position of the emulation window. The next time you open PowerTerm the same window appears.
Confirm Disconnect Session	If you close PowerTerm during a session, you will be requested to confirm disconnect.

Setup for 3270 and 5250

Most of the parameters for VT and general emulations apply to 3270 and 5250 as well. This topic describes the differences that appear when you select 3270 or 5250 emulation types.

In the General tab of the Terminal Setup window, the following parameters appear for 3270 and 5250:

Show Response Time

Check this parameter if you want to display the number of seconds that elapsed since data was sent to the host, and until the host responds.

Keyboard Type Ahead

Check this parameter if you want to be able to type data ahead (before the host responds). The Reset button empties the type ahead buffer.

HLLAPI Names

You can specify the short and long HLLAPI names.

Display

In Character Set specify the national character set.

In Status Line select the type of the status line that you want to use.

Alternate Size

If you check the Enable parameter, you override the terminal alternate size. Type the number of rows and columns you want.

General

In the General tab the following parameters appear for 3270 and 5250.

Show Response Time

Check this parameter if you want to display the number of seconds that elapsed since data was sent to the host, and until the host responds.

Keyboard Type Ahead

Check this parameter if you want to be able to type data ahead (before the host responds). The Reset button empties the type ahead buffer.

HLLAPI Names

You can specify the short and long HLLAPI names.

Display

In Character Set specify the national character set.

In Status Line select the type of the status line that you want to use.

Alternate Size

If you check the Enable parameter, you override the terminal alternate size. Type the number of rows and columns you want.

This command is equivalent to the VT Reset command. It restores the VT terminal defaults. This command does not apply to PowerTerm's exclusive terminal parameters (such as color).

When communication is established, this command is checked. Select the checked command to go off line. When the system is off line, all keyboard input is directed to the screen (it is not sent to the host). If you select the command again, PowerTerm resumes normal operations.

The on line/off line state is indicated in the status bar.

Stops communications and freezes the screen. To resume normal work select the command again.

Communication

The communication menu contains the commands needed to establish PC-host communication and to send/receive data.

Connect

Modify Connection

Disconnect

Reset Communication

File Transfer Setup

Receive File

Send File

Utilities

Connect

Selecting the Communication | Connect command displays the Connect dialog box in which you can determine the communication settings.

Following is a description of the parameters that can be defined in the Connect dialog box:

Session Type

Session types differ according to emulation. For example, under 3270 you will see a session type name TN3270 while under a VT you will see TELNET.

Click a session type. For each session type PowerTerm displays a set of session parameters on the right side of the dialog box (some types have identical parameters). A description each type follows.

Session Type	Description
TELNET	Uses the Telnet protocol over TCP/IP for network communication. For this type you should specify the host computer name or the IP address. You can also specify the TELNET port number (default 23). The WINSOCK.DLL file must be on the search path.
COM	Uses serial communication with the PC's COM ports. For this type you should define the port number, baud rate, parity, and stop bits. Optionally you can specify a phone (dial) number.
INT 14	Uses BIOS interrupt 14. For this type you should specify the port number, baud rate, parity, and stop bits.
BAPI	For TCP/IP connections with parameters similar to those of TELNET. Before you use this option make sure you installed on your PC the BAPI support software.
CTERM	Uses the Digital CTERM protocol for network communication with a remote or local VAX/VMS host via Digital PATHWORKS 32. For this type you should specify the host computer name.
LAT	Uses Digital Local Area Transport protocol for network communication with a VAX/VMS host via Digital PATHWORKS 32. For this type you should specify a Service name and optionally a Password if required.
TN3270	TELNET for 3270. Check the Use TN3270E Protocol box if you want to work with TELNET SNA extensions. You can also specify the LU name of the host (LU name or LU pool).
MS SNA Server	For connection via Microsoft SNA Server. Specify the LU Name (or LU pool).
NWSAA (IPX)	For connection via IPX to Novel Network for SAA. Note that Service Name is the same as Profile. You must select an LU Category.

NWSAA (TCP/IP)	Same a previous for TCP/IP connection.
TN5250	TELNET for 5250.
APPC	In System Name and Device Name specify the appropriate AS/400 names. Check Auto Signon if you want to skip the sign on stage.

Login Command File

In this parameter you can specify the name of a script to be run after communication is established.

To specify the name, type either the script name in the text box or:

1. Click the Browse button. PowerTerm displays the Browse Login Script dialog box.

   The Browse Login Script dialog box displays the script files located in the PowerTerm directory. By default it lists the files that carry the .PSL extension which is the factory default. However, you can name your scripts anyway you want. If your script files carry a different extension, you can list them by typing the extension (preceded by an asterisk and a dot) in the File Name text box and pressing ENTER.

2. Double-click the script file name.

Terminal Setup File

In this parameter you can specify the name of the terminal setup file to be opened after communication is established.

To specify the name, either type it in the text box or browse in the same way as in Login Command file.

Session List

If you saved session settings using the Save As... button in the Connect dialog box, you will see the session names in this list. You can select a session by double-clicking its name. This establishes communication with the host. Note that, as you select a session, its parameters appear in the dialog box.

Connect

Connects to the host observing the displayed session parameters.

Save As...

Use the Save As... button to save the current session settings. After saving a set of communication parameters, you can use it to start a communication session via Session List.

1. Click the Save As... button. PowerTerm displays the Save Session dialog box.
2. In the Session Name text box PowerTerm displays a default session name. You can type in a different name, if required.
3. Click the OK button to save the session settings under the name that appears in the Session Name box.

Modify

Updates the selected session in the Session List, with the data entered in the upper section of the dialog box.

Delete

Deletes a session setting.

1. Click the name you want to delete.
2. Click the Delete button.
3. Confirm the operation in the Delete Session dialog box.

This command applies to the COM session type only. It opens the Modify Connection dialog box. This dialog box similar to the Connect dialog box. The title bar of the box displays the session type. You can change the session parameters (but not Session Type) and the Login Command File.

This command enables you to close the communication session from the PC.

This command is relevant for COM type communication. It resets the communication port.

This command enables you to inform PowerTerm about the type of data on the PC and on the host. Determine the data types before you perform any file transfers.

Open the drop-down lists and select the appropriate data types.

Receive File

This command serves to send a file from the host to the PC. When you select this command, PowerTerm opens a dialog box with tabs that correspond to the transmission options. Each option receives files using the protocol the option represents.

To receive a file with a protocol other than ASCII follow these steps.

1. Establish communication.
2. In the work area type the command the host uses to receive a file.
3. Select Communication | Receive File and click one of the tabs, except ASCII.
4. Select a directory, if required.
5. Type a file name if you want to receive the data into a file that carries a different name. Leave *.* (or empty) if you want to receive the data into a file that carries the same name as the name used by the host.
6. Click the OK button. The received file will be saved on the PC disk under the name you specify (in the directory you selected). The PC starts to receive the file.

To receive an ASCII file follow these steps:

1. Establish communication.
2. In the work area type in the command that displays the file (examples: for UNIX - cat filename; for VAX type filename). Do not press ENTER.
2. Select Communication | Receive File and click the ASCII tab.
3. In File Name enter the name of the PC file that will store the data, and click OK.
4. Press ENTER to activate the command.
5. At the end of the capturing, select the Communication | Stop Receiving ASCII File command.

Send File

This command serves to send a file from the PC to the host. When you select this command, PowerTerm opens a sub-menu that lists the transmission options. Each options sends files using the protocol the option represents.

All file transmission options lead to the same dialog box, in which you select the file to be sent.

To send a file other than ASCII follow these steps:

1. Establish communication.
2. In the work area type the host's file reception command.
3. Select Communication | Send File.
4. Click a tab other than ASCII.
5. Select the file to be sent and click the OK button. The PC starts to send the file.

To send an ASCII file follow these steps:

1. Sending a file is the same as actually typing the contents of the ASCII file from the keyboard. Activate the command that will accept the data on the host (examples: .for UNIX -cat > filename, or entering an editor and moving to data entry state within the editor).
2. Select Communication | Send File, and click the ASCII tab.
3. Enter the PC file name and click OK.

Utilities

The Utilities command leads to a sub-menu. The commands included in the Utilities sub-menu are relevant for COM-type sessions.

Dial

Break

Set/Clear DTR/RTS

This command leads to a dialog box where you can specify a phone number. As you click OK, PowerTerm begins dialing.

This command sends a break (equivalent to CTRL+BREAK).

The four DTR/RTS commands send/clear “data terminal ready” and “request to send” signals.

Options

Keyboard Map

Powerpad Setup

Start /Stop Trace

Input File

Hide Menu

Hide Buttons, Status Bar, Toolbar

Show Powerpad

Keyboard Map

This command displays the Keyboard Mapping dialog box where you can map PC keys to host keys. Keyboard mapping definitions are stored in the terminal parameters (PTK) file.

To map a keyboard key, perform one of the following:

- To map a PC key to a terminal key, drag a key from the upper terminal keyboard to a PC key on the lower keyboard.
- You can click on the Shift and/or Control keys in the upper terminal keyboard to display additional key functions that can only be viewed by first selecting either of these keys. You can then drag the required key from the upper terminal keyboard to a key on the lower PC keyboard.
- To cancel a definition, drag the PC key definition to the wastebasket. This restores the default function of the PC key.
- To replace a PC key with another PC key, drag the key you want to the key that it will replace. This cancels the function of the original key. To cancel the replacement, drag the replaced key back to its original position.
- The Defaults button restores the default mapping. In the default mapping each PC keyboard key is mapped to its counterpart in the host keyboard (for example, Num Lock is mapped to PF1 in VT terminal emulation).

Click Options | PowerPad Setup to set the dimensions of the PowerPad.

Select the number of rows and columns for the PowerPad. Display the PowerPad using the Options | Show PowerPad command, and experiment with various row and column values.

Start /Stop Trace

Start Trace stores received data in the TRACE.LOG and CAPTURE.LOG files. These files are located in the working directory.

CAPTURE.LOG stores raw data, as received from the host. TRACE.LOG stores formatted data with readable escape sequences.

After activating Start Trace, the menu command changes to Stop Trace. Select it to stop storing the data.

Inputs the contents of the CAPTURE.LOG file to PowerTerm, as if it was received from the host.

Removes the menu bar altogether. To restore the menu, click the Control-menu box and select Restore Menu.

These commands alternately hide and display the button bar, the status bar and the tool bar.

Displays the PowerPad. After displaying the PowerPad the command changes to Hide PowerPad.

Macro

Via the macro menu you can record keystrokes and transmit them.

Start/Stop Recording

Activate Macro

Select Start Recording (CTRL+ALT+F9) to begin a macro recording session. PowerTerm activates the macro display area - the right section of the desktop's bottom line.

As you type, the characters are shown in the macro display area. When the area is full scrolling occurs.

Select Stop Recording (CTRL+ALT+F9) to stop recording your keyboard input.

Runs the recorded macro. Press ALT+F9 to send the recorded characters to the host.

Script

The Script menu leads to commands designed to write scripts in PowerTerm Script Language (PSL) and to run them.

Run Script

Edit Script

Script Command

Start, Pause, Stop Script Recording

Run Script

Executes a script. As you select this command, PowerTerm displays the Run Script dialog box which is identical to other file selection dialogs throughout the system. By default, the Run Script dialog lists the files that carry the .PSL extension in the PowerTerm directory (however, you can assign any name and extension that comply with DOS file naming conventions).

To run a script, just double-click the script file you want.

{button ,AL(`Running a Script at Startup',0,`',`sec')} Related Topics

Running a Script at Startup

You can run a script as you start PowerTerm. To achieve this, you should add the script name to the PowerTerm shortcut or command line. Follow these steps:

To create a shortcut to the script name:

1. Click the Start button and select Settings | Taskbar.
2. In the Taskbar properties dialog box click the Advanced button. The Exploring Programs window opens.
3. In All Folders double-click Programs.
4. Double-click PowerTerm.
5. In the right section of the Exploring… screen right-click PowerTerm, and select Properties. The PowerTerm Properties dialog opens.
6. Click the Shortcut tab.
7. The Target box shows the PowerTerm command line. Press END to move the insertion point to the end of the command screen, and type a space.
8. Type the name of the script file. For example, if TELNET.PSL is the script file, the Command Line string should look as follows:

   C:\PTW\PTW.EXE TELNET.PSL

   You can also execute the script with parameters. Example:

   C:\PTW\PTW.EXE TELNET.PSL HOSTA
9. Click the OK button.

When you start PowerTerm, the script will be automatically executed.

PowerTerm provides several standard scripts designed for frequent tasks. The following table lists the standard scripts and their parameters.

Script	**Parameters**	**Parameter Values**
COMM.PSL	Port number	1 - 32
	Baud rate	All
	Protocol type	none, xonxoff, hardware.
		All parameters must be typed in lowercase letters.
TELNET.PSL	Host name	Specify the name of the host.
LAT.PSL	Service name	Specify the name of the service.
CTERM.PSL	Node name	Specify the name of the CTERM node.

Edit Script

Enables you to edit an existing script or to create a new script. PowerTerm uses Notepad (for Windows) as a script text editor. For details about using Notepad consult your Windows documentation.

This command leads you to the Edit Script dialog box where you can do one of the following:

- Select a script file for further editing.
- Create a new script file.

To select a file, double-click the name you want in the Files list. This activates Notepad and displays the script.

To create a new file:

1. Type a name in the File Name box. You can type any name and extension that comply with DOS file naming conventions. However, the .PSL extension is handier since PowerTerm defaults to this extension to list the existing files.

2. Notepad displays a message box asking you whether you want to create a new file. Click Yes. The Notepad window is displayed.

This command allows you to execute individual script commands.

Start, Pause, Stop Script Recording

The Start Script Recording command enables you to write a script automatically. After requesting start, the manual operations you perform in the emulation screen are recorded into a script file, until you chose the Stop Script command.

Follow these steps:

1. Select Script | Start Script Recording. The Record Script dialog box opens.
2. In File name, type the name of the script file with the psl extension. The manual operations you will perform will be stored in this file. Click OK.
3. Perform the manual operations you want to record. If you want to omit some operations Select Script | Pause Script Recording, and then select again Start Script Recording in order to resume script recording.
4. When you have finished performing the operations you want to store, select Script | Stop Script Recording. Now you have a script file that you can run anytime to repeat the actions you performed while script recording was on.

PowerTerm Script Language

The PowerTerm Script Language is intended for users who have programming skills. The language is based on C and UNIX shell.

lsort
menu
message
open
open-setup-file
proc
puts
pwd
read
regexp
regsub
rename
return
run
scan
screen
screen-rect
seek
send
session
set
split
string
switch
tell
time
unset
uplevel
upvar
wait
while
window

PSL Syntax

A command consists of one or more fields separated by spaces or tabs. The first field is the name of a command, which may be either a built-in command, or a procedure consisting of a sequence of PSL commands. Newline characters are used as command separators, and semi-colons may be used to separate commands on the same line. Each PSL command returns a string result, or an empty string.

There are four additional syntactic constructs in PSL.

Curly braces are used to group complex arguments; they act as nestable quote characters. If the first character of an argument is an open brace, then the argument is not terminated by white space. Instead, it is terminated by the matching close brace. The argument passed to the command consists of everything between the braces, with the enclosing braces stripped off.

For example:

```
host = {vms unix {aix hp sun} aos}
```

Variable host will receive one arguments: "vms unix {aix hp sun} aos".

This particular command will set the variable host to the specified string.

If an argument is enclosed in braces, then none of the other substitutions described below is made on the argument. One of the most common uses of braces is to specify a PSL subprogram as an argument to a PSL command.

The second syntactic construct in PSL is square brackets, which are used to invoke command substitution. If an open bracket appears in an argument, then everything from the open bracket up to the matching close bracket is treated as a command and executed recursively by PSL. The result of the command is then substituted into the argument in place of the bracketed string.

For example:

```
msg = [format {Data is %s bytes long} 99]
```

The format command does print-like formatting (from the C language) and returns the string "Data is 99 bytes long", which is then assigned to variable msg.

The third syntactic construct is the dollar sign, which is used for variable substitution. If it appears in an argument then the following characters are treated as a variable name; the contents of the variable are substituted into the argument in place of the dollar sign and name.

For example:

```
num = 99
msg = [format {Data is %s bytes long} $num]
```

The result is the same as the single command in the previous paragraph.

The last syntactic construct is the backslash character, which may be used to insert special characters into arguments, such as curly braces or non-printing characters.

PSL Data Types

There is only one type of data in PSL: strings. All commands, arguments to commands, results returned by commands, and variable values are ASCII strings.

Although everything in PSL is a string, many commands expect their string arguments to have particular formats. There are three particularly common formats for strings: lists, expressions, and commands. A list is just a string containing one or more fields separated by white space, similar to a command. Curly braces may be used to enclose complex list elements; these complex list elements are often lists in their own right.

For example:

```
{vms unix {aix hp sun} aos}
```

A list with four elements, the third of which is a list with three elements. PSL provides commands for a number of list-manipulation operations, such as creating lists, extracting elements, and computing list lengths.

The second common form for a string is a numeric expression. PSL expressions have the same operators and precedence as expressions in the C language.

The **expr** PSL command evaluates a string as an expression and returns the result (as a string, of course).

For example:

```
expr {($x < $y) || ($z != 0)}
```

Returns "1'' if the numeric value of variable x is less than that of variable y, or if variable z is not zero; otherwise it returns "0''. Several other commands, such as **if** and **for**, expect one or more of their arguments to be expressions.

The third common interpretation of strings is as commands (or sequences of commands). Arguments of this form are used in PSL commands that implement control structures.

For example:

```
if {$x < $y} {
        swap = $x
        x = $y
        y = $swap
}
```

The **if** command receives two arguments here, each of which is delimited by curly braces. **if** is a built-in command that evaluates its first argument as an expression; if the result is non-zero, if executes its second argument as a PSL command. This particular command swaps the values of the variables x and y if x is less than y.

PSL also allows users to define command procedures written in the PSL language. The **proc** built-in command is used to create a PSL procedures (PSLproc).

For example:

```
proc factorial x {
        if {$x == 1} {return 1}
        return [expr {$x * [factorial [expr $x - 1]]}]
}
```

This PSL command defines a recursive factorial procedure:

The **proc** command takes three arguments: a name for the new PSLproc, a list of variable names (in this case the list has only a single element, x), and a PSL command that comprises the body of the PSLproc. Once this **proc** command has been executed, factorial may be invoked just like any other PSL command.

For example:

```
factorial 4
```

returns the string "24''.

In addition to the commands already mentioned, PSL provides commands for manipulating strings (comparison, matching, and printf/scanf-like C language operations), commands for manipulating files and file names. The built-in PSL commands provide a simple but complete programming language.

Variable Assignment

Assignments to variables follow this syntax:

```
varName = value
varName(index) = value
```

The assignment sets the value of varName to a value, creating a new variable if one doesn't already exist. If varName contains an open parenthesis and ends with a close parenthesis, then it refers to an array element: the characters before the first open parenthesis are the name of the array, and the characters between the parentheses are the index within the array. The index may take any value. Otherwise varName refers to a scalar variable.

If no procedure is active, then varName refers to a global variable. If a procedure is active, then varName refers to a parameter or local variable of the procedure unless the global command has been invoked to declare varName to be global.

Variable assignment returns to its value.

The following line assign the value "green" to variable host.

```
host = "green"
```

This line assign the value "vms" to item vax of array name.

```
name(vax) = "vms"
```

Activating Scripts From the Host

A host application may activate a script file or script commands via special escape sequences.

Escape Sequences for VT

Activating a script file by the name Script-Name:

ESCP$s**Script-Name**ESC \

Example: activating the message.psl script

ESCP$smessage.pslESC\

Note: ESC is the ASCII 27 code.

Activating script commands Script-Commands:

ESCP$t**Script-Commands**ESC \

Example: activating the "message testing ; send end" commands

ESCP$tmessage testing ; send endESC\

Escape Sequences for DG

Activating a script file by the name Script-Name:

ESCWs**Script-Name**000

Note : ESC is the ASCII 30 code.

000 is the ASCII 0 code.

Activating script commands Script-Commands:

ESCWtScript-Commands000

append

Description

Append to a variable

Syntax

append varName value [value value ...]

Notes

Appends all of the value arguments to the current value of variable varName. If varName doesn't exist, it is given a value equal to the concatenation of all the value arguments.

This command provides an efficient way to build up long variables incrementally.

Example

The following line appends variables y and z to variable x.

```
append x $y $z
```

It is much more efficient than

```
x = $x$y$z
```

if $x is long.

array

Description

Manipulate array variables.

Syntax

array option arrayName [arg arg ...]

Notes

This command performs one of several operations on the variable given by arrayName.

ArrayName must be the name of an existing array variable.

The option argument determines what action is carried out by the command.

A description of each valid option (which may be abbreviated) follows:

array anymore arrayName searchId

Returns 1 if there are any more elements left to be processed in an array search, 0 if all elements have already been returned.

SearchId indicates which search on arrayName to check, and must have been the return value from a previous invocation of **array startsearch**.

This option is particularly useful if an array has an element with an empty name, since the return value from **array nextelement** won't indicate whether the search has been completed.

array donesearch arrayName searchId

This command terminates an array search and destroys all the state associated with that search. SearchId indicates which search on arrayName to destroy, and must have been the return value from a previous invocation of **array startsearch**. Returns an empty string.

array names arrayName

Returns a list containing the names of all of the elements in the array.

If there are no elements in the array then an empty string is returned.

array nextelement arrayName searchId

Returns the name of the next element in arrayName, or an empty string if all elements of arrayName have already been returned in this search. The searchId argument identifies the search, and must have been the return value of an **array startsearch** command.

Warning: if elements are added to or deleted from the array, then all searches are automatically terminated just as if **array donesearch** had been invoked; this will cause **array nextelement** operations to fail for those searches.

array size arrayName

Returns a decimal string giving the number of elements in the array.

array startsearch arrayName

This command initializes an element-by-element search through the array given by arrayName, such that invocations of the **array nextelement** command will return the names of the individual elements in the array.

When the search has been completed, the **array donesearch** command should be invoked.

The return value is a search identifier that must be used in **array nextelement** and **array donesearch** commands; it allows multiple searches to be underway simultaneously for the same array.

break

Description

Abort looping command.

Syntax

break

Notes

This command may be invoked only inside the body of a looping command such as **for**, **foreach** or **while**.

Example

break terminates the **while** loop when variable x equals 5.

```
x = 0
while {$x < 10} {
        incr x
        if {$x == 5}
                break
        .
        .
        .
}
```

cd

Description

Change working directory.

Syntax

cd dirName

Notes

Changes the current working directory to dirName.

Returns

This line returns an empty string.

Example

Changes working directory to pterm.

```
cd pterm
```

close

Description

Close an open file.

Syntax

close fileId

Notes

Closes the file given by fileId.

fileId must be the return value from a previous invocation of the **open** command; after this command, it should not be used anymore.

Returns

The normal result of this command is an empty string, but errors are returned if there are problems in closing the file.

Example

opens file "sales.dat", reads 10 bytes, closes it and displays the data in a message box.

```
fileId = [open sales.dat]

data = [read $fileId 10]

close $fileId

message $data
```

concat

Description

Join lists together.

Syntax

concat arg [arg ...]

Notes

This command treats each argument as a list and concatenates them into a single list.

It also eliminates leading and trailing spaces in the arg's and adds a single separator space between arg's.

It permits any number of arguments.

Returns

Returns a single list with all elements.

Example

This command returns {a b c d e f {g h}}.

```
concat a b {c d e} {f {g h}}
```

continue

Description

Skip to the next iteration of a loop.

Syntax

continue

Notes

This command may be invoked only inside the body of a looping command such as **for**, **foreach** or **while**.

It signals the innermost containing loop command to skip the remainder of the loop's body but to continue with the next iteration of the loop.

Example

Skips over the **while** loop commands when variable x is less then 5.

```
x = 0
while {$x < 10} {
        incr x
        if {$x < 5}
                continue
        .
        .
        .
}
```

cursor

Description

Moves to cursor to a new position.

Syntax

cursor row col

Notes

The cursor command changes to cursor position, but does not send any indication to the host.

Returns

Returns an empty string.

Example

cursor 4 34

dde

Description

Uses standard MS-Windows DDE mechanism to communicate with other Windows applications.

Syntax

dde option [args...]

Notes

DDE permits applications to communicate with each other. A DDE connection between two applications requires that one will be a server and the other a client.

The DDE server application waits for requests from DDE clients, and allows them to supply it with information or receive information.

For example: A spread sheet DDE server will let clients get data from cells and put data into cells of a file.

PowerTerm can be a DDE client application or a DDE server.

As a DDE server PowerTerm Uses the server name PTW with topic PSL. Any application can request it to execute commands and return the related return data.

A client application can access PowerTerm with the DDE EXECUTE command or the DDE REQUEST command, and an item which is any valid PSL commands separated with ";".

The single DDE server PSL command is:

dde return **value**

After a DDE REQUEST commands are executed the, PowerTerm DDE server sends the value from the last **dde return** command executed in this request. If no dde return command executed it returns an empty answer.

Examples for PowerTerm as a DDE server might be:

- Sending information to the host.
- Reading information from the emulation screen.

As a DDE client performs one of several dde operations, depending on option. The legal options are:

dde initiate applicationName topicName

Connects to the applicationName DDE server with topicName. Returns a conversation id for use with successive dde commands.

dde execute convId command

Executes a server command. Return an empty string.

dde request convId item

Returns an item from the server.

dde poke convId item value

Changes an item of the server to the new value. Returns an empty string.

dde terminate convId

Terminates a DDE conversation with the server.

Returns

Returns a value according to the option..

Example

1. Initiates a DDE conversation with Excel for a file products.xls. Reads 3 numbers from the emulation screen. Puts them in three cells and gets a calculated result from a forth cell. Sends the result to the host computer.

conv = [**dde initiate** EXCEL PRODUCTS.XLS]

dde poke $conv R1C1 [**screen-rect** 1 1 10]

dde poke $conv R2C1 [**screen-rect** 2 1 10]

dde poke $conv R3C1 [**screen-rect** 3 1 10]

result = [**dde request** $conv R5C1]

dde terminate $conv

send $result

Note: R5C1 indicates row 5 column 1.

2. Initiate a DDE conversation with an other PowerTerm with is connected to another computer. Reads information from the screen (of the other host) and sends it to its own host.

conv = [**dde initiate** PTW PSL]

data = [**dde request** $conv {

dde return [**screen** 10 1 15 80]}]

dde terminate $conv

send $data

Note: The command {**dde return** [**screen** 10 1 15 80]} is executed in the PowerTerm DDE server.

display

Description

Displays a string on the current cursor position.

Syntax

display string

Notes

The **display** command displays the string on the screen, but does not send it to the host.

Returns

Returns an empty string.

Example

display "Hit ENTER to continue"

eof

Description

Check for end-of-file condition on open file.

Syntax

eof fileId

Notes

Returns 1 if an end-of-file condition has occurred on fileId, 0 otherwise.

fileId must have been the return value from a previous call to **open**.

Example

Opens file "input.dat" for reading and file "output.dat" for writing. While not end of input file reads a line and writes it to the output file. Closes both files.

```
inFile = [open input.dat]
outFile = [open output.dat w]
gets $inFile data
while {! [eof $inFile]} {
        puts $outFile $data
        gets $inFile data
}
close $inFile
close $outFile
```

eval

Description

Evaluate a PSL script.

Syntax

eval arg [arg ...]

Notes

eval takes one or more arguments, which together comprise a PSL script containing one or more commands.

eval concatenates all its arguments in the same fashion as the **concat** command, passes the concatenated string to the PSL recursively.

Returns

Returns the result of the evaluation (or any error generated by it).

Example

assigns command variable with the **expr** command. executes the command and displays its output.

```
command = "expr 3 * 8"
result = [eval $command]
message $result
```

exec

Description

Invoke a program

Syntax

exec Program

Notes

This command executes a program. The Program variable may contain parameters.

Example

Activates the Notepad program with parameter pt.psl.

exec "notepad pt.psl"

expr

Description

Evaluate an expression.

Syntax

expr arg [arg arg ...]

Notes

Concatenates arg's (adding separator spaces between them), evaluates the result as a PSL expression, and returns the value.

The operators permitted in PSL expressions are a subset of the operators permitted in the C language expressions, and they have the same meaning and precedence as the corresponding C language operators.

Expressions almost always yield numeric results (integer or floating-point values).

PSL expressions support non-numeric operands and string comparisons.

For example the command "**expr** 8.2 + 6" evaluates to 14.2.

Operands

A PSL expression consists of a combination of operands, operators, and parentheses.

White space may be used between the operands and operators and parentheses; it is ignored by the expression processor.

Where possible, operands are interpreted as integer values.

Integer values may be specified in decimal (the normal case), in octal (if the first character of the operand is 0, or in hexadecimal (if the first two characters of the operand are 0x).

If an operand does not have one of the integer formats, then it is treated as a floating-point number if that is possible. Floating-point numbers may be specified in any of the ways accepted by an ANSI-compliant C language compiler (except that the f, F, l, and L suffixes will not be permitted in most installations). For example, all of the following are valid floating-point numbers: 2.1, 3., 6e4, 7.91e+16.

If no numeric interpretation is possible, then an operand is left as a string (and only a limited set of operators may be applied to it).

Operands may be specified in any of the following ways:

- As an numeric value, either integer or floating-point.
- As a PSL variable, using standard $ notation.
- The variable's value will be used as the operand.
- As a string enclosed in double-quotes.

The expression parser will perform backslash, variable, and command substitutions on the information between the quotes, and use the resulting value as the operand.

As a string enclosed in braces.

The characters between the open brace and matching close brace will be used as the operand without any substitutions.

As a PSL command enclosed in brackets.

The command will be executed and its result will be used as the operand.

As a mathematical function whose arguments have any of the above forms for operands, such as sin($x). See below for a list of defined functions.

Where substitutions occur above (e.g. inside quoted strings), they are performed by the expression processor.

However, an additional layer of substitution may already have been performed by the command parser before the expression processor was called.

As discussed below, it is usually best to enclose expressions in braces to prevent the command parser from performing substitutions on the contents.

For some examples of simple expressions, suppose the variable x has the value 3 and the variable y has the value 6.

Then the command on the left side of each of the lines below will produce the value on the right side of the line:

```
expr 3.1 + $x                                   6.1
expr 2 + "$x.$y"                            5.6
expr 4 * [llength "6 2"]                8
expr {word one} < "word $x"     0
```

Operators

The valid operators are listed below, grouped in decreasing order of precedence:

"-" "~" "!"

Unary minus, bit-wise NOT, logical NOT. None of these operands may be applied to string operands, and bit-wise NOT may be applied only to integers.

"*" "/" "%"

Multiply, divide, remainder. None of these operands may be applied to string operands, and remainder may be applied only to integers.

The remainder will always have the same sign as the divisor and an absolute value smaller than the divisor.

"+" "-"

Add and subtract. Valid for any numeric operands.

"<<" ">>"

Left and right shift. Valid for integer operands only.

"<" ">" "<=" ">="

Boolean less, greater, less than or equal, and greater than or equal.

Each operator produces 1 if the condition is true, 0 otherwise.

These operators may be applied to strings as well as numeric operands, in which case string comparison is used.

"==" "!="

Boolean equal and not equal. Each operator produces a zero/one result.

Valid for all operand types.

"&"

Bit-wise AND. Valid for integer operands only.

"^"

Bit-wise exclusive OR. Valid for integer operands only.

"|"

Bit-wise OR. Valid for integer operands only.

"&&"

Logical AND. Produces a 1 result if both operands are non-zero, 0 otherwise.

Valid for numeric operands only (integers or floating-point).

"||"

Logical OR. Produces a 0 result if both operands are zero, 1 otherwise.

Valid for numeric operands only (integers or floating-point).

"x ? y : z"

If-then-else, as in the C language. If x evaluates to non-zero, then the result is the value of y. Otherwise the result is the value of z.

The x operand must have a numeric value.

See the C language manual for more details on the results produced by each operator.

All of the binary operators group left-to-right within the same precedence level.

For example, the command

```
expr 4 * 2 < 7
```

returns 0.

The &&, ||, and ?: operators have "lazy evaluation", just as in C, which means that operands are not evaluated if they are not needed to determine the outcome.

For example, in the command

```
expr {$z ? [x] : [y]}
```

only one of [x] or [y] will actually be evaluated, depending on the value of $z.

Note, however, that this is only true if the entire expression is enclosed in braces; otherwise PSL will evaluate both [x] and [y] before invoking the **expr** command.

Mathematical functions

PSL supports the following mathematical functions in expressions:

acos	cos	hypot	sinh
asin	cosh	log	sqrt
atan	exp	log10	tan
atan2	floor	pow	tanh
ceil	fmod	sin	

Each of these functions invokes the C language math library function of the same name.

Conversion Functions

PSL also implements functions for conversion between integers and floating-point numbers. A description of these functions follows.

abs(arg)

Returns the absolute value of arg. Arg may be either integer or floating-point, and the result is returned in the same form.

double(arg)

If arg is a floating value, returns arg, otherwise converts arg to floating and returns the converted value.

int(arg)

If arg is an integer value, returns arg, otherwise converts arg to integer by truncation and returns the converted value.

round(arg)

If arg is an integer value, returns arg, otherwise converts arg to integer by rounding and returns the converted value.

Types, Conversion and Precision

Conversion among internal representations for integer, floating-point, and string operands is done automatically as needed.

For arithmetic computations, integers are used until some floating-point number is introduced, after which floating-point is used.

For example:

```
expr 5 / 4
```

returns 1, while

```
expr 5 / 4.0
```

```
expr 5 / ( [string length "abcd"] + 0.0 )
```

both return 1.25.

Floating-point values are always returned with a "." or an "e" so that they will not look like integer values.

For example:

```
expr 20.0/5.0
```

returns "4.0", not "4".

String Operations

String values may be used as operands of the comparison operators, although the expression evaluator tries to do comparisons as integer or floating-point when it can.

If one of the operands of a comparison is a string and the other has a numeric value, the numeric operand is converted back to a string.

For example, the commands

```
expr "0x03" > "2"

expr "0y" < "0x12"
```

both return 1. The first comparison is done using integer comparison, and the second is done using string comparison after the second operand is converted back to the string "18".

file

Description

Manipulate file names and attributes

Syntax

file option name [arg arg ...]

Notes

This command provides several operations on a file's name or attributes. Name is the name of a file.

Option indicates what to do with the file name. Any unique abbreviation for option is acceptable. A description of the valid options follows.

file atime name

Returns a decimal string giving the time at which file name was last accessed. The time is measured in the standard POSIX fashion as seconds from a fixed starting time (often January 1, 1970).

If the file doesn't exist or its access time cannot be queried then an error is generated.

file dirname name

Returns all of the characters in name up to but not including the last slash character. If there are no slashes in name then returns ".". If the last slash in name is its first character, then return "\".

file exists name

Returns 1 if file name exists, 0 otherwise.

file extension name

Returns all of the characters in name after and including the last dot in name. If there is no dot in name then returns the empty string.

file isdirectory name

Returns 1 if file name is a directory, 0 otherwise.

file isfile name

Returns 1 if file name is a regular file, 0 otherwise.

file mtime name

Returns a decimal string giving the time at which file name was last modified. The time is measured in the standard POSIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file doesn't exist or its modified time cannot be queried then an error is generated.

file rootname name

Returns all of the characters in name up to but not including the last "." character in the name. If name doesn't contain a dot, then returns name.

file size name

Returns a decimal string giving the size of file name in bytes. If the file doesn't exist or its size cannot be queried then an error is generated.

file stat name varName

Invokes the stat system call on name, and uses the variable given by varName to hold information returned from the system call. VarName is treated as an array variable, and the following elements of that variable are set: atime, ctime, dev, mode, mtime, size, type.

Each element except type is a decimal string with the value of the corresponding field from the stat return structure, see the manual entry for stat for details on the meanings of the values.

The type element gives the type of the file in the same form returned by the command file type. This command returns an empty string.

file tail name

Returns all of the characters in name after the last backslash. If name contains no backslashes then returns name.

file type name

Returns a string giving the type of file name, which will be one of file or directory.

flush

Description

Flush buffered output for a file

Syntax

flush file

Notes

Flushes any output that has been buffered for file. file must have been the return value from a previous call to open. it must refer to a file that was opened for writing.

Returns

The command returns an empty string.

Example

Opens file "sales.dat" for writing. Writes 10 lines, flushes data to file and than closes the file.

```
fileId = [open sales.dat w]
for {i = 0} {$i < 10} {incr i} {puts $fileId "Line no $i"}
flush $fileId
close $fileId
```

for

Description

"For'' loop.

Syntax

for start test next body

Notes

for is a looping command, similar in structure to the C language for statement. The start, next, and body arguments must be PSL command strings, and test is an expression string.

The **for** command first invokes PSL to execute start. Then it repeatedly evaluates test as an expression; if the result is non-zero it invokes PSL on body, then invokes PSL on next, then repeats the loop. The command terminates when test evaluates to 0.

If a **continue** command is invoked within body then any remaining commands in the current execution of body are skipped, processing continues by invoking PSL on next, then evaluating test, and so on.

If a **break** command is invoked within body or next, then the **for** command will return immediately.

The operation of **break** and **continue** are similar to the corresponding statements in the C language.

Returns

The command returns an empty string.

Example

Executes 10 times the message command.

```
for {i = 1} {i <= 10} {i = i + 1} {message "i = $i"}
```

foreach

Description

Iterate over all elements in a list.

Syntax

foreach varname list body

Notes

In this command varname is the name of a variable, list is a list of values to assign to varname, and body is a PSL script.

For each element of list (in order from left to right), **foreach** assigns the contents of the field to varname as if the index command had been used to extract the field, then calls PSL to execute body. The **break** and **continue** statements may be invoked inside body, with the same effect as in the **for** command.

Returns

The command returns an empty string.

Example

For each item in the list, a message is displayed.

```
foreach var {Item1 Item2 Item3} {message "Item : $var"}
```

format

Description

Format a string in the style of sprintf

Syntax

format formatString [arg arg ...]

Notes

This command generates a formatted string in the same way as the ANSI C sprintf procedure.

FormatString indicates how to format the result, using % conversion specifiers as in sprintf, and the additional arguments, if any, provide values to be substituted into the result.

Returns and Formatting

The return value is the formatted string.

The command operates by scanning formatString from left to right. Each character from the format string is appended to the result string unless it is a percent sign.

If the character is "%" then it is not copied to the result string. Instead, the characters following the % character are treated as a conversion specifier.

The conversion specifier controls the conversion of the next successive arg to articular format and the result is appended to the result string in place of the conversion specifier.

If there are multiple conversion specifiers in the format string, then each one controls the conversion of one additional arg.

The format command must be given enough args to meet the needs of all of the conversion specifiers in formatString.

Each conversion specifier may contain up to six different parts:

A set of flags, a minimum field width, a precision, a length modifier, and a conversion character. Any of these fields may be omitted except for the conversion character.

The fields that are present must appear in the order given above. The paragraphs below discuss each of these fields in turn.

If the % is followed by a decimal number and a $, as in "%2$d", then the value to convert is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first arg.

If the conversion specifier requires multiple arguments because of * characters in the specifier then successive arguments are used, starting with the argument given by the number.

If there are any positional specifiers in formatString then all of the specifiers must be positional.

The second portion of a conversion specifier may contain any of the following flag characters, in any order:

\-

Specifies that the converted argument should be left-justified in its field (numbers are normally right-justified with leading spaces if needed).

+

Specifies that a number should always be printed with a sign, even if positive.

space

Specifies that a space should be added to the beginning of the number if the first character isn't a sign.

0

Specifies that the number should be padded on the left with zeroes instead of spaces.

#

Requests an alternate output form. For o and O conversions it guarantees that the first digit is always 0.

For x or X conversions, 0x or 0X (respectively) will be added to the beginning of the result unless it is zero. For all floating-point conversions (e, E, f, g, and G) it guarantees that the result always has a decimal point.

For g and G conversions it specifies that trailing zeroes should not be removed.

The third portion of a conversion specifier is a number giving a minimum field width for this conversion. It is typically used to make columns line up in tabular printouts.

If the converted argument contains fewer characters than the minimum field width then it will be padded so that it is as wide as the minimum field width.

Padding normally occurs by adding extra spaces on the left of the converted argument, but the 0 and \- flags may be used to specify padding with zeroes on the left or with spaces on the right, respectively.

If the minimum field width is specified as * rather than a number, then the next argument to the format command determines the minimum field width; it must be a numeric string.

The fourth portion of a conversion specifier is a precision, which consists of a period followed by a number. The number is used in different ways for different conversions.

For e, E, and f conversions it specifies the number of digits to appear to the right of the decimal point.

For g and G conversions it specifies the total number of digits to appear, including those on both sides of the decimal point (however, trailing zeroes after the decimal point will still be omitted unless the # flag has been specified).

For integer conversions, it specifies a minimum number of digits to print (leading zeroes will be added if necessary). For s conversions it specifies the maximum number of characters to be printed; if the string is longer than this then the trailing characters will be dropped.

If the precision is specified with * rather than a number then the next argument to the format command determines the precision, it must be a numeric string.

The fourth part of a conversion specifier is a length modifier, which must be h or l.

If it is h it specifies that the numeric value should be truncated to a 16-bit value before converting. This option is rarely useful. The l modifier is ignored.

The last thing in a conversion specifier is an alphabetic character that determines what kind of conversion to perform.

The following conversion characters are currently supported:

d

Convert integer to signed decimal string.

u

Convert integer to unsigned decimal string.

i

Convert integer to signed decimal string; the integer may either be in decimal, in octal (with a leading 0) or in hexadecimal (with a leading 0x).

o

Convert integer to unsigned octal string.

x or X

Convert integer to unsigned hexadecimal string, using digits "0123456789abcdef" for x and "0123456789ABCDEF" for X).

c

Convert integer to the 8-bit character it represents.

s

No conversion; just insert string.

f

Convert floating-point number to signed decimal string of the form xx.yyy, where the number of y's is determined by the precision (default: 6).

If the precision is 0 then no decimal point is output.

e or E

Convert floating-point number to scientific notation in the form x.yyye+-zz, where the number of y's is determined by the precision (default: 6).

If the precision is 0 then no decimal point is output. If the E form is used then E is printed instead of e.

g or G

If the exponent is less than \-4 or greater than or equal to the precision, then convert floating-point number as for %e or %E. Otherwise convert as for %f.

Trailing zeroes and a trailing decimal point are omitted.

%

No conversion: just insert %.

For the numerical conversions the argument being converted must be an integer or floating-point string; format converts the argument to binary and then converts it back to a string according to the conversion specifier.

gets

Description

Gets file.

Syntax

gets fileId varName

Notes

This command reads the next line from the file given by file and discards the terminating newline character.

If varName is specified then the line is placed in the variable by that name and the return value is a count of the number of characters read (not including the newline).

If the end of the file is reached before reading any characters then (-1) is returned and varName is set to an empty string.

If varName is not specified then the return value will be the line (minus the newline character) or an empty string if the end of the file is reached before reading any characters.

An empty string will also be returned if a line contains no characters except the newline, so **eof** may have to be used to determine what really happened.

If the last character in the file is not a newline character then gets behaves as if there were an additional newline character at the end of the file.

File must be the return value from a previous call to **open.** it must refer to a file that was opened for reading.

Returns

Returns the line length (if varName specified) or the line itself.

Example

Gets first line from "sales.dat" file.

```
fileId = [open sales.dat]
gets $fileId data
close $fileId
```

glob

Description

Return names of files that match patterns.

Syntax

glob swiches [pattern pattern ...]

Notes

This command performs file name "globbing" in a fashion similar to the CSH shell. It returns a list of the files whose names match any of the pattern arguments.

If the initial arguments to **glob** start with - then they are treated as swiches. The following swiches are currently supported:

-nocomplain

Allows an empty list to be returned without error; without this switch an error is returned if the result list would be empty.

--

Marks the end of swiches. The argument following this one will be treated as a pattern even if it starts with a "-".

The pattern arguments may contain any of the following special characters:

?

Matches any single character.

*

Matches any sequence of zero or more characters.

[chars]

Matches any single character in chars. If chars contains a sequence of the form a-b then any character between a and b (inclusive) will match.

x

Matches the character x.

{a,b,...}

Matches any of the strings a, b, etc.

As with csh, a "." at the beginning of a file's name or just after a "\" must be matched explicitly or with a {} construct. In addition, all "\" characters must be matched explicitly.

Returns

Returns the files list.

Example

Displays all files with ".dat" extension.

```
message [glob *.dat]
```

global

Description

Access global variables

Syntax

global varname [varname ...]

Notes

This command is ignored unless a PSL procedure is being interpreted. If so then it declares the given varname's to be global variables rather than local ones. For the duration of the current procedure (and only while executing in the current procedure), any reference to any of the varnames will refer to the global variable by the same name.

Returns

Returns an empty string.

Example

```
global varname1 varname2
```

if

Description

Execute scripts conditionally.

Syntax

if expr1 [**then**] body1 [**elseif** expr2 [**then**] body2] [**elseif** ...] [[**else**] bodyN]

Notes

The **if** command evaluates expr1 as an expression (in the same way that the **expr** command evaluates its argument). The value of the expression must be a boolean (a numeric value, where 0 is false and anything else is true, or a string value such as true or yes for true and false or no for false).

If it is true then body1 is executed by passing it to the PSL.

Otherwise expr2 is evaluated as an expression and if it is true then body2 is executed, and so on.

If none of the expressions evaluates to true then bodyN is executed.

The **then** and **else** arguments are optional, to make the command easier to read.

There may be any number of **elseif** clauses, including zero.

BodyN may also be omitted as long as else is omitted too.

Returns

The return value from the command is the result of the body script that was executed, or an empty string, if none of the expressions was non-zero and there was no bodyN.

Example

Displays "yes" if variable host equels "vax", or "no" if it is not.

```
if {$host == "vax"} {message "yes"} else {message "no"}
```

incr

Description

Increment the value of a variable

Syntax

incr varName increment

Notes

Increments the value stored in the variable whose name is varName. The value of the variable must be an integer. If increment is supplied then its value (which must be an integer) is added to the value of variable varName, otherwise 1 is added to varName.

The new value is stored as a decimal string in variable varName and also returned as result.

Returns

Rreturn the new varName value.

Example

Use the **incr** command to increase **for** loop counter.

for {i = 0} {$i < 10} {**incr** i} {commands...}

info

Description

Return information about the state of the PSL interpreter

Syntax

info option [arg arg ...]

Notes

This command provides information about various internals of the PSL interpreter.

The legal option's (which may be abbreviated) are:

info args procname

Returns a list containing the names of the arguments to procedure procname, in order. Procname must be the name of a PSL command procedure.

info body procname

Returns the body of procedure procname. Procname must be the name of a PSL command procedure.

info cmdcount

Returns a count of the total number of commands that have been invoked in this interpreter.

info commands [pattern}

If pattern isn't specified, returns a list of names of all the PSL commands, including both the built-in commands and the command procedures defined using the **proc** command. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

info complete command

Returns 1 if command is a complete PSL command in the sense of having no unclosed quotes, braces, brackets or array element names, If the command doesn't appear to be complete then 0 is returned. This command is typically used in line-oriented input environments to allow users to type in commands that span multiple lines; if the command isn't complete, the script can delay evaluating it until additional lines have been typed to complete the command.

info default procname arg varname

Procname must be the name of a PSL command procedure and arg must be the name of an argument to that procedure. If arg doesn't have a default value then the command returns 0. Otherwise it returns 1 and places the default value of arg into variable varname.

info exists varName

Returns 1 if the variable named varName exists in the current context (either as a global or local variable), returns 0 otherwise.

info globals [pattern]

If pattern isn't specified, returns a list of all the names of currently-defined global variables. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

info level [number]

If number is not specified, this command returns a number giving the stack level of the invoking procedure, or 0 if the command is invoked at top-level. If number is specified, then the result is a list consisting of the name and arguments for the procedure call at level number on the stack. If number is positive then it selects a particular stack level (1 refers to the top-most active procedure, 2 to the procedure it called, and so on), otherwise it gives a level relative to the current level (0 refers to the current procedure, -1 to its caller, and so on).

See the **uplevel** command for more information on what stack levels mean.

info library

Returns the name of the library directory in which standard PSL scripts are stored.

The default value for the library is "lib", but it may be overridden by setting the PSL_LIBRARY environment variable.

info locals [pattern]

If pattern isn't specified, returns a list of all the names of currently-defined local variables, including arguments to

the current procedure, if any.

Variables defined with the **global** and **upvar** commands will not be returned.

If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

info procs [pattern]

If pattern isn't specified, returns a list of all the names of PSL command procedures.

If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

info script

If a PSL script file is currently being evaluated, then this command returns the name of the innermost file being processed. Otherwise the command returns an empty string.

info vars [pattern]

If pattern isn't specified, returns a list of all the names of currently-visible variables, including both locals and currently-visible globals. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

Returns

Returns the appropriate info value.

Example

Displays "Yes" if variable varName exists, otherwise "No".

```
if {[info exists varName] == "vax"} {message "yes"} else {message "no"}
```

join

Description

Create a string by joining together list elements

Syntax

join list [joinString]

Notes

The list argument must be a valid PSL list. This command returns the string formed by joining all of the elements of list together with joinString separating each adjacent pair of elements. The joinString arrgument defaults to a space character.

Returns

Returns the list items separated by joinString.

Example

Displays "1#2#3#4#5#6#7".

```
message [join {1 2     3     4    5 6              7} #]
```

key

Description

Map a key to do a certain action

Syntax

key [**alt+**][**ctrl+**][**shift+**]pckey option [args...]

Notes

A PC key or a terminal key may be a combination of control keys plus another key.

Performs one of several key operations, depending on option. The legal options are:

key [alt+][ctrl+][shift+]pckey run scriptName

Maps a PC key to run a PSL script.

key [alt+][ctrl+][shift+]pckey do commands

Maps a PC key to executes PSL commands.

key [alt+][ctrl+][shift+]pckey send [alt+][ctrl+][shift+]vtkey

Maps a PC key to send a VT key.

Note: the VT key above is a Virtual Terminal key, not Digital's VT.

The available PC keys are:

- a-z, 0-9,
- special symbol keys: (` - = \ [ ] ; ' , . /),
- space, tab, capslock, esc, backspace, return,
- right, left, up, down,
- insert, delete, home, end, pgup, pgdn,
- f1-f12,
- print, scroll, pause,
- numlock, divide, multiply, subtract, add, decimal, enter,
- numpad0-numpad9

The available VT keys are:

- esc, tab, backspace, return, f1-f20, help, do,
- udk6-udk20,
- pf1-pf4, comma, minus, enter,
- numpad0-numpad9
- insert, remove, find, select, next, prev,
- delete, home, end, pgup, pgdn,
- right, left, up, down,
- divide, multiply, subtract, add, decimal, enter,
- c1-c4, eop, eol

Note: because of support for several terminals, not all keys supports all terminal keyboards, yet it is quite easy to find the name of a requested key in any terminal keyboard.

Returns

Returns an empty string.

Example

The following line maps ctrl+alt+f5 to activate notepad.exe.

```
key ctrl+alt+f5 do {exec notepad.exe}
```

The following line maps shift+alt+a to run script menu.psl.

```
key shift+alt+a run menu.psl
```

The following line maps the scroll lock key to send do.

```
key scroll send do
```

The following line maps alt+f6 to send ctrl+g

```
key alt+f6 send ctrl+g
```

lappend

Description

Append list elements onto a variable

Syntax

lappend varName value [value value ...]

Notes

This command treats the variable given by varName as a list and appends each of the value arguments to that list as a separate element, with spaces between elements.

If varName doesn't exist, it is created as a list with elements given by the value arguments. lappend is similar to append except that the values are appended as list elements rather than raw text. This command provides a relatively efficient way to build up large lists. For example, "**lappend** a $b" is much more efficient than "a = [**concat** $a [**list** $b]]" when $a is long.

Returns

Returns the concatenated list.

Example

The following displays "a b {1 2 3} d {1 2 3} 4 5"

```
a = {a b {1 2 3} d}

message [lappend a {1 2 3} 4 5]
```

lindex

Description

Retrieve an element from a list

Syntax

lindex list index

Notes

This command treats list as a PSL list and returns the lindex'th element from it (0 refers to the first element of the list). In extracting the element, **lindex** observes the same rules concerning braces and quotes and backslashes as the PSL command interpreter; however, variable substitution and command substitution do not occur.

Returns

Returns the specified item. If lindex is negative or greater than or equal to the number of elements in value, then an empty string is returned.

Example

Displays second item from a list in variable List.

```
message [lindex $List 2]
```

linsert

Description

Insert elements into a list

Syntax

linsert list index element [element element ...]

Notes

This command produces a new list from list by inserting all of the element arguments just before the index element of list. Each element argument will become a separate element of the new list. If index is less than or equal to zero, then the new elements are inserted at the beginning of the list. If index is greater than or equal to the number of elements in the list, then the new elements are appended to the list.

Returns

Returns the new list with old and new items.

Example

Thew line inserts items "vax {hp digital ibm} dg" after item 3 in List.

```
message [linsert $List 3 vax {hp digital ibm} dg]
```

list

Description

Create a list

Syntax

list arg [arg arg ...]

Notes

This command returns a list comprised of all the args. Braces and backslashes get added as necessary, so that the index command may be used on the result to re-extract the original arguments, and also so that eval may be used to execute the resulting list, with arg comprising the command's name and the other args comprising its arguments. list produces slightly different results than concat:

concat removes one level of grouping before forming the list, while list works directly from the original arguments.

Returns

Returns the new list.

Example

```
list a b {c d e} {f {g h}}
```

will return "a b {c d e} {f {g h}}"

```
concat a b {c d e} {f {g h}}
```

will return "a b c d e f {g h}"

llength

Description

Count the number of elements in a list

Syntax

llength list

Notes

Treats list as a list and returns a decimal string giving the number of elements in it.

Example

```
llength {1 2 {ab cd fff} {18 19}}
```

will return 4.

lrange

Description

Return one or more adjacent elements from a list

Syntax

lrange list first last

Notes

list must be a valid PSL list. This command will return a new list consisting of elements first through last, inclusive. Last may be end (or any abbreviation of it) to refer to the last element of the list. If first is less than zero, it is treated as if it were zero. If last is greater than or equal to the number of elements in the list, then it is treated as if it were end. If first is greater than last then an empty string is returned.

"lrange list first first" does not always produce the same result as "lindex list first" (although it often does for simple fields that aren't enclosed in braces), it does, however, produce exactly the same results as "list [lindex list first]"

Returns

Returns the items in the range.

Example

lrange "1 2 3 {3 4 5} a { b c} d" 2 5

will return "3 {3 4 5} a { b c}"

lreplace

Description

Replace elements in a list with new elements

Syntax

lreplace list first last [element element ...]

Notes

lreplace returns a new list formed by replacing one or more elements of list with the element arguments. First gives the index in list of the first element to be replaced.

If first is less than zero then it refers to the first element of list, the element indicated by first must exist in the list. Last gives the index in list of the last element to be replaced; it must be greater than or equal to first. Last may be end (or any abbreviation of it) to indicate that all elements between first and the end of the list should be replaced.

The element arguments specify zero or more new arguments to be added to the list in place of those that were deleted. Each element argument will become a separate element of the list. If no element arguments are specified, then the elements between first and last are simply deleted.

Returns

Returns the new list.

Example

```
lreplace "1 2 3 {3 4 5} a { b c} d" 2 5 one two
```

will return "1 2 one two d".

lsearch

Description

See if a list contains a particular element

Syntax

lsearch [mode] list pattern

Notes

This command searches the elements of list to see if one of them matches pattern.

If so, the command returns the index of the first matching element.

If not, the command returns (-1).

The mode argument indicates how the elements of the list are to be matched against pattern and it must have one of the following values:

-exact

The list element must contain exactly the same string as pattern.

-glob

Pattern is a glob-style pattern which is matched against each list element using the same rules as the **string match** command.

-regexp

Pattern is treated as a regular expression and matched against each list element using the same rules as the **regexp** command.

If mode is omitted then it defaults to **-glob**.

Returns

Returns the index of the first matched item or -1.

Example

```
lsearch "vax unix ibm" *n*
```

will find the item "unix" and return 1.

lsort

Description

Sort the elements of a list

Syntax

lsort [switches] list

Notes

This command sorts the elements of list, returning a new list in sorted order. By default ASCII sorting is used with the result returned in increasing order.

However, any of the following switches may be specified before list to control the sorting process (unique abbreviations are accepted):

-ascii

Use string comparison with ASCII collation order. This is the default.

-integer

Convert list elements to integers and use integer comparison.

-real

Convert list elements to floating-point values and use floating comparison.

-command command

Use command as a comparison command. To compare two elements, evaluate a PSL script consisting of command with the two elements appended as additional arguments. The script should return an integer less than, equal to, or greater than zero if the first element is to be considered less than, equal to, or greater than the second, respectively.

-increasing

Sort the list in increasing order ("smallest" items first). This is the default.

-decreasing

Sort the list in decreasing order ("largest" items first).

Returns

Returns the sorted list.

Example

This line returns "ibm unix vax".

```
lsort "vax unix ibm"
```

menu

Description

Change menu type

Syntax

menu option

Notes

Changes PowerTerm main menu type.

menu hide

Removes the main menu.

menu minimize

Minimizes the main menu (with one click on the menu you can restore it).

menu restore

Restores the menu to its normal state.

Example

This line hides the menu.

```
menu hide
```

message

Description

Display a message

Syntax

message string [**title** text] [option]

Notes

Diplays the message string on the screen.

The "**title** text" option replaces the default "PowerTerm" title with text.

The option can be one of the following and will change the type of the message box.

info, error, question

Example

Displays the message "Hello" with title "Welcome" and message type info.

```
message Hello title Welcome info
```

open

Description

Open a file

Syntax

open fileName [access] [permissions]

Notes

This command opens a file and returns an identifier that may be used in future invocations of commands like **read**, **gets**, **puts**, and **close**. fileName gives the name of the file to open.

The access argument indicates the way in which the file is to be accessed. It may take two forms, either a string or a list of POSIX access flags. It defaults to ``r''.

In the first form access may have any of the following values:

r

Open the file for reading only; the file must already exist.

r+

Open the file for both reading and writing; the file must already exist.

w

Open the file for writing only. Truncate it if it exists. If it doesn't exist, create a new file.

w+

Open the file for reading and writing. Truncate it if it exists. If it doesn't exist, create a new file.

a

Open the file for writing only. The file must already exist, and the file is positioned so that new data is appended to the file.

a+

Open the file for reading and writing. If the file doesn't exist, create a new empty file.

Sets the initial access position to the end of the file.

In the second form, access consists of a list of any of the following flags, all of which have the standard POSIX meanings. One of the flags must be either RDONLY, WRONLY or RDWR.

RDONLY

Open the file for reading only.

WRONLY

Open the file for writing only.

RDWR

Open the file for both reading and writing.

APPEND

Set the file pointer to the end of the file prior to each write.

CREAT

Create the file if it doesn't already exist (without this flag it is an error for the file not to exist).

EXCL

If CREAT is specified also, an error is returned if the file already exists.

NOCTTY

If the file is a terminal device, this flag prevents the file from becoming the controlling terminal of the process.

NONBLOCK

Prevents the process from blocking while opening the file. For details refer to your system documentation on the open system call's O_NONBLOCK flag.

TRUNC

If the file exists it is truncated to zero length. If a new file is created as part of opening it, permissions (an integer) is used to set the permissions for the new file in conjunction with the process's file mode creation mask. permissions defaults to 0666. If a file is opened for both reading and writing then seek must be invoked between a read and a write, or vice versa (this restriction does not apply to command pipelines opened with open).

Returns

Returns the file ID.

Example

The following opens file "output.dat" for writing, writes data and closes.

```
File = [open output.dat w]

pets $File "message test"

close $File
```

open-setup-file

Description

Open setup file

Syntax

open-setup-file file-name

Notes

Opens a saved setup file, replacing the PowerTerm parameters. (Similar to File | Open).

Returns

This command returns an empty string.

Example

The following line opens setup file "prog.pts".

```
open-setup-file ptog.pts
```

proc

Description

Create a PSL procedure

Syntax

proc name args body

Notes

The proc command creates a new PSL procedure named name, replacing any existing command or procedure there may have been by that name. Whenever the new command is invoked, the contents of body will be executed by the PSL interpreter. Args specifies the formal arguments to the procedure. It consists of a list, possibly empty, each of those elements specifies one argument. Each argument specifier is also a list with either one or two fields. If there is only a single field in the specifier then it is the name of the argument; if there are two fields, then the first is the argument name and the second is its default value.

When name is invoked a local variable will be created for each of the formal arguments to the procedure, its value will be the value of corresponding argument in the invoking command or the argument's default value.

Arguments with default values need not be specified in a procedure invocation. However, there must be enough actual arguments for all the formal arguments that don't have defaults, and there must not be any extra actual arguments. There is one special case to permit procedures with variable numbers of arguments. If the last formal argument has the name args, then a call to the procedure may contain more actual arguments than the procedure has formals. In this case, all of the actual arguments starting at the one that would be assigned to args are combined into a list (as if the list command had been used), this combined value is assigned to the local variable args.

When body is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when the procedure returns. One local variable is automatically created for each of the procedure's arguments.

Global variables can only be accessed by invoking the **global** command or the **upvar** command.

When a procedure is invoked, the procedure's return value is the value specified in a return command. If the procedure doesn't execute an explicit return, then its return value is the value of the last command executed in the procedure's body. If an error occurs while executing the procedure body, then the procedure-as-a-whole will return that same error.

Returns

Returns an empty string.

Example

This command defines a recursive factorial procedure:

```
proc factorial x {
        if {$x == 1} {return 1}
        return [expr {$x * [factorial [expr $x - 1]]}]
}
message [facorial 4] will display 24.
```

puts

Description

Write to a file

Syntax

puts [**-nonewline**] fileId string

Notes

Writes the characters given by string to the file given by fileId. FileId must have been the return value from a previous call to open. It must refer to a file that was opened for writing.

puts normally outputs a newline character after string, but this feature may be suppressed by specifying the **-nonewline** switch. Output to files is buffered internally by PSL, the flush command may be used to force buffered characters to be output.

Returns

Returns an empty string.

Example

The following opens file "data" for writing, writes data and closes.

```
outFile = [open data w]
puts $outFile Information
close $outFile
```

pwd

Description

Return the current working directory

Syntax

pwd

Notes

Returns the path name of the current working directory.

Example

Displays the current working directory.

```
message [pwd]
```

read

Description

Read from a file

Syntax

read [-**nonewline**] fileId

or

read fileId numBytes

Notes

In the first form, all of the remaining bytes are read from the file given by fileId, they are returned as the result of the command. If the **-nonewline** switch is specified then the last character of the file is discarded if it is a newline. In the second form, the extra argument specifies how many bytes to read, exactly this many bytes will be read and returned, unless there are fewer than numBytes bytes left in the file, in this case, all the remaining bytes are returned. FileId must the return value from a previous call to open, it must refer to a file that was opened for reading.

Returns

Returns the data read from the file.

Example

The following opens file "data" for reading. reads 10 bytes and closes.

```
inFile = [open data]
data = [read $inFile 20]
close $inFile
```

regexp

Description

Match a regular expression against a string

Syntax

regexp [switches] exp string [matchVar] [subMatchVar subMatchVar ...]

Notes

Determines whether the regular expression exp matches part or all of string and returns 1 if it does, 0 if it doesn't.

If additional arguments are specified after string then they are treated as the names of variables in which to return information about which part(s) of string matched exp.

MatchVar will be set to the range of string that matched all of exp. The first subMatchVar will contain the characters in string that matched the leftmost parenthesized subexpression within exp, the next subMatchVar will contain the characters that matched the next parenthesized subexpression to the right in exp, and so on.

If the initial arguments to regexp start with "-" then they are treated as switches. The following switches are supported:

-nocase

Causes upper-case characters in string to be treated as lower case during the matching process.

-indices

Changes what is stored in the subMatchVars. Instead of storing the matching characters from string, each variable will contain a list of two decimal strings giving the indices in string of the first and last characters in the matching range of characters.

--

Marks the end of switches. The argument following this one will be treated as exp even if it starts with a "-".

If there are more subMatchVar's than parenthesized subexpressions within exp, or if a particular subexpression in exp doesn't match the string (e.g. because it was in a portion of the expression that wasn't matched), then the corresponding subMatchVar will be set to "-1 -1" if **-indices** has been specified or to an empty string otherwise.

Regular Expressions

A regular expression is zero or more branches, separated by "|". It matches anything that matches one of the branches. A branch is zero or more pieces, concatenated.

It matches a match for the first, followed by a match for the second, etc.

A piece is an atom possibly followed by "*", "+", or "?".

An atom followed by "*" matches a sequence of 0 or more matches of the atom. An atom followed by "+" matches a sequence of 1 or more matches of the atom. An atom followed by "?" matches a match of the atom, or the null string.

An atom is a regular expression in parentheses (matching a match for the regular expression), a range (see below), "." (matching any single character), "^" (matching the null string at the beginning of the input string), "$" (matching the null string at the end of the input string), a "\" followed by a single character (matching that character), or a single character with no other significance (matching that character).

A range is a sequence of characters enclosed in "[]". It normally matches any single character from the sequence. If the sequence begins with "^", it matches any single character not from the rest of the sequence. If two characters in the sequence are separated by "-", this is shorthand for the full list of ASCII characters between them (e.g. "[0-9]" matches any decimal digit). To include a literal "]" in the sequence, make it the first character (following a possible "^"). To include a literal "-", make it the first or last character.

Choosing Among Aternative Matches

In general there may be more than one way to match a regular expression to an input string. For example, consider the command,

```
regexp (a*)b* aabaaabb x y
```

Considering only the rules given so far, x and y could end up with the values aabb and aa, aaab and aaa, ab and a, or any of several other combinations. To resolve this potential ambiguity regexp chooses among alternatives using the rule "first then longest". In other words, it consders the possible matches in order working from left to right across the input string and the pattern, and it attempts to match longer pieces of the input string before

shorter ones. More specifically, the following rules apply in decreasing order of priority:

If a regular expression could match two different parts of an input string then it will match the one that begins earliest.

If a regular expression contains "|" operators then the leftmost matching sub-expression is chosen.

In "*", "+", and "?" constructs, longer matches are chosen in preference to shorter ones.

In sequences of expression components the components are considered from left to right. In the example from above, (a*)b* matches aab: the (a*) portion of the pattern is matched first and it consumes the leading aa, then the b* portion of the pattern consumes the next b.

Returns

Returns 1 if it matches, 0 if it doesn't.

Example

consider the following example:

regexp (ab|a)(b*)c abc x y z

After this command x will be abc, y will be ab, and z will be an empty string.

Rule 4 specifies that (ab|a) gets first shot at the input string and Rule 2 specifies that the ab sub-expression is checked before the a sub-expression. Thus the b has already been claimed before the (b*) component is checked and (b*) must match an empty string.

regsub

Description

Perform substitutions based on regular expression pattern matching

Syntax

regsub [switches] exp string subSpec varName

Notes

This command matches the regular expression exp against string, and it copies string to the variable whose name is given by varName. The command returns 1 if there is a match and 0 if there isn't. If there is a match, then while copying string to varName the portion of string that matched exp is replaced with subSpec. If subSpec contains a "&" or "0", then it is replaced in the substitution with the portion of string that matched exp.

If subSpec contains a "n", where n is a digit between 1 and 9, then it is replaced in the substitution with the portion of string that matched the n-th parenthesized subexpression of exp. Additional backslashes may be used in subSpec to prevent special interpretation of "&" or "0" or "n" or backslash.

The use of backslashes in subSpec tends to interact badly with the PSL parser's use of backslashes, so it's generally safest to enclose subSpec in braces if it includes backslashes.

If the initial arguments to regexp start with "-" then they are treated as switches. The following switches are supported:

-all

All ranges in string that match exp are found and substitution is performed for each of these ranges. Without this switch only the first matching range is found and substituted.

If **-all** is specified, then "&" and "n" sequences are handled for each substitution using the information from the corresponding match.

-nocase

Uppercase characters in string will be converted to lower-case before matching against exp, however, substitutions specified by subSpec use the original unconverted form of string.

--

Marks the end of switches. The argument following this one will be treated as exp even if it starts with a "-".

Returns

Returns 1 if it matches, 0 if it doesn't.

Example

Returns 1 and variable new will contain "*abc*abc*def".

The "abc" portion of "abcdef" was substituted by "*abc*abc*" and the "def" portion was left a the end.

```
regsub (ab|a)(b*)c abcdef *&*&* new
```

rename

Description

rename or delete a command

Syntax

rename oldName newName

Notes

Renames the command that used to be called oldName so that it is now called newName. If newName is an empty string then oldName is deleted.

Returns

The **rename** command returns an empty string as result.

Example

Renames the **for** command to **loop** and then use it.

rename for loop

loop {i = 0} {$i < 10} {incr i} {commands...}

return

Description

Return from a procedure

Syntax

return [**-code** code] [**-errorcode** code] [string]

Notes

Returns immediately from the current procedure (or top-level command or **run** command), with string as the return value. If string is not specified then an empty string will be returned as result.

Exceptional Returns

In the usual case where the **-code** option isn't specified the procedure will return normally. However, the **-code** option may be used to generate an xeptional return from the procedure. code may have any of the following values:

ok

Normal return: same as if the option is omitted.

error

Error return: same as if the error command were used to terminate the procedure, except for handling of errorInfo and errorCode variables (see below).

return

The current procedure will return with a completion code of PSL_RETURN, so that the procedure that invoked it will return also.

break

The current procedure will return with a completion code of PSL_BREAK, which will terminate the innermost nested loop in the code that invoked the current procedure.

continue

The current procedure will return with a completion code of PSL_CONTINUE, which will terminate the current iteration of the innermost nested loop in the code that invoked the current procedure.

value

Value must be an integer; it will be returned as the completion code for the current procedure.

The **-code** option is rarely used. It is provided so that procedures that implement new control structures can reflect exceptional conditions back to their callers.

An additional option, **-errorcode**, may be used to provide additional information during error returns.

This option is ignored unless code is error.

If the **-errorcode** option is specified then code provides a value for the errorCode variable. If the option is not specified then errorCode will default to NONE.

Returns

Returns the string specified or an empty string.

Example

Defines a division procedure with two parameters.

If the second parameter is zero, return with **continue** error code, otherwise return division of the two.

In the rest of the code, in a loop: inputs two numbers and activate the divide procedure. If second parameter is zero divide will act as a **continue** command and return to start of the **while** loop, otherwise division result will be displayed and the loop will be terminated.

```
proc divide {x y} {
if {$y == 0} {return -code continue 0} else {
return [expr $x / $y]}
}
```

```
while {1} {
#       commands to input data for variables x & y ...
        result = [divide $x $y]
        message "$x / $y = $result"
        break
}
```

run

Description

Evaluate a file as a PSL script

Syntax

run fileName [parameters...]

Notes

Read file fileName and pass the contents to the PSL interpreter as a script to evaluate in the normal fashion. The return value from **run** is the return value of the last command executed from the file. If an error occurs in evaluating the contents of the file then the **run** command will return that error. If a **return** command is invoked from within the file then the remainder of the file will be skipped and the **run** command will return normally with the result from the **return** command.

If parameters are included, variables named $p1 to $pN will exists, according to number of parameters.

Returns

Returns the value from the last command in the script.

Example

run test.psl

scan

Description

Parse string using conversion specifiers in the style of sscanf.

Syntax

scan string format varName [varName ...]

Notes

This command parses fields from an input string in the same fashion as the ANSI C sscanf procedure and returns a count of the number of fields sucessfully parsed.

string gives the input to be parsed and format indicates how to parse it, using % conversion specifiers as in sscanf. Each varName gives the name of a variable, when a field is scanned from string the result is converted back into a string and assigned to the corresponding variable.

Details on Scanning

scan operates by scanning string and formatString together. If the next character in formatString is a blank or tab then it is ignored. Otherwise, if it isn't a % character then it must match the next non-white-space character of string. When a % is encountered in formatString, it indicates the start of a conversion specifier.

A conversion specifier contains three fields after the %:

a *, which indicates that the converted value is to be discarded instead of assigned to a variable; a number indicating a maximum field width; and a conversion character.

All of these fields are optional except for the conversion character.

When **scan** finds a conversion specifier in formatString, it first skips any white-space characters in string. Then it converts the next input characters according to the conversion specifier and stores the result in the variable given by the next argument to scan. The following conversion characters are supported:

d

The input field must be a decimal integer. It is read in and the value is stored in the variable as a decimal string.

o

The input field must be an octal integer. It is read in and the value is stored in the variable as a decimal string.

x

The input field must be a hexadecimal integer. It is read in and the value is stored in the variable as a decimal string.

c

A single character is read in and its binary value is stored in the variable as a decimal string. Initial white space is not skipped in this case, so the input field may be a white-space character. This conversion is different from the ANSI standard in that the input field always consists of a single character and no field width may be specified.

s

The input field consists of all the characters up to the next white-space character; the characters are copied to the variable.

e or **f** or **g**

The input field must be a floating-point number consisting of an optional sign, a string of decimal digits possibly containing a decimal point, and an optional exponent consisting of an **e** or **E** followed by an optional sign and a string of decimal digits. It is read in and stored in the variable as a floating-point string.

[chars]

The input field consists of any number of characters in chars. The matching string is stored in the variable. If the first character between the brackets is a "]" then it is treated as part of chars rather than the closing bracket for the set.

[^chars]

The input field consists of any number of characters not in chars.

The matching string is stored in the variable. If the character immediately following the "^" is a "]" then it is treated as part of the set rather than the closing bracket for the set.

The number of characters read from the input for a conversion is the largest number that makes sense for that

particular conversion (e.g. as many decimal digits as possible for %d, as many octal digits as possible for %o, and so on).

The input field for a given conversion terminates either when a white-space character is encountered or when the maximum field width has been reached, whichever comes first. If a "*" is present in the conversion specifier then no variable is assigned and the next scan argument is not consumed.

Returns

Returns the number of scanned parameters and fills the variables with their values.

Example

This script scans the following string and sets the variables:

var1 = 123 var2 = 65 ("A" ASCII) var3 = unix.

```
scan "123 A unix" "%d %c %s" var1 var2 var3
```

screen

Description

Copy data from the screen

Syntax

screen startRow startCol [endRow] endCol

Notes

screen copies complete lines from the starting position (startRow, startCol) to and including the end position (endRow, endCol).

If endRow is not specified endRow equals startRow.

Returns

Returns the data copied from the screen.

Example

Consider the following screen:

Line 1 : "line 1: 1234567890"

Line 2 : "line 2: 1234567890"

Line 3 : "line 3: 1234567890"

Line 4 : "line 4: 1234567890"

data = [**screen** 2 15 4 10]

Sets variable data to screen data from (2, 15) to (4, 10) :

"7890\nline 2: 1234567890\nline 3: 12"

where "\n" is a line separator.

screen-rect

Description

Copy data from the screen

Syntax

screen-rect startRow startCol [endRow] endCol

Notes

screen-rect copies rectangular data from the starting position (startRow, startCol) to and including the end position (endRow, endCol).

If endRow is not specified endRow equals startRow.

Returns

Returns the data copied from the screen.

Example

Consider the following screen:

Line 1 : "line 1: 1234567890"

Line 2 : "line 2: 1234567890"

Line 3 : "line 3: 1234567890"

Line 4 : "line 4: 1234567890"

data = [**screen-rect** 2 2 4 15]

Sets variable data to screen data from (2, 2) to (4, 15) :

"ine 1: 1234567\nine 2: 1234567\nine 3: 1234567\n"

where "\n" is a line separator.

seek

Description

Change the access position for an open file

Syntax

seek fileId offset [origin]

Notes

Change the current access position for fileId. FileId must have been the return value from a previous call to open.

The offset and origin arguments specify the position at which the next read or write will occur for fileId. Offset must be an integer (which may be negative) and origin must be one of the following:

start

The new access position will be offset bytes from the start of the file.

current

The new access position will be offset bytes from the current access position; a negative offset moves the access position backwards in the file.

end

The new access position will be offset bytes from the end of the file. A negative offset places the access position before the end-of-file, and a positive offset places the access position after the end-of-file.

The origin argument defaults to **start**.

Returns

This command returns an empty string.

Example

The following opens file "file1" for reading. Moves to position 55 in the file. Reads 10 bytes to variable data and closes file.

```
fileId = [open file1]

seek $fileId 55 start

data = [read $fileId 10]

close $fileId
```

send

Description

Sends data to the host.

Syntax

send data

Notes

The command sends data like it was typed from the keyboard.

Returns

Returns an empty string.

Example

The host will receive the following data and will show the current directory.

send "dir^M"

Note: "^M" sends ctrl-m. The "^" sign followed by a letter represents the control code of that letter.

session

Description

Modify the communication session status.

Syntax

session option

Notes

Performs one of several session operations, depending on option. The legal options are:

session open

Opens a session according to communication parameters previously defained with the **set** command.

session modify

Modifies the current session according to communication parameters previously defained with the **set** command.

session close

Closes the current session.

Returns

Returns an empty string.

Example

1. Opens a COM session with the following parameters.

```
set comm-type com

set port-number 2

set baud-rate 19200

set protocol-type xonxoff

session open
```

2. Modifies the COM session to 9600 buad-rate.

```
set baud-rate 9600

session modify
```

3. Opens the setup file "abc.pts" for working with specific PowerTerm parameters for the "abc" host (Similar to the menu File | Open). Then opens a telnet session to host "abc" (Similar to Communication | Connect).

```
open-setup-file abc.pts

set comm-type telnet

set host-name abc

session open
```

4. Opens a lat session to host "abc" through "Digital's Pathworks".

```
set comm-type lat

set service-name abc

session open
```

5. Opens a lat session to host "abc" through "Novell's Netware for LAT".

```
set comm-type lat

set server-name NovellServerName

set service-name abc

session open
```

6. Closes the current session.

```
session close
```

Note: In order to automatically connect to a host, make the appropriate script. At the File | Properties in the program manager add the name of the script (with parameters) to the Command Line. Every time the icon will be activated, PowerTerm will run the script and open the session.

Consider the following script named "telnet.psl" to connect to a host with telnet protocol:

```
set comm-type telnet

set host-name $p1

session open
```

Enter in the File | Properties:

```
C:\PTW\PTW.EXE telnet.psl HostName
```

Every time you will click the icon, "telnet.psl" will execute and connect to HostName.

In this manner you can make several icon for automatic connection to all your organization computers.

set

Description

Set a new value to a PowerTerm parameter

Syntax

set parameter value [value2]

Notes

Sets a new value to one to the PowerTerm parameters.

set baud-rate value

Sets the serial communication baud rate to one of the following values:

300, 600, 1200, 1800, 2400, 3600, 4800, 7200,

9600, 14400, 19200, 38400, 57600, 115200.

set protocol-type value

Sets the serial communication protocol type to one of the following values: none, xonxoff, hardware.

set parity bits type

Sets the serial communication parity bits and type to one of the following values:

bits: 7, 8.

type: none, even, odd, mark, space.

set stop-bits value

Sets the serial communication stop bits to one of the following values: 1, 2.

set comm-type value

Sets the communication type to one of the following values:

com: Serial communication.

int14: Communication with BIOS interrupt 14.

lat: Network communication with Digital lat.

cterm: Serial communication with Digital cterm.

telnet: Network communication with TCPIP telnet.

nwlat: Network communication Netware for LAT.

set port-number value

Sets the serial communication port number to one of the following values: 1, 2, 3, 4.

set host-name string

Sets the telnet and cterm communication host name.

set service-name string

Sets the lat communication service name (may also specify a host name).

set user-name string

Sets the Netwate for LAT user name.

set server-name string

Sets the Netwate for LAT user name.

set session-name string

Sets the communication session name on the emulation's title bar.

Returns

This command returns an empty string.

split

Description

Split a string into a proper PSL list

Syntax

split string [splitChars]

Notes

Returns a list created by splitting string at each character that is in the splitChars argument. Each element of the result list will consist of the characters from string that lie between instances of the characters in splitChars. Empty list elements will be generated if string contains adjacent characters in splitChars, or if the first or last character of string is in splitChars. If splitChars is an empty string then each character of string becomes a separate element of the result list. SplitChars defaults to the standard white-space characters.

Returns

Returns the split string.

Example

```
split "comp.unix.misc" "."
```

returns "comp unix misc"

```
split "Hello world" ""
```

returns "H e l l o { } w o r l d"

string

Description

Manipulate strings

Syntax

sring option arg [arg ...]

Notes

Performs one of several string operations, depending on option. The legal options (which may be abbreviated) are:

sring compare string1 string2

Perform a character-by-character comparison of strings string1 and string2 in the same way as the C strcmp procedure. Return -1, 0, or 1, depending on whether string1 is lexicographically less than, equal to, or greater than string2.

sring first string1 string2

Search string2 for a sequence of characters that exactly match the characters in string1. If found, return the index of the first character in the first such match within string2. If not found, return -1.

sring index string charIndex

Returns the charIndex'th character of the string argument. A charIndex of 0 corresponds to the first character of the string. If charIndex is less than 0 or greater than or equal to the length of the string then an empty string is returned.

sring last string1 string2

Search string2 for a sequence of characters that exactly match the characters in string1. If found, return the index of the first character in the last such match within string2. If there is no match, then return -1.

"**string last** ab 123abc456ab12", will return 9.

sring length string

Returns a decimal string giving the number of characters in string.

sring match pattern string

See if pattern matches string, return 1 if it does, 0 if it doesn't. For the two strings to match, their contents must be identical except that the following special sequences may appear in pattern:

*

Matches any sequence of characters in string, including a null string.

?

Matches any single character in string.

[chars]

Matches any character in the set given by chars. If a sequence of the form x-y appears in chars, then any character between x and y, inclusive, will match.

\x

Matches the single character x. This provides a way of avoiding the special interpretation of the characters *?[]\e in pattern.

sring range string first last

Returns a range of consecutive characters from string, starting with the character whose index is first and ending with the character whose index is last. An index of 0 refers to the first character of the string. Last may be **end** to refer to the last character of the string. If first is less than zero then it is treated as if it were zero, and if last is greater than or equal to the length of the string then it is treated as if it were **end**. If first is greater than last then an empty string is returned.

sring tolower string

Returns a value equal to string except that all upper case letters have been converted to lower case.

sring toupper string

Returns a value equal to string except that all lower case letters have been converted to upper case.

string trim string [chars]

Returns a value equal to string except that any leading or trailing characters from the set given by chars are removed.

If chars is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

sring trimleft string [chars]

Returns a value equal to string except that any leading characters from the set given by chars are removed. If chars is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

sring trimright string [chars]

Returns a value equal to string except that any trailing characters from the set given by chars are removed. If chars is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

Returns

Returns the converted string.

Example

Returns "UNIX".

```
string toupper Unix
```

switch

Description

Evaluate one of several scripts, depending on a given value.

Syntax

switch [options] string [pattern body [pattern body ...]]

or

switch [options] string {pattern body [pattern body...]}

Notes

The **switch** command matches its string argument against each of the pattern arguments in order.

As soon as it finds a pattern that matches string it evaluates the following body argument by passing it recursively to the PSL interpreter and returns the result of that evaluation.

If the last pattern argument is default then it matches anything.

If no pattern argument matches string and no default is given, then the switch command returns an empty string.

If the initial arguments to **switch** start with "-" then they are treated as options. The following options are currently supported:

-exact

Use exact matching when comparing string to a pattern. This is the default.

-glob

When matching string to the patterns, use glob-style matching (i.e. the same as implemented by the **string match** command).

-regexp

When matching string to the patterns, use regular expression matching (i.e. the same as implemented by the **regexp** command).

--

Marks the end of options. The argument following this one will be treated as string even if it starts with a "-".

Two syntaxes are provided for the pattern and body arguments.

The first uses a separate argument for each of the patterns and commands. This form is convenient if substitutions are desired on some of the patterns or commands.

The second form places all of the patterns and commands together into a single argument. The argument must have proper list structure, with the elements of the list being the patterns and commands.

The second form makes it easy to construct multi-line switch commands, since the braces around the whole list make it unnecessary to include a backslash at the end of each line.

Since the pattern arguments are in braces in the second form, no command or variable substitutions are performed on them. This makes the behavior of the second form different than the first form in some cases.

If a body is specified as "-'' it means that the body for the next pattern should also be used as the body for this pattern (if the next pattern also has a body of "-" then the body after that is used, and so on). This feature makes it possible to share a single body among several patterns.

Returns

Returns the output of the last command of the executed in body.

Example

```
switch abc a-b {format 1} abc {format 2} default {format 3}
```

will return "2".

```
switch -regexp aaab {
        ^a.*b$      -
        b                       {format 1}
        a*                      {format 2}
```

```
        default     {format 3}
}
```

will return "1".

```
switch xyz {
        a                       -
        b                       {format 1}
        a*                      {format 2}
        default     {format 3}
}
```

will return "3".

tell

Description

Return current access position for an open file

Syntax

tell fileId

Notes

Returns a decimal string giving the current access position in fileId.

FileId must have been the return value from a previous call to open.

Returns

Returns the file position.

Example

Opens a file and Reads twice 10 bytes. **tell** will return "20".

```
fileId = [open data"]
read $fileId 10
read $fileId 10
position = [tell $fileId]
close $fileId
```

time

Description

Time the execution of a script

Syntax

time scriptCommand count

Notes

This command will call the PSL interpreter count times to evaluate scriptCommand (or once if count isn't specified). It will then return a string of the form "503 microseconds per iteration" which indicates the average amount of time required per iteration, in microseconds.

Time is measured in elapsed time, not CPU time.

Returns

Returns the elapsed time off a PSL command.

Example

Displays the average elapsed time of "**run** test.psl" command.

```
message [time "run test.ppl" 100]
```

unset

Description

Delete variables

Syntax

unset name [name ...]

Notes

This command removes one or more variables.

Each name is a variable name, specified in any of the ways acceptable to variable assignment.

If a name refers to an element of an array then that element is removed without affecting the rest of the array.

If a name consists of an array name with no parenthesized index, then the entire array is deleted.

An error occurs if any of the variables doesn't exist, and any variables after the non-existent one are not deleted.

Returns

Returns an empty string.

Example

Sets and than deletes varibale a.

```
a = abc
unset a
```

uplevel

Description

Execute a script in a different stack frame.

Syntax

uplevel [level] arg [arg ...]

Notes

All of the arg arguments are concatenated as if they had been passed to concat. The result is then evaluated in the variable context indicated by level. **uplevel** returns the result of that evaluation.

If level is an integer then it gives a distance (up the procedure calling stack) to move before executing the command. If level consists of "#" followed by a number then the number gives an absolute level number. If level is omitted then it defaults to 1. Level cannot be defaulted if the first command argument starts with a digit or "#".

Returns

Returns the result of the evaluation.

Example

Suppose that procedure a was invoked from top-level, and that it called b, and that b called c.

Suppose that c invokes the **uplevel** command. If level is 1 or #2 or omitted, then the command will be executed in the variable context of b. If level is 2 or #1 then the command will be executed in the variable context of a.

If level is 3 or #0 then the command will be executed at top-level (only global variables will be visible).

The **uplevel** command causes the invoking procedure to disappear from the procedure calling stack while the command is being executed.

In the above example, suppose c invokes the command

uplevel 1 {x = 43; d}

where d is another PSL procedure. The "x = 43" command will modify the variable x in b's context, and d will execute at level 3, as if called from b. If it in turn executes the command

uplevel {x = 42}

then the "x = 42" command will modify the same variable x in b's context: the procedure c does not appear to be on the call stack when d is executing. The command "**info level**" may be used to obtain the level of the current procedure.

uplevel makes it possible to implement new control constructs as PSL procedures (for example, **uplevel** could be used to implement the **while** construct as a PSL procedure).

upvar

Description

Create link to variable in a different stack frame.

Syntax

upvar [level] otherVar myVar [otherVar myVar ...]

Notes

This command arranges for one or more local variables in the current procedure to refer to variables in an enclosing procedure call or to global variables.

Level may have any of the forms permitted for the **uplevel** command, and may be omitted if the first letter of the first otherVar isn't "#" or a digit (it defaults to 1).

For each otherVar argument, **upvar** makes the variable by that name in the procedure frame given by level (or at global level, if level is #0 accessible in the current procedure) by the name given in the corresponding myVar argument.

The variable named by otherVar need not exist at the time of the call. It will be created the first time myVar is referenced, just like an ordinary variable.

upvar may only be invoked from within procedures.

MyVar may not refer to an element of an array, but otherVar may refer to an array element.

The upvar command simplifies the implementation of call-by-name procedure calling and also makes it easier to build new control constructs as PSL procedures.

Returns

Returns an empty string.

Example

Consider the following procedure:

```
proc add2 name {
  upvar $name x
  x = [expr $x+2]
}
```

Add2 is invoked with an argument giving the name of a variable, and it adds two to the value of that variable.

Although add2 could have been implemented using **uplevel** instead of **upvar**, **upvar** makes it simpler for add2 to access the variable in the caller's procedure frame.

If an **upvar** variable is **unset** (e.g. x in add2 above), the **unset** operation affects the variable it is linked to, not the **upvar** variable. There is no way to **unset** an **upvar** variable except by exiting the procedure in which it is defined. However, it is possible to retarget an **upvar** variable by executing another **upvar** command.

wait

Description

Wait for specific strings received from the host.

Syntax

wait option [value]

Notes

PowerTerm has the ability to check concurrently few strings received from the host.

When one is received, the variable result will contain its value and a script will be executes.

If timeout occurs before one of the strings received the result variable will be empty.

wait string string

Sets the next string.

wait script string

Sets the script name to be executed when a string is received.

wait timeout number

Sets timeout to number of seconds.

wait reset

Reset the all previous wait string commands and stops pending wait.

wait start

Start checking strings received from the host.

Example

The following script causes to wait for two strings "login" and "Password" during the login process. When one is received the script wait.psl will be executed.

```
wait string login
wait string Password
wait script wait.psl
wait start
```

Checks the received string and sends "username" and "password" to the host. After the user name is sent, waits for another string.

Note: the "^M" string is control-m on the keyboard.

```
switch $result {
        login                   { send "username^M" ; wait start }
        Password        { send "password^M" }
}
```

while

Description

Execute script repeatedly as long as a condition is met.

Syntax

while test body

Notes

The **while** command evaluates test as an expression (in the same way that the expr command evaluates its argument).

The value of the expression must a proper boolean value. If it is a true value then body is executed by passing it to the PSL.

Once body has been executed then test is evaluated again, and the process repeats until eventually test evaluates to a false boolean value. **continue** commands may be executed inside body to terminate the current iteration of the loop, and **break** commands may be executed inside body to cause immediate termination of the while command.

Returns

Returns an empty string.

Example

Displays "yes" while variable host it equal to "vax".

```
while {$host == vax} {
        message yes
        commands...
}
```

window

Description

Change the emulation window status.

Syntax

window option [args...]

Notes

Performs one of several window operations, depending on option. The legal options are:

window [maximize | minimize | restore | hide | show]

Changes the emulation window accordingly.

window size **height width**

Changes the height and width (in pixels) of the window.

window position **y x**

Changes the x and y coordinates (in pixels) of the window.

Returns

Returns an empty string.

Example

Restores the window (may be in maximize) and sets its position and size.

```
window restore
window position 50 50
window size 400 600
```