1

INTRODUCTION

What is Perspective?

Description

Perspective is a program that makes computers as easy to use as filing cabinets. All your data is stored on printable, what-you-see-is-what-you-get pages. Pages are filed in folders, which are kept in drawers. You can draw on pages. You can type on pages. You can even put formulas on pages. And you can process pages like records in a file with the built-in programming language.

Applications

We run our entire business, our home school, and our local church using Perspective. You can, too. When we process an order or an invoice, we do it with Perspective. All of our training materials, documentation, packaging, newsletters, flyers — even our business cards are produced using this program. When you register for our classes, we use Perspective to sign you up, print a map, and send the confirmation. Our mailing lists, labels, and envelopes are all products of Perspective. This is the program we use for everything from financial reports to flash cards.

Hardware and Software

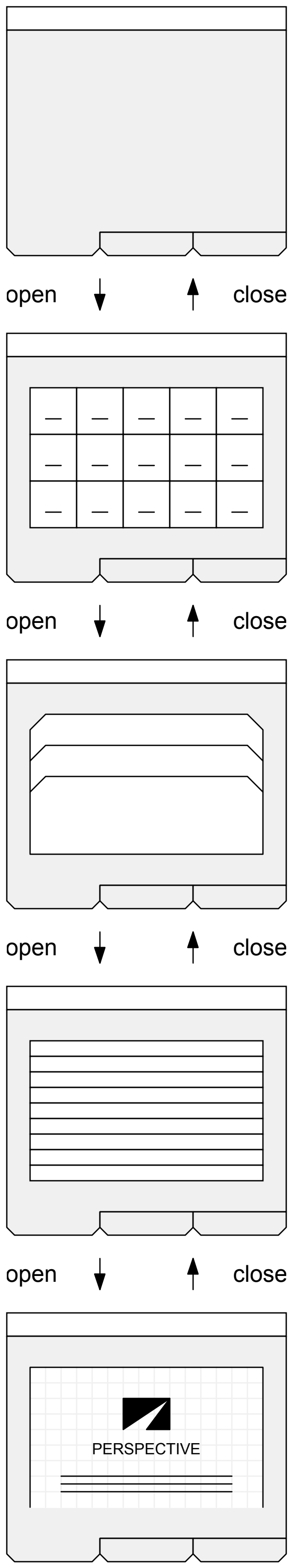
Perspective works on any computer running Windows 95, but uses very little of Windows and looks nothing like Windows in operation. This means that your Perspective system will stay the same even when you-know-who decides to mess with the operating system again.

Prerequisites

Perspective is easy to understand, easy to use. You do, however, need to know a bit about Windows to install it. And if you plan to write your own programs, some previous programming experience is helpful. You do not have to know how to spell — the spelling checker is built in.

License and Distribution

Ownership of this package entitles you to use Perspective at a single site — a home, a church, a classroom, an office. This copy may be shared by all users on a single local area network.



AREA VIEW

Perspective's work areas are initially blank at startup.

When you open a cabinet you enter...

CABINET VIEW

... and all of the drawers in that cabinet are shown.

When you open a drawer you enter...

DRAWER VIEW

... and all of the folders in that drawer are shown.

When you open a folder you enter...

FOLDER VIEW

... and all of the pages in that folder are shown.

When you open a page you enter...

PAGE VIEW

... and all of the shapes on that page are shown. Pages can contain graphics, text, and calculated values.

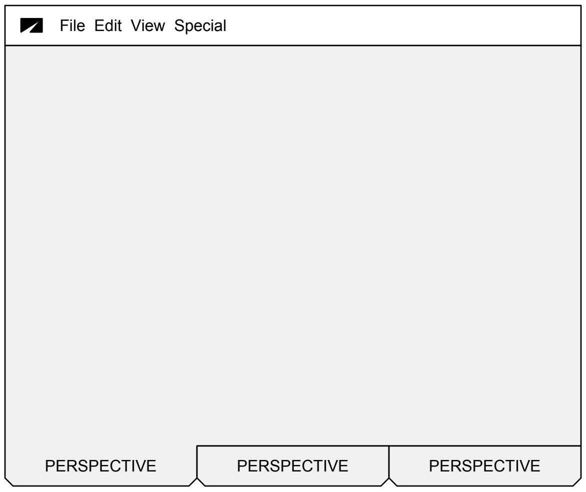
The User Interface

Overview

The Perspective interface is intentionally plain and simple. All unnecessary components have been eliminated to minimize distractions and to leave most of the screen available for your work.

The Desktop

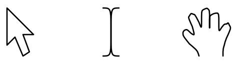
The desktop consists of a single menu bar and three overlapping work areas, as shown here:



The desktop automatically fills the entire screen, and cannot be moved or sized in any way.

The Cursors

When you push the mouse around, a small symbol moves, correspondingly, on the desktop. This small symbol is the cursor. Perspective uses three distinct cursor shapes:



The ARROW cursor is used for general purpose pointing and clicking. The I-BEAM cursor is used when editing text. The HAND appears whenever you press the right mouse button over an object that can be scrolled (moved to another position).

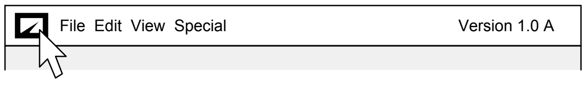
The cursor disappears when Perspective is busy.

The Menu Bar

Perspective uses the menu bar for a variety of purposes: displaying the current program version, displaying status and error messages, executing commands, and getting additional input.

The Program Version

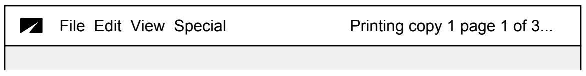
You can display the version of Perspective that is currently running by clicking the LOGO at the far left of the menu bar:



The number preceding the decimal point is the version; the number following indicates minor improvements and/or bug fixes. The letter at the end is the file format — all versions with the same letter can read and write each other's files.

Messages

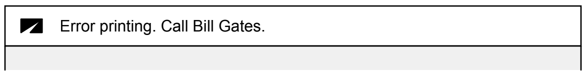
All non-critical messages are displayed at the far right of the menu bar, like this:



Printing copy 1 page 1 of 3...

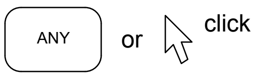
Errors

Perspective flashes the LOGO and emits a short beep or click when an error occurs.



Error printing. Call Bill Gates.

Error messages are displayed in red, temporarily erasing the menus, as shown above.

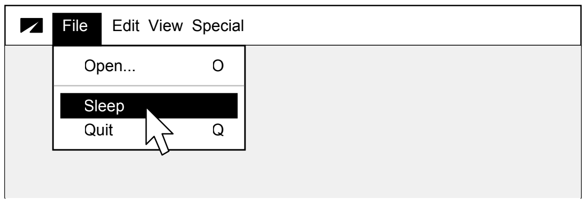


You can refresh the menus and continue working by pressing any key or clicking the mouse.

The User Interface

Commands

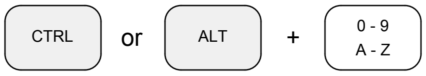
You can execute any command by pressing on the name of a menu with the mouse, dragging down to the desired command, and letting go.



Perspective's menus are spring-loaded — if you pull down a menu but let go when the cursor is not over a command, the menu simply disappears.

Shortcut Keys

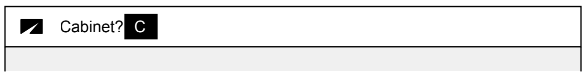
Menu commands with a letter at the far right can be executed from the keyboard by holding either CTRL or ALT and pressing the indicated key.



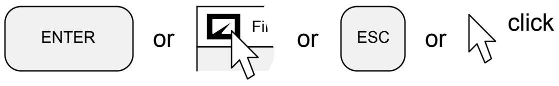
Most people find the ALT key more convenient.

Additional Input

Menu commands followed by three dots require further input. Perspective asks you questions...



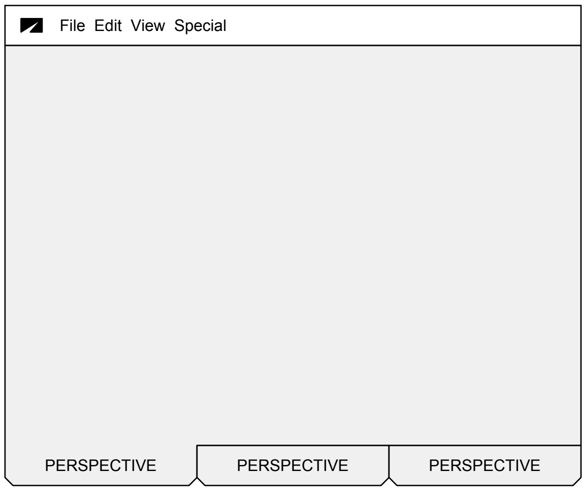
You respond via the keyboard. Text is edited as described later in this chapter. Standard shortcut keys can also be used to cut, copy, and paste.



You can press ENTER or click the LOGO to continue with the command. To cancel, just press ESC or click anywhere outside the menu bar.

The Work Areas

The work areas occupy the bulk of the screen, giving you lots of room to work. Your cabinets, drawers, folders, and pages appear here:

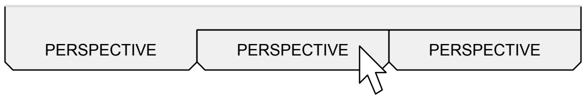


At any time, only one work area is fully visible; the tabs of the others can, however, be seen at the bottom of the screen. The leftmost work area is active in the illustration above.

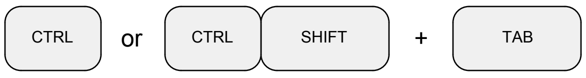
The work areas are independent of one another, so you can work on up to three different tasks at the same time.

Switching Work Areas

You can switch work areas by clicking the tabs at the bottom of the screen:



You can also switch work areas via the keyboard:

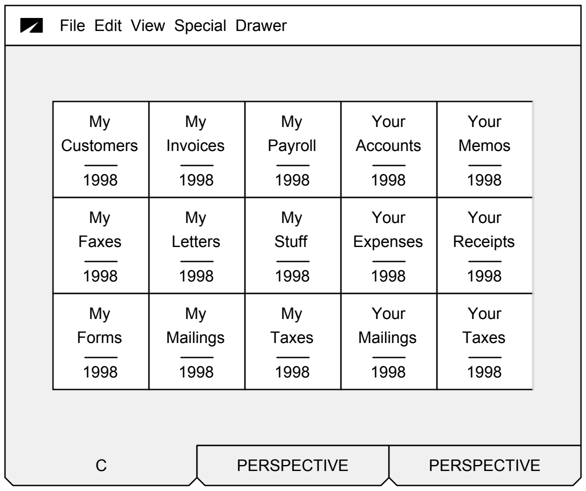


Holding CTRL and pressing TAB will move you to the right; holding CTRL with SHIFT and pressing TAB will move you to the left.

The Views

Cabinet View

A cabinet is a collection of drawers that maps to a physical or logical disk drive. Cabinets are usually identified by a letter between A and Z.



My Customers 1998	My Invoices 1998	My Payroll 1998	Your Accounts 1998	Your Memos 1998
My Faxes 1998	My Letters 1998	My Stuff 1998	Your Expenses 1998	Your Receipts 1998
My Forms 1998	My Mailings 1998	My Taxes 1998	Your Mailings 1998	Your Taxes 1998

When you open a cabinet, all of its drawers are displayed in the work area. The cabinet name appears in the tab at the bottom of the screen.

Each drawer has a group, a name, and a version. The group is on top; the version is on the bottom.

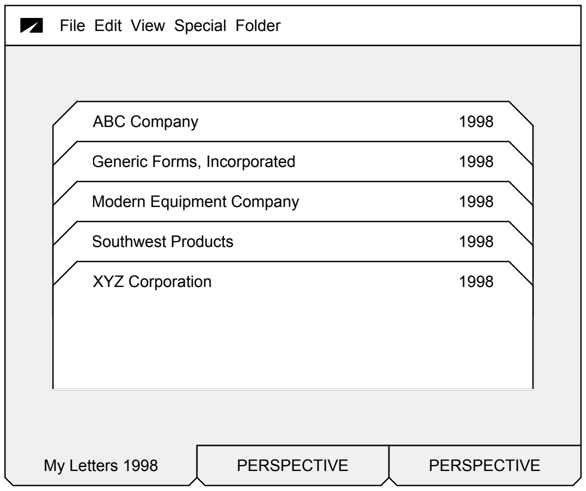
In Cabinet View, You Can...

- Select one or more drawers
- Scroll drawers left and right
- Sort drawers by group, name, and/or version
- Search for drawers by group, name, or version
- Add new drawers
- Rename existing drawers
- Delete obsolete drawers
- Copy, paste, and duplicate drawers

You can also close a cabinet to return to Area View, or open a drawer, which takes you to...

Drawer View

A drawer is a collection of folders. You can have as many drawers as you need, and you can store any number of folders in each drawer.



ABC Company	1998
Generic Forms, Incorporated	1998
Modern Equipment Company	1998
Southwest Products	1998
XYZ Corporation	1998

When you open a drawer, all of its folders are displayed in the work area. The drawer name appears in the tab at the bottom of the screen.

Each folder has a name and a version. The name appears at the left; the version at the far right.

In Drawer View, You Can...

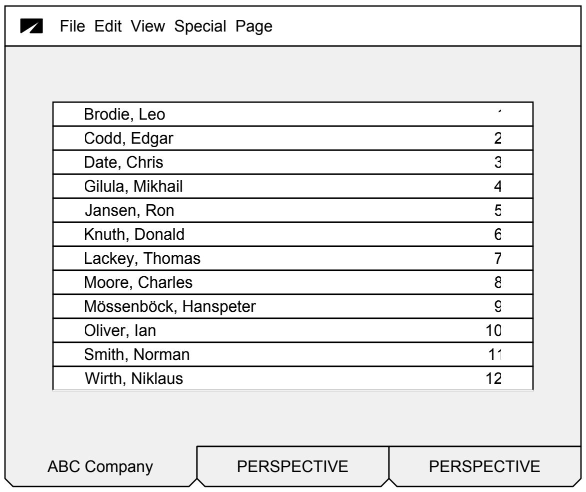
- Select one or more folders
- Scroll folders up and down
- Sort folders by name and/or version
- Search for folders by name or version
- Add new folders
- Rename existing folders
- Delete obsolete folders
- Copy, paste, duplicate, and backup folders

You can also close a drawer to return to Cabinet View, or open a folder, which takes you to...

The Views

Folder View

A folder is a collection of pages. You can store any number of pages in each folder. Folders are typically used to hold between 10 and 250 pages.



Brodie, Leo	1
Codd, Edgar	2
Date, Chris	3
Gilula, Mikhail	4
Jansen, Ron	5
Knuth, Donald	6
Lackey, Thomas	7
Moore, Charles	8
Mössenböck, Hanspeter	9
Oliver, Ian	10
Smith, Norman	11
Wirth, Niklaus	12

When you open a folder, the top or bottom edges of all of its pages are displayed. The folder name appears in the tab at the bottom of the screen.

Top edges contain user data like names and dates. Bottom edges are used by programmers.

In Folder View, You Can...

• Select one or more pages

• Scroll pages up and down

• Sort pages by the values on their edges

• Search for pages by any value

• Add new pages

• Delete obsolete pages

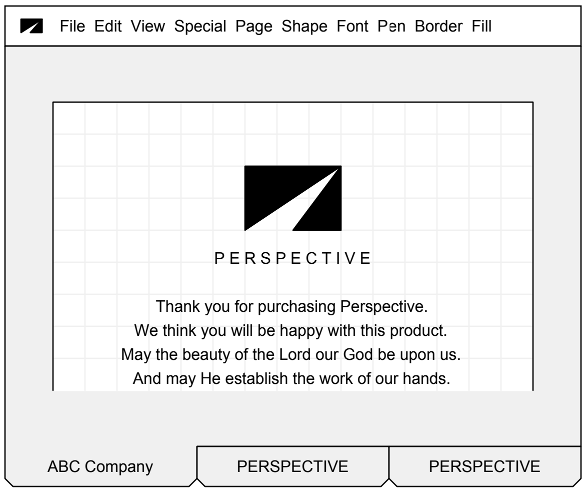
• Cut, copy, paste, duplicate, and number pages

• Print one or more pages or their edges

You can also close a folder to return to Drawer View, or open a page, which takes you to...

Page View

A page is a collection of graphic and text shapes. You can have any number of shapes on a page. Pages can be drawings, forms, or functions.



Thank you for purchasing Perspective.
We think you will be happy with this product.
May the beauty of the Lord our God be upon us.
And may He establish the work of our hands.

When you open a page, the full page and all the shapes on it are shown. The folder name remains in the tab at the bottom of the screen.

Pages can be resized in half-inch increments. Minimum size is 2 x 2; maximum size is 11 x 11.

In Page View, You Can...

• Add, change, and delete shapes on drawings

• Enter, update, and erase data on forms

• Write programs on function pages

• Enlarge, reduce, and scroll any page

• Spell check any page

• Undo and redo the last ten changes to any page

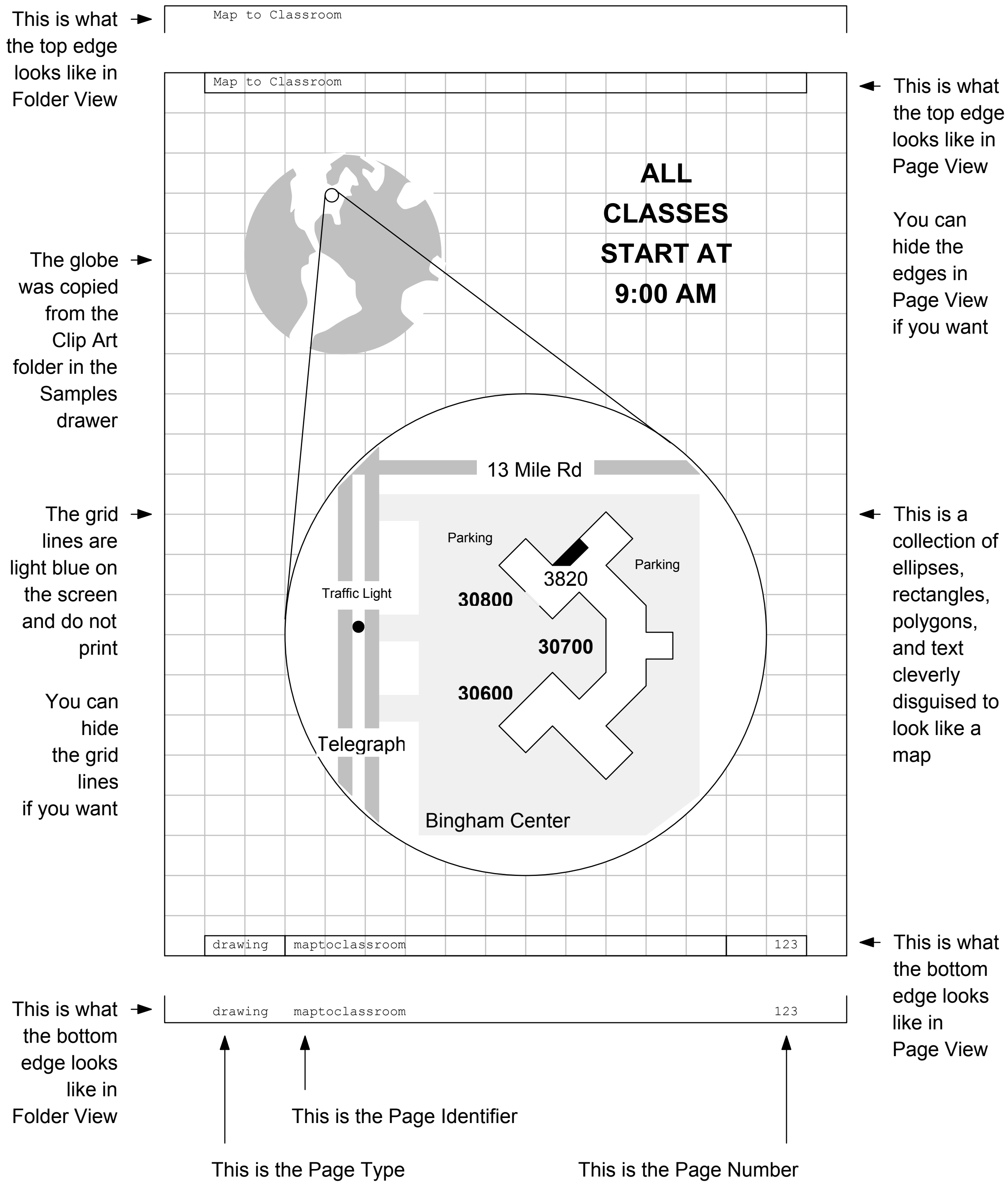
• Cut, copy, paste, and duplicate on any page

• Print one or more copies of any page

You can also flip thru pages (in either direction), or close a page to return to Folder View.

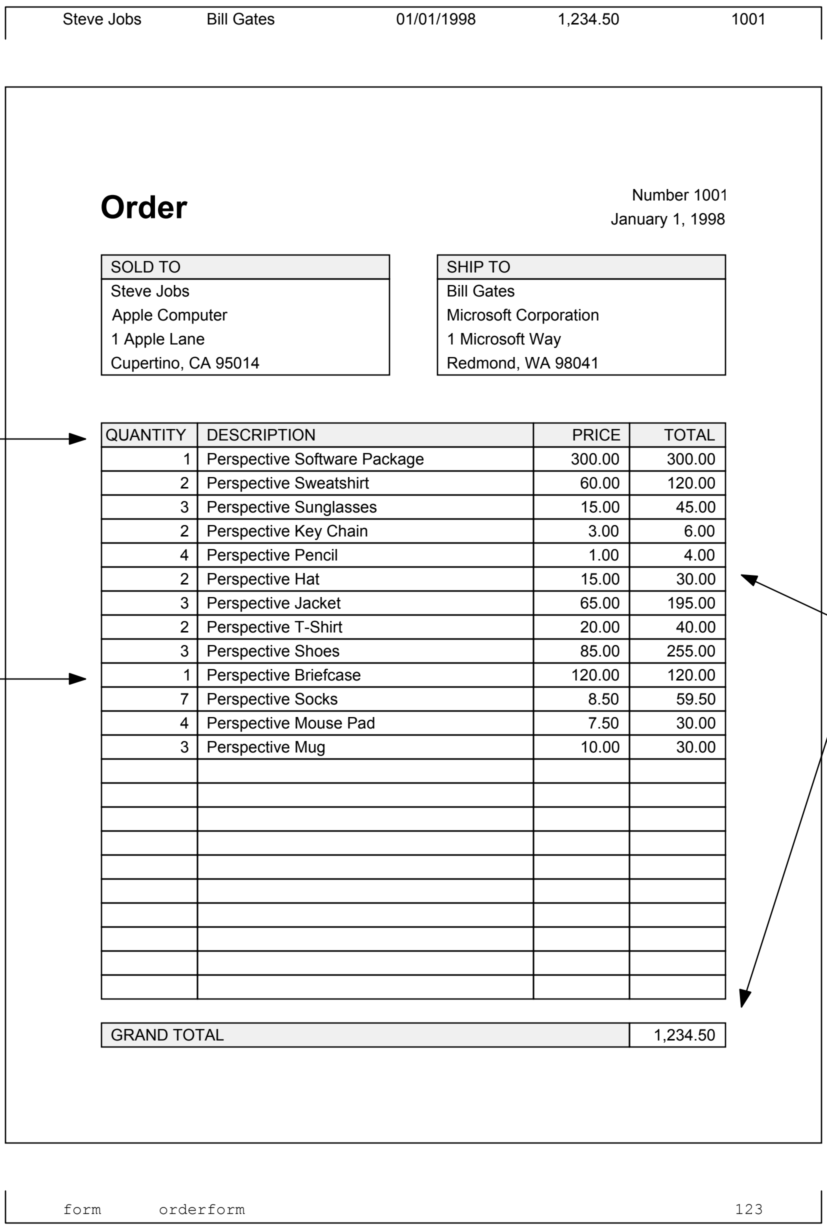
Drawings

Drawings are general-purpose pages. You can combine graphic and text shapes any way you like on a drawing page. We use drawing pages for all of our training materials, documentation, packaging, newsletters, flyers — even our business cards. Drawing pages are fully WYSIWYG and can be black-and-white or color.


This is what the top edge looks like in Folder View
Map to Classroom
Map to Classroom
This is what the top edge looks like in Page View
You can hide the edges in Page View if you want
ALL CLASSES START AT 9:00 AM
The globe was copied from the Clip Art folder in the Samples drawer
13 Mile Rd
Parking
3820
Parking
Traffic Light
30800
30700
30600
Telegraph
Bingham Center
The grid lines are light blue on the screen and do not print
You can hide the grid lines if you want
This is a collection of ellipses, rectangles, polygons, and text cleverly disguised to look like a map
drawing
maptoclassroom
123
This is what the bottom edge looks like in Page View
This is what the bottom edge looks like in Folder View
drawing
maptoclassroom
123
This is the Page Identifier
This is the Page Type
This is the Page Number

Forms

Forms are printable pages that contain both editable and non-editable shapes. On the form shown below, for example, the headings and calculated values are not editable and act as background elements of the form. We use forms for orders and registrations, for confirmations and invoices, and for a wide variety of reports.



This is the top edge in Folder View

Steve Jobs	Bill Gates	01/01/1998	1,234.50	1001

Top edges do not show on forms in Page View

Edge values are taken from the body of the page

Order

Number 1001
January 1, 1998

SOLD TO	SHIP TO
Steve Jobs	Bill Gates
Apple Computer	Microsoft Corporation
1 Apple Lane	1 Microsoft Way
Cupertino, CA 95014	Redmond, WA 98041

Labels and other background items are not editable

QUANTITY	DESCRIPTION	PRICE	TOTAL
1	Perspective Software Package	300.00	300.00
2	Perspective Sweatshirt	60.00	120.00
3	Perspective Sunglasses	15.00	45.00
2	Perspective Key Chain	3.00	6.00
4	Perspective Pencil	1.00	4.00
2	Perspective Hat	15.00	30.00
3	Perspective Jacket	65.00	195.00
2	Perspective T-Shirt	20.00	40.00
3	Perspective Shoes	85.00	255.00
1	Perspective Briefcase	120.00	120.00
7	Perspective Socks	8.50	59.50
4	Perspective Mouse Pad	7.50	30.00
3	Perspective Mug	10.00	30.00

GRAND TOTAL	1,234.50

Data items in the foreground are editable

Totals are calculated based on formulas hidden on the page and are not editable

Bottom edges do not show on forms in Page View

This is what the bottom edge looks like in Folder View

```
form      orderform                                          123
```

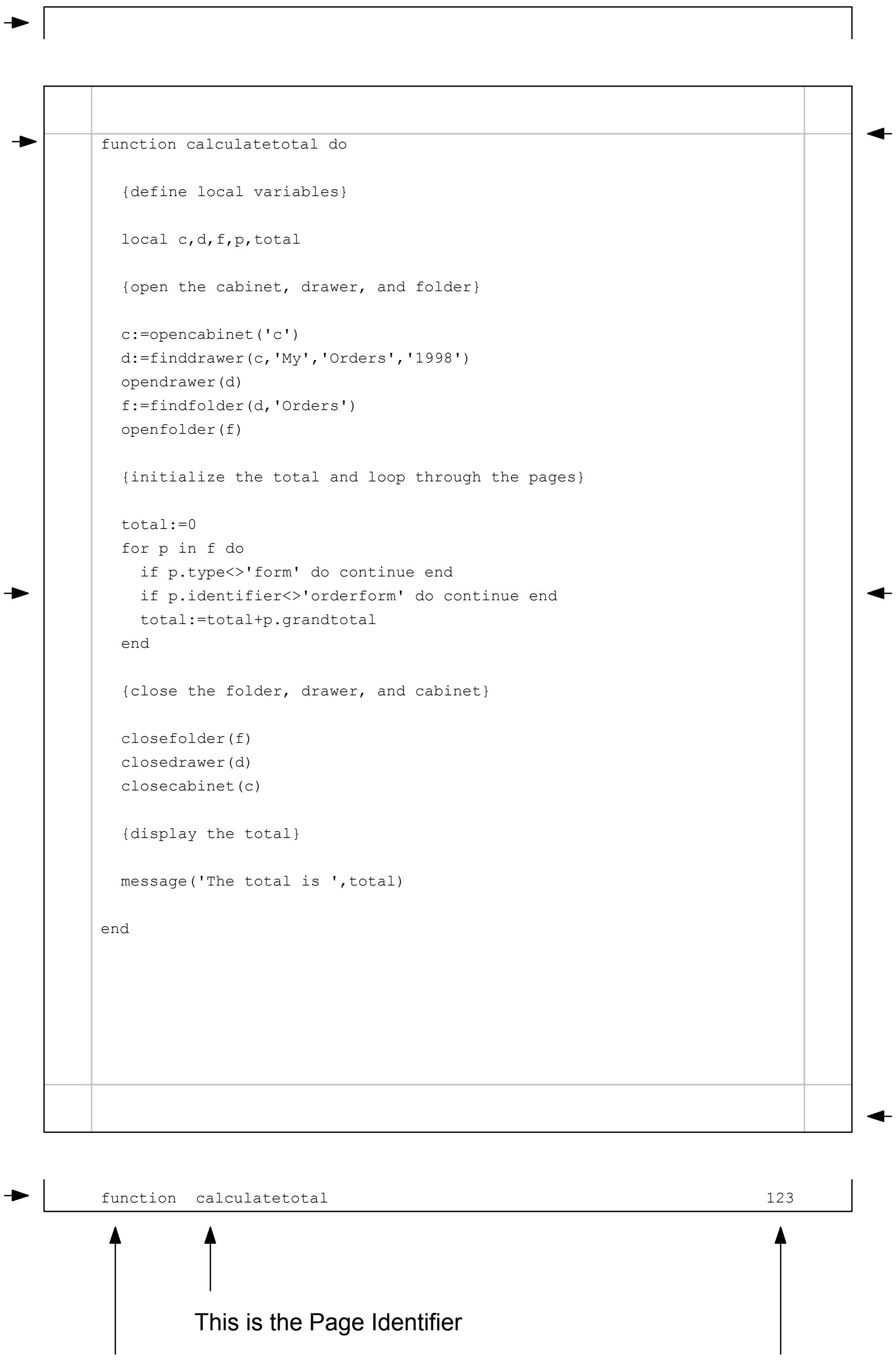
This is the Page Identifier

This is the Page Type

This is the Page Number

Functions

Functions are pages that can be compiled and executed by Perspective's built-in language processor. The function below calculates the total for all orders in a particular folder and displays it on the screen. We use functions to produce mailing labels, to tally financial reports, and for other purposes too numerous to mention.



Top edges are blank

Function names are automatically extracted from the code and copied to the bottom edge of the page

The grid can be hidden and does not print

```
function calculatetotal do

  {define local variables}

  local c,d,f,p,total

  {open the cabinet, drawer, and folder}

  c:=opencabinet('c')
  d:=finddrawer(c,'My','Orders','1998')
  opendrawer(d)
  f:=findfolder(d,'Orders')
  openfolder(f)

  {initialize the total and loop through the pages}

  total:=0
  for p in f do
    if p.type<>'form' do continue end
    if p.identifier<>'orderform' do continue end
    total:=total+p.grandtotal
  end

  {close the folder, drawer, and cabinet}

  closefolder(f)
  closedrawer(d)
  closecabinet(c)

  {display the total}

  message('The total is ',total)

end
```

Code automatically indents

Keywords appear in red and comments appear in blue

You can choose to code in a variety of font sizes and styles

Bottom edges do not show on functions in Page View

```
function  calculatetotal                                   123
```

This is what the bottom edge looks like in Folder View

This is the Page Identifier

This is the Page Type

This is the Page Number

Installation

Before You Begin...

There is no installation program supplied with Perspective, so you won't have to spend $39 for some other program to help you remove it.

These instructions do, however, assume some familiarity with Windows. If you are not comfortable with terms like folder, shortcut, and command line, consult the documentation that came with your computer before going further.

Installing on a Single Machine

To install Perspective, simply drag the PERSPECTIVE folder to your hard disk. This folder must be kept at the root level. Do not change its name. The program, supporting files, and all your drawers, folders, and pages are stored in this folder.

Installing on a Network

Perspective should be installed on only one computer on a network. If you have a server, install it on that machine; otherwise, pick a machine that will be running all the time.

To install Perspective, simply drag the PERSPECTIVE folder to the hard disk of the designated machine. This folder must be kept at the root level. Do not change its name. The program (prspctv.exe), the lexicon (for the spelling checker), the lock and configuration files, indexes, and drawers, folders, and pages are stored here. Drawers, folders, pages, and additional indexes and lock files may also be stored elsewhere on the network.

Configuring a Network

If you use the default configuration file supplied with Perspective, each disk accessible to a user will behave as a separate cabinet and will be identified by drive letter (such as A or C). All printing will take place on the default printer assigned to that machine in Windows.

See the appendix entitled 'The Configuration File' for further info on customizing your configuration.

Re-Installing Perspective

DO NOT re-install Perspective over an existing installation — ALL DATA WILL BE LOST. You can, however, replace the program (prspctv.exe), the lexicon, and the configuration file without damaging your drawers, folders, and pages. The lexicon reverts to its original form when replaced.

Upgrading Perspective

Upgrades are shipped with special installation instructions. Follow the instructions that are supplied with your upgrade.

Removing from a Single Machine

To remove Perspective from a single machine, delete the PERSPECTIVE folder. All programs and data will be erased.

Removing from a Network

To remove Perspective from a network, you must delete all PERSPECTIVE folders from all machines. These folders will exist at the root level on the original server, the root level of any disk accessed from Perspective, and in any other location designated as a cabinet in the configuration file.

Getting Help

Problems can be reported to us via fax at:

`248-646-5493`

or via the internet:

`www.prspctv.com`
`help@prspctv.com`

Be sure to include both voice and fax numbers for our reply. If you're really stuck and need help right away, call us at:

`248-646-6901`

Don't contact us unless you have at least tried to find the answer to your question in this book.

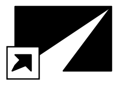
Getting Started

Starting Up

You can start Perspective in a number of ways. The most convenient ones are described below.

Using a Shortcut

Create a shortcut referencing prspctv.exe and leave it on the desktop. For a variety of reasons, this is the recommended method.



You can give the shortcut any name you want. You can also specify a startup cabinet for each tab — just list the cabinet names, separated by spaces, on the command line. Double-click the shortcut to begin.

Direct Access

You can also start Perspective by opening the PERSPECTIVE folder and double-clicking the prspctv.exe file directly.

This method is acceptable for occasional use, but is not as handy as using a shortcut. It also does not allow startup cabinets.

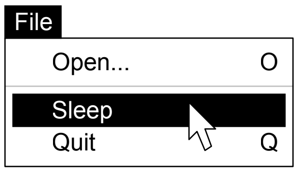
Using the Start Menu

Add Perspective to the Windows 95 Start Menu. To start up, select the item from the menu.

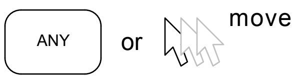
This method is not recommended because the collection of hierarchical menus on the Start menu is much too extensive already.

Sleeping

You can put Perspective to sleep by selecting the Sleep command from the File menu.



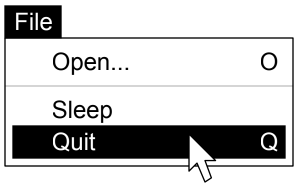
The screen saver will also automatically activate if there is no activity for 15 consecutive minutes. When Perspective is running, our screen saver overrides any other that may be installed.



Press any key or move the mouse to continue.

Shutting Down

To exit Perspective, select the Quit command from the File menu.



You can quit from any view, and do not need to close open cabinets, drawers, folders, and pages to do so. All changes are automatically saved.

Helpful Hints

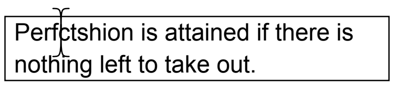
• Computers last longer when you leave them on all the time. The stress on electronic circuitry at startup is much more likely to do damage than the wear-and-tear of constant use. If you'll be back within 3 to 5 days, leave the machine on.

• Don't quit Perspective when you reach a breaking point; just put it to sleep. This way, you won't have to look at Windows when you return.

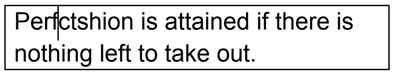
Editing Text

Getting Started

Editable text can be found in dialogs and on pages. The I-BEAM cursor is used when editing text. It automatically appears whenever the mouse is over text that you can modify.


Perfctshion is attained if there is
nothing left to take out.

To edit text, simply position the I-BEAM at the appropriate place, and click. A small blinking vertical bar, called the insertion point, will appear:


Perfctshion is attained if there is
nothing left to take out.

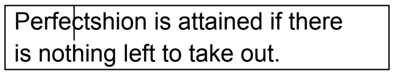
You can use the HOME, END, and ARROW keys to move the insertion point within text. You can also click again in another location to move greater distances.

HOME END ← → ↑ ↓

The HOME key moves the insertion point to the beginning of the current line; the END key moves it to the end of the line. The ARROW keys move one character or line at a time. You can hold CTRL or ALT while arrowing left or right to move a word (instead of a character) at a time.

Inserting Text

You can add characters at the insertion point simply by typing. The letters to the right will move to make room for the new ones, wrapping around to the next line if necessary.


Perfectshion is attained if there
is nothing left to take out.

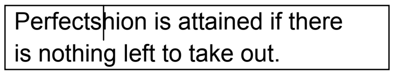
When the letter 'e' is added to the first word at the beginning of this sentence, for example, the word 'is' (originally at the end of the first line) moves to the second line.

Deleting Text

You can delete characters in two ways, using either the BACKSPACE or the DELETE key.

BACKSPACE or DEL

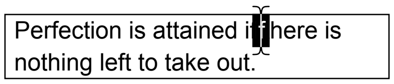
The BACKSPACE key removes the character to the left of the insertion point; the DELETE key removes the character to the right.


Perfectshion is attained if there
is nothing left to take out.

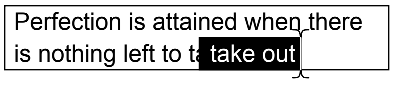
With the insertion point positioned as shown above, the BACKSPACE key will remove the 's', while the DELETE key will remove the 'h'.

Working with Words and Phrases

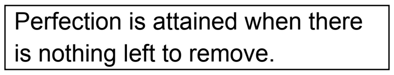
You can select a word by double-clicking on it. The entire word is highlighted and will be replaced with whatever you type next.


Perfection is attained if there is
nothing left to take out.

You can select larger portions of text by dragging the I-BEAM across the desired section.


Perfection is attained when there
is nothing left to take out

The highlighted portion will be replaced with whatever you type:


Perfection is attained when there
is nothing left to remove.

You can also select to the left and right of the insertion point by holding the SHIFT key and pressing HOME, END, or the LEFT or RIGHT ARROW keys. If you hold SHIFT and either CTRL or ALT, the arrows will select whole words.

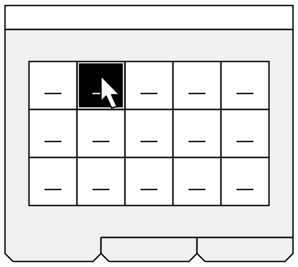
2

FILING

Selecting and Scrolling

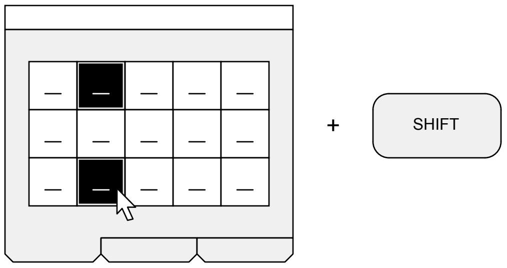
Simple Selection

A single drawer, folder, or page can be selected simply by clicking on it with the mouse.



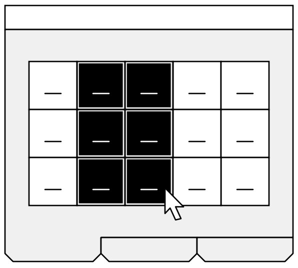
Selecting More or Less

You can add to or remove from a selection by holding the SHIFT key while clicking.



Selecting Groups

You can operate on groups of objects by dragging with the mouse, with or without the SHIFT key:



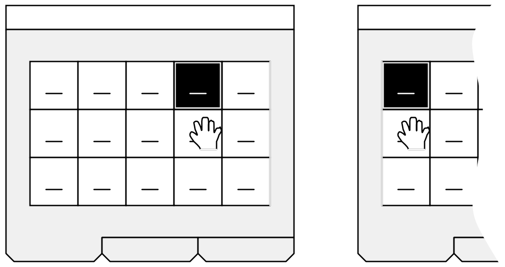
Selecting All or None

You can select everything using the Select All command on the Edit menu. You can select nothing by clicking in the work area.

Scrolling

Perspective eliminates the clutter of scroll bars by using the right mouse button for all scrolling. The HAND cursor indicates scrolling in progress:

Cabinets, for example, scroll left and right. Simply position the cursor over a drawer, press the right mouse button, and drag.

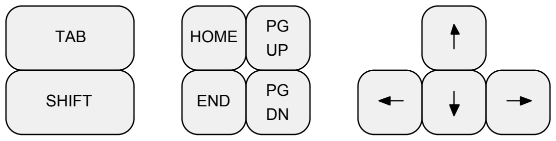


A missing black line at the edge indicates that there is more in that direction. Similar indications are provided for drawers, folders, and pages.

Scrolling Faster

You can scroll faster and further by pressing the SHIFT key while dragging with the mouse.

With the SHIFT key down, a single swipe across the screen will scroll from one end of an object to the other. You can 'tap' the SHIFT key while you are scrolling to make smaller jumps.

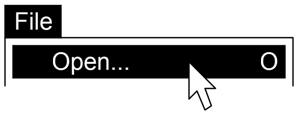


You also can use the keys above to move the selection on the screen. TAB moves right, SHIFT-TAB left. The others operate intuitively.

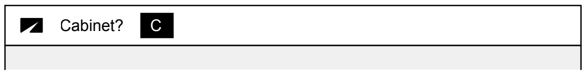
Opening and Closing

Opening and Closing Cabinets

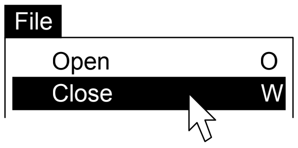
You open a cabinet with the Open command.



Perspective will ask you which cabinet to open:



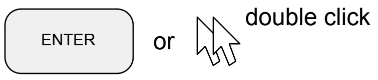
You close a cabinet with the Close command.



This is the only way to open and close cabinets.

Opening Drawers, Folders, and Pages

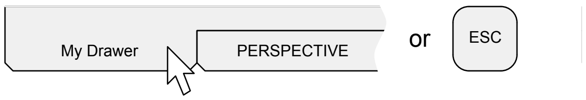
You can open a drawer, folder, or page by selecting one and executing the Open command.



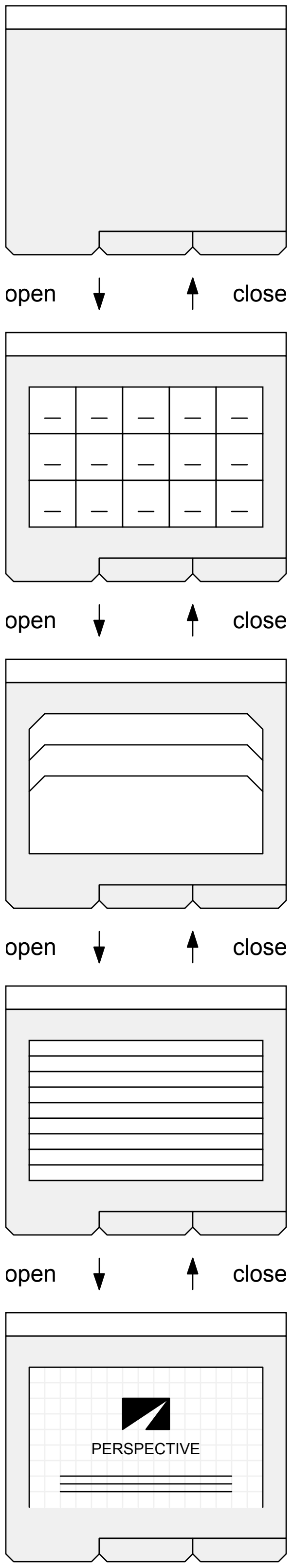
You can also open a drawer, folder, or page by selecting it and pressing the ENTER key. Or by double-clicking one with the mouse (this is the most convenient and frequently used method).

Closing Drawers, Folders, and Pages

You can close drawers, folders, and pages by executing the Close command.



You can also close a drawer, folder, or page by clicking the work area's tab at the bottom of the screen. Pressing the ESC key will also do the job.



AREA VIEW

Perspective's work areas are initially blank at startup.

When you open a cabinet you enter...

CABINET VIEW

... and all of the drawers in that cabinet are shown.

When you open a drawer you enter...

DRAWER VIEW

... and all of the folders in that drawer are shown.

When you open a folder you enter...

FOLDER VIEW

... and all of the pages in that folder are shown.

When you open a page you enter...

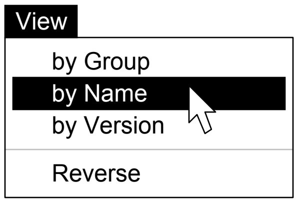
PAGE VIEW

... and all of the shapes on that page are shown. Pages can contain graphics, text, and calculated values.

Sorting and Searching

Sorting

Drawers, folders, and pages can be sorted in a variety of ways. Drawers, for example, can be ordered by group, name, or version. Just select the appropriate command from the View menu:

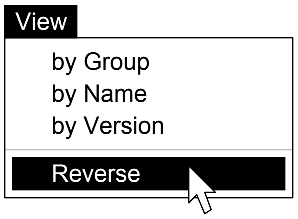


Folders can be sorted by name and version.

Pages can be sorted in various ways, depending on which values are displayed on their edges. The sort commands in Folder View are based on the first selected page, if there is one; on the first page in the folder, otherwise.

Reversing

You can reverse the order of drawers, folders, and pages by using the Reverse command.



The Reverse command allows you to sort page edges, say, by date (oldest on top), then reverse the order so the most recent is first.

Helpful Hints

• To sort one item within another, sort the minor item first. To sort name within group, for example, sort by name first, then by group.

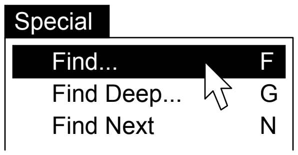
• Drawers automatically sort by version within name within group when opened; folders sort by version within name; pages sort as they were sorted last by the most recent user.

Searching

You can quickly locate drawers, folders, and pages using the Find, Find Deep, and Find Next commands on the Special menu.

Finding

When you execute the Find command, you are prompted for the text you would like to locate.

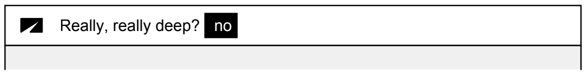


If the text is found in the current view, the item is selected and positioned to make it visible. If the text is not found, a message is displayed.

Finding Deeper

The Find command searches the current view only. To search inside unopened drawers, folders, and pages, use the Find Deep command.

Find Deep requires that you select which objects are to be searched before you execute the command. You are then prompted for the text to find, and are asked how deep you want to go:



If you answer 'yes', every shape on every page will be searched. This a very thorough, but very slow search; normally, limiting the search to the page edge level (by answering 'no') is sufficient.

If the text is found, the object is selected and the work area positioned to make it visible. If the text is not found, a message is displayed.

Finding More

To continue any search — single level, deep, or really deep — use the Find Next command.

Copying and Pasting

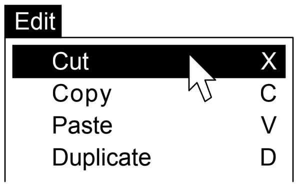
The Clipboards

A clipboard is a holding place for items you want to move or copy. Perspective has five: one each for drawers, folders, pages, shapes, and text.

The drawer and folder clipboards actually store references only, since these items are typically too large to fit into memory; the others contain actual pages, shapes, and text. The clipboards are invisible and cannot be viewed.

Cutting

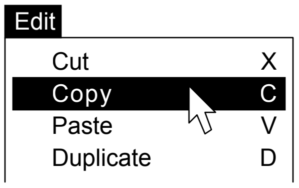
To move items to a clipboard, select the items you want to move and choose the Cut command:



The original items are removed from their current location and placed on the appropriate clipboard. Only pages, shapes, and text can be cut.

Copying

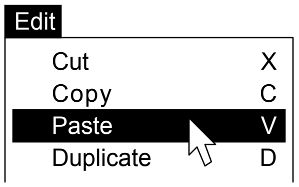
To copy items to a clipboard, select the items you want to replicate and execute Copy:



Replicas of the selected items are placed on the appropriate clipboard (just the names of drawers and folders). The original items are not affected.

Pasting

To make a copy of what is currently on a clipboard and place it elsewhere, just Paste:



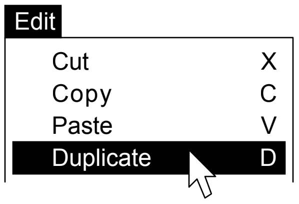
A copy of the items on the clipboard is inserted into the current cabinet, drawer, folder, page, or text shape, as appropriate. Drawers and folders are actually copied from disk at this time.

Pasted drawers, folders, and pages are placed after the selection or at the end (if there is no selection). Pasted text replaces the selection. Pasted shapes appear in their original locations.

The items on the clipboard remain intact.

Duplicating

You can also replicate drawers, folders, pages, and shapes — bypassing the clipboards — with the Duplicate command on the Edit menu:



When this command is executed, all selected items are replicated immediately and placed just below (or to the right of) the current selection.

Helpful Hints

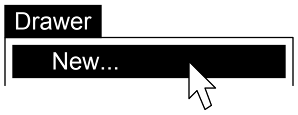
• Duplicating is handier if you only want one copy; copying and pasting is better otherwise.

• See 'Duplicating Shapes' in the Drawing chapter for more information about replicating shapes.

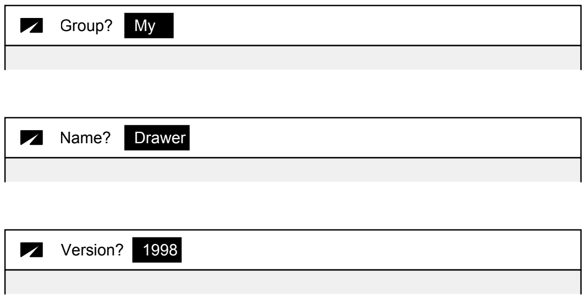
Working with Drawers

Creating Drawers

You can make new drawers by selecting the New command from the Drawer menu in Cabinet View:

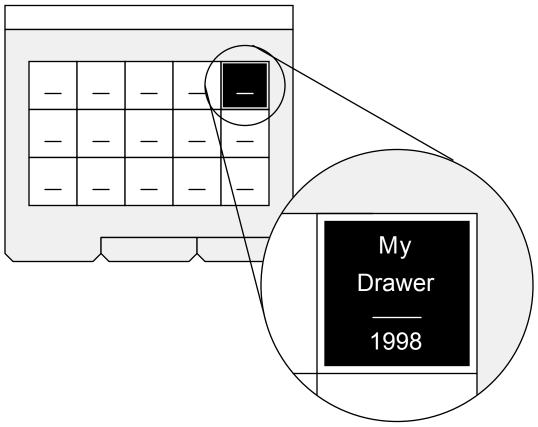


Perspective will prompt you for the new drawer's group, name, and version:



Each of these values can be up to 20 characters in length. All characters, including spaces, are allowed. If you don't want a group, name, or a version, just press ENTER to continue.

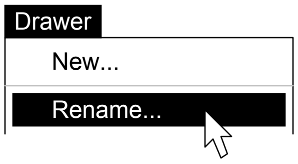
The new drawer is added immediately following the selected drawer, if any. It is added to the far right of the cabinet, otherwise.



Since drawers can be sorted in various ways, and since other users may also add drawers, the position of a drawer will change from time to time.

Renaming Drawers

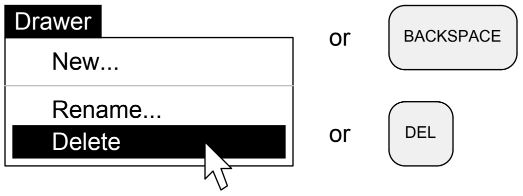
To rename a drawer, first select the drawer you want to change. Then run the Rename command from the Drawer menu:



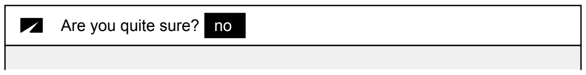
You will be prompted for a new group, name, and version for the drawer. The default responses are set to the current values, so if there is no change you can simply press ENTER to continue.

Deleting Drawers

You can delete drawers by selecting one or more of them and executing the Delete command:



Since this command cannot be undone, you are prompted before the delete takes place:



If you answer 'yes', the drawers are gone.

Helpful Hints

• Use short, but meaningful names. Long names clutter the screen and are hard to find.

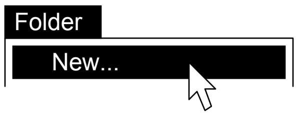
• Always keep current backups of all your work. Backup the contents of drawers before deleting them in case you change your mind.

• Do not rename drawers that are referenced in programs because the programs will probably not work properly if you do.

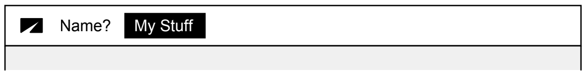
Working with Folders

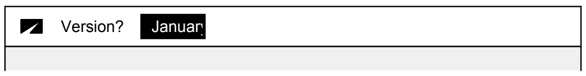
Creating Folders

You can make new folders by selecting the New command from the Folder menu in Drawer View:



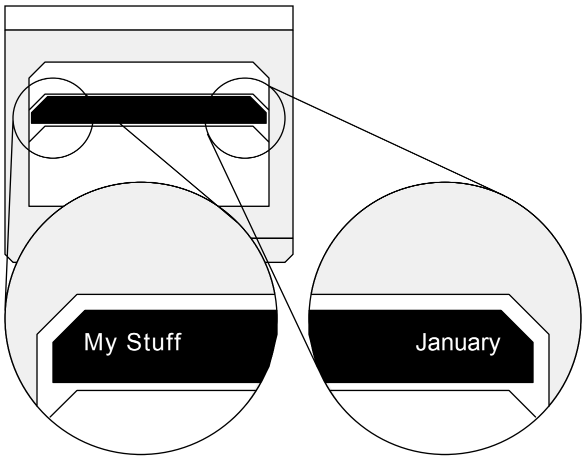
Perspective will prompt you for the new folder's name and version:


Name? My Stuff


Version? Januar

Folder names can be up to 60 characters in length. Versions can be up to 20. All characters, including spaces, are allowed. If you don't want a name or a version, just press ENTER to continue.

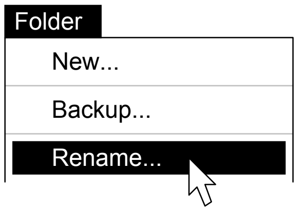
The new folder is added immediately following the selected folder, if any. It is added to the bottom of the drawer, otherwise.



Since folders can be sorted in various ways, and since other users may also add folders, the position of a folder within a drawer may change from time to time. Folders are sorted by version within name when they are opened.

Renaming Folders

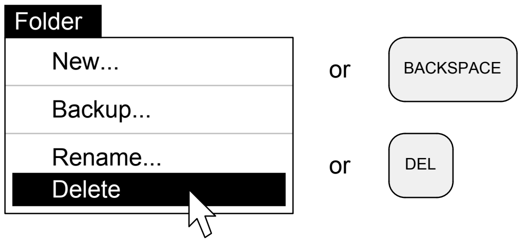
To rename a folder, first select the folder you want to change. Then run the Rename command from the Folder menu:



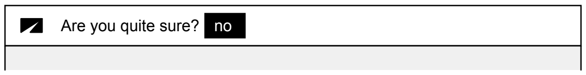
You will be prompted for a new name and version for the folder. The default responses are set to the current values, so if there is no change, you can simply press ENTER to continue.

Deleting Folders

You can delete folders by selecting one or more of them, then executing the Delete command:



Since this command cannot be undone, you are prompted before the delete takes place:


Are you quite sure? no

If you answer 'yes', the folders are gone.

Helpful Hints

• Backup folders before deleting them in case you change your mind.

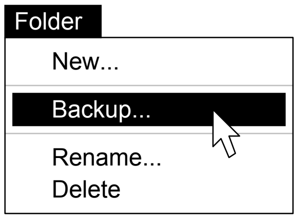
• Do not rename folders that are referenced in programs because the programs will probably not work properly if you do.

Working with Folders

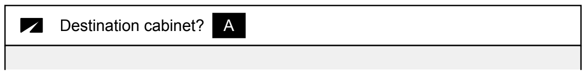
Backing Up

You can make copies of drawers, folders, and pages with the Copy command, then paste them elsewhere as backups. But it is easier and less error-prone to make backups as described below.

The Backup command only appears in Folder View. Just select the folder or folders you want to archive, and execute the command, like so:



Perspective will ask you for a destination cabinet:



A backup copy of the folder(s) will be placed in the specified cabinet in a drawer with the same name as the current one. If such a drawer does not exist, it will be created automatically.

Helpful Hints

• Folders are free, and they are the only means of grouping pages in Perspective. So use as many as you need to keep related pages together and unrelated pages somewhere else.

• You can use the Backup command to copy folders to floppy disks (usually Cabinet A).

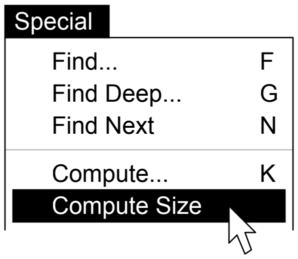
• In a multi-user environment, it is a good practice to allocate cabinets for backups only. A separate backup cabinet for each active cabinet is ideal. You can give backup cabinets easy-to-remember names in the configuration file.

• If your drawer names are not unique across cabinets, this function should not be used since naming conflicts may arise.

Computing Sizes

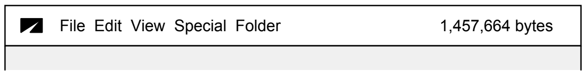
When taking backups, it is sometimes necessary to know the actual number of characters required to store a drawer, folder, or page on disk.

You can use the Compute Size command to display the size of selected drawers, folders, and pages in bytes.



In all views except Page View, you must select the desired items before executing the command. In Page View, the current page is used.

The result is displayed in the message area of the menu bar, like so:



Note that there is overhead in drawers and folders that is not included in computed figures at the page level. Three pages, for example, may total 15,000 bytes — but a folder containing those pages could be as much as 3500 bytes larger.

Helpful Hints

• Page overhead is about 850 bytes per page. This space is used to store edge data.

• Folder overhead is about 950 bytes per folder. This space is used for internal folder information.

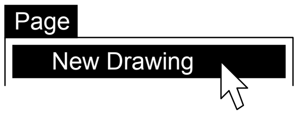
• Drawer overhead is about 100 bytes per drawer. This space is used for the group, name, etc.

• When taking backups, compute sizes at the folder level. These figures include everything but the drawer overhead, which is minimal.

Working with Pages

Creating Pages

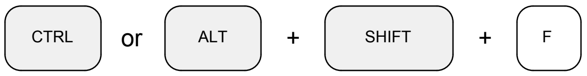
You can make new drawing pages in any folder with the New command on the Page menu:



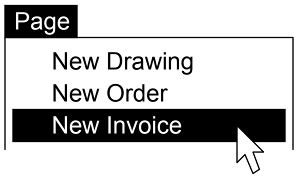
The new page is added after the selected page, if there is one; it is added at the bottom, otherwise.

Creating Forms

You can convert drawings into forms (or forms into drawings) by pressing...



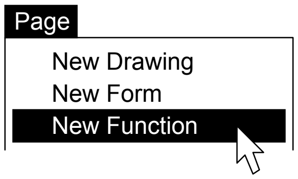
You can also add 'New Form' items to the Page menu via the programming language:



See the Programming chapter for information about adding 'New Form' items to the Page menu.

Creating Functions

You can create function pages only when you are in a folder in the Perspective Programs drawer:

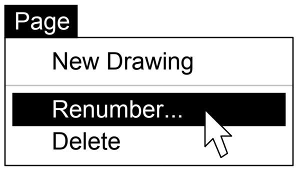


See the Programming chapter for information about the Perspective Programs drawer and the kinds of folders and pages you can put in it.

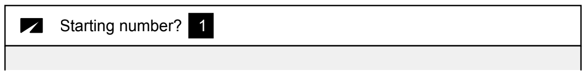
Renumbering Pages

All pages have a page number on their bottom edge; pages can have page numbers in the body of the page and on their top edges, as well.

To renumber the pages in a folder, you must first arrange them into the desired sequence. Use the Cut and Paste commands on the Edit menu. Then select Renumber from the Page menu:



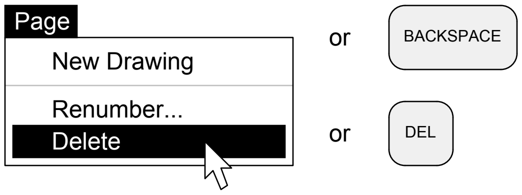
You will be prompted for a starting page number:



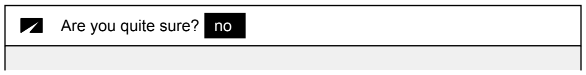
See 'The Name Slice' in the Form Making chapter for more information about page numbers.

Deleting Pages

You can delete one or more pages by selecting them, then executing the Delete command:



You will be prompted before the delete occurs:



If you answer 'yes', the pages are gone.

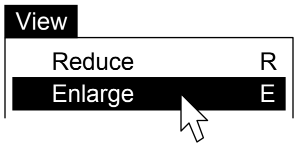
Helpful Hint

• Renumber immediately after adding new pages.

Enlarging and Reducing

Enlarging

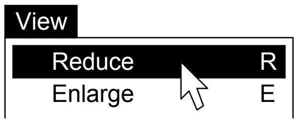
In Page View, you can magnify the work area so that shapes appear twice as large as they actually are. Simply select the Enlarge command from the View menu, and it's done.



You can enlarge further, making objects appear four times as large as they really are, by executing the Enlarge command on the View menu a second time.

Reducing

The Reduce command on the View menu reverses this process. If you are currently at 400% of actual size, executing the Reduce command will return your page to 200%.



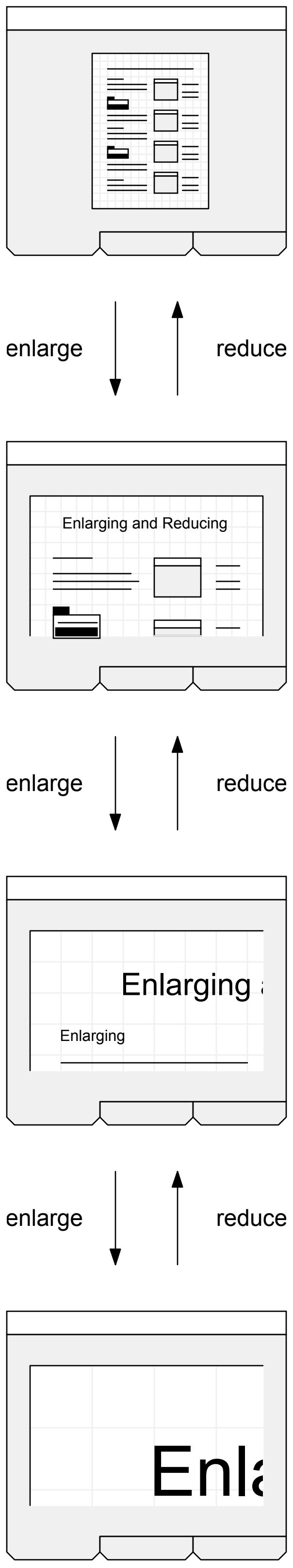
Reducing when the current view is 200% sets the display back to normal.

You can even reduce the page to less than normal size by selecting the Reduce command when the page is displayed at 100%. At this magnification level, shapes are drawn 1/2 their actual size.

Helpful Hints

• You can use the 50% magnification setting to evaluate the quality of your page designs. Make sure your eye is attracted to the most important parts of the page, and that your margins and inter-shape spacings are consistent.

• You can also use the 50% setting to review the flow of a document when your titles are 3 lines per inch or larger.



50%
REDUCTION

Useful for viewing an 8-1/2 by 11 inch page on a standard VGA monitor.

100%
ACTUAL SIZE

This is the normal, what-you-see-is-what-you-get-when-you-print view.

200%
ENLARGEMENT

Useful for working with small objects and with greater precision.

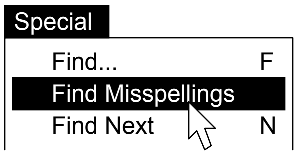
400%
ENLARGEMENT

Useful for working with extremely small objects and exacting precision.

Spell Checking

Finding Misspellings

In Page View, you can locate misspellings on any kind of page with the Find Misspellings command on the Special menu:

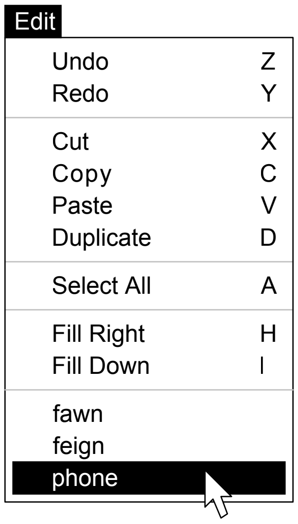


This command examines every text shape on drawing and function pages, and every editable text shape on forms.

When an error is found, checking is suspended, and the problem word is highlighted.

Correcting Misspellings

Suggestions for correction, if any, appear at the very bottom of the Edit menu.

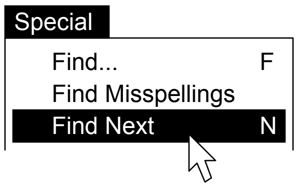


If you select one of the words listed at the bottom of the Edit menu, it will replace the highlighted word. The suggestion is capitalized, as necessary, to match the original word.

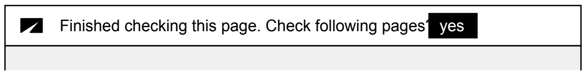
You can also retype the word yourself, of course, if you know the correct spelling.

Checking Further

To continue checking the page for spelling errors, simply execute the Find Next command from the Special menu:



If no further errors can be found on the page, you will be asked if you want to continue:



The default is 'yes'. Press ENTER if you want to check the remaining pages in the folder. Change the response to 'no' and press ENTER, or press ESC, or click in the work area to stop checking.

Adding and Deleting Words

You can add and delete words from the lexicon using the Add Word and Delete Word commands on the Special menu. A dialog will prompt you for the word to add or delete. The currently selected word, if any, will be supplied as the default.

Helpful Hints

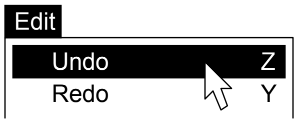
• Spell checking starts on the current page, in the shape containing the insertion point or text selection (if there is one). You must click on a blank part of a page before checking to scan an entire page, and you must start at the first page if you want to check all the pages in a folder.

• If you don't know how to spell a word, spell it phonetically, even if it looks odd (like 'elifant' or 'fone' or 'katastrofee'). Perspective's suggestion algorithm is strictly phonetic and works best when misspelled words are entered exactly as they sound. Suggestions are not provided for transposed letters and other typing errors that you can correct more efficiently yourself.

Undoing, Redoing, and Saving

Undoing

Before making a change to a page, Perspective saves a copy of the page in memory. This means that if you make a mistake, you can restore the page to its previous state simply by executing the Undo command on the Edit menu.

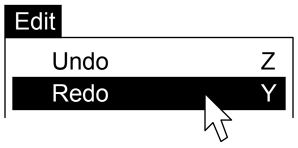


Perspective keeps up to 10 copies of each page in memory at any time, so you can undo more than once, if necessary.

If Perspective cannot undo an operation, the LOGO will blink and an audible click will be heard when you attempt to execute this command.

Redoing

If you undo too much, you can 'undo the undo' by executing the Redo command:



You must redo immediately after you undo. You can redo as many times as necessary to return to your starting position.

Undoing and Redoing Text

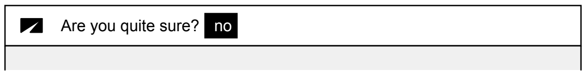
Perspective's text editor has its own undo and redo buffers where copies of edited text are kept while editing is in progress. These copies are also kept 10 levels deep.

You can undo (and redo) while editing text from the point in time that you started editing, to the current point in time. The text undo and redo buffers are cleared, however, whenever you end an edit (usually by clicking elsewhere on the page or tabbing to some other text shape).

Warnings

Some commands cannot be undone, but can be reversed in one way or another. You can delete a drawer that you added by mistake, for example, or you can rename a folder a second time if you got the name wrong the first time.

Operations that cannot be undone in any way (like deleting drawers, folders, and pages) are always preceded by a message confirming that you want to proceed, and giving you a chance to cancel:

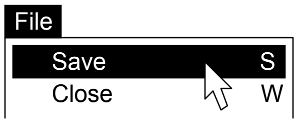

Are you quite sure? no

If you answer 'no', the command is aborted.

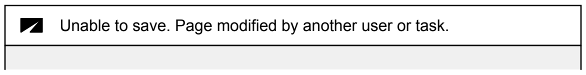
Saving

You do not have to worry about saving your work in Perspective. All operations on drawers and folders are immediate and permanent, and any changes you make on pages are automatically saved when you print, close, select another work area, or quit the program.

If you are working on a page for a long time, however, you may want to force Perspective to save your work on disk. You can do so with the Save command on the File menu:



If two or more people attempt to modify the same page at the same time, the first person's changes will be saved, and the rest will be notified like so:


Unable to save. Page modified by another user or task.

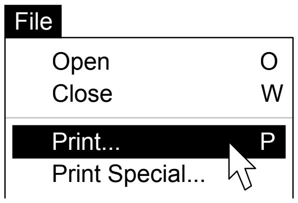
At this point, you can copy your data, reopen the page to see the other person's changes, and then re-enter or paste your information, as appropriate.

Printing

Printing Pages

The Print command on the File menu allows you to print one or more copies of one or more pages using the default printer and settings.

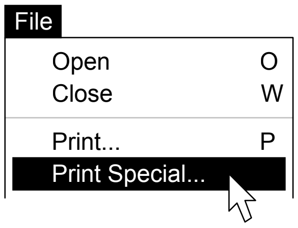
In Folder View, you must select the pages you want to print before executing the command. You are prompted for the number of copies.



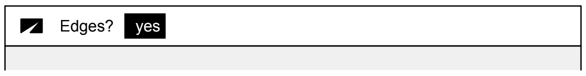
In Page View, only the current page is printed.

Printing Edges

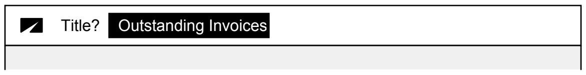
The Print Special command allows you to specify a variety of printing options.



In Folder View, you can print the edges of the selected pages (instead of the actual pages) by answering 'yes' to the first prompt:


Edges? yes

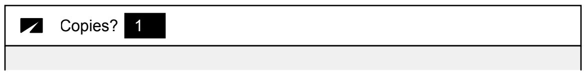
If you choose to print edges, you will be prompted for a title to appear on the printout:


Title? Outstanding Invoices

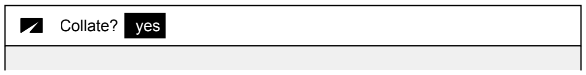
The default title is the folder name and version, as it is displayed in the work area's tab.

Printing Options

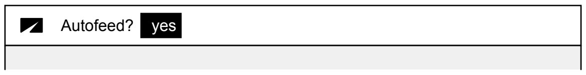
The Print Special command, in any case, will continue to prompt you for additional information:


Copies? 1

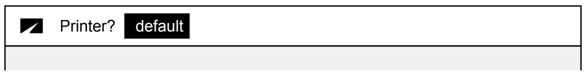
If more than one copy of more than one page is specified, you will be asked if you want the output collated:


Collate? yes

You will also be asked if you want to use the default paper tray, or if you will be manually feeding the printer (as with odd-sized pages):


Autofeed? yes

Finally, you will be asked which printer you intend to use. The default is acceptable in most cases.


Printer? default

Other printers can be specified by entering their names as assigned by your system administrator and described in the Appendixes under the title 'The Configuration File'.

Helpful Hints

• Odd-sized pages (such as envelopes) should be fed into the printer short side first, with the longer edge parallel to the page guides. You may or may not have to use the Print Special command.

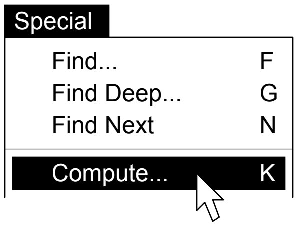
• In Folder View, don't forget to select the pages you want to print before executing the command.

• When printing a large number of pages, divide them into small batches to simplify recovery from printer jams and other mechanical problems.

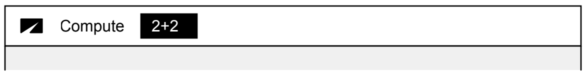
Computing

Computing Figures

The Compute command displays a dialog that can be used like a calculator.



Simply enter the expression you wish to evaluate:

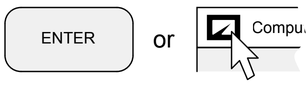


Numbers, parentheses, and the usual arithmetic operators are allowed:

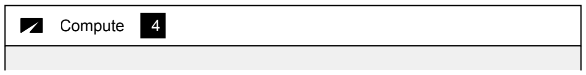
SYMBOL	DESCRIPTION
+	add
-	subtract
*	multiply
/	real division
DIV	integer division
MOD	remainder after integer division
^	raise to power

Commas, dollar signs, and other formatting characters should not be entered.

Press ENTER or click the LOGO to evaluate:



The result is displayed as the next entry, so you can copy it, cut it, edit around it, or replace it with another expression to be evaluated:



Press ESC or click outside the menu bar to exit.

Built-In Functions

Besides arithmetic expressions, you can enter any of Perspective's built-in functions in the Compute dialog. A partial list appears here:

FUNCTION	RETURNS
ARCCOS(number)	arccosine of number
ARCSIN(number)	arcsine of number
ARCTAN(number)	arctangent of number
COS(number)	cosine of number
GETDATE	current date
GETDAYOFWEEK(date)	day number, 1=Sunday
GETTIME	current time
SQRT(number)	square root of number
SIN(number)	sine of number
LOG(number,base)	logarithm of number
TAN(number)	tangent of number

Functions can be entered in upper or lower case. Parentheses around parameters are required.

See the Function Reference chapter for a complete description of all built-in functions, and the Appendixes for an alphabetical list.

Program Execution

The Compute command can also be used to execute and debug programs you have written with Perspective's built-in programming language. The full name of each function must be entered. This is a handy way to execute conversion, end-of-year, and other functions that you don't want cluttering the menus.

See the 'Functions' topic in the Programming chapter for further information.

Helpful Hints

• Select and copy complicated formulas before you press ENTER in case there are errors. Then, instead of re-entering the entire expression, you can simply paste and make corrections.

• You can copy values from pages and paste them into the Compute dialog. You can also copy calculated values from the dialog into pages.

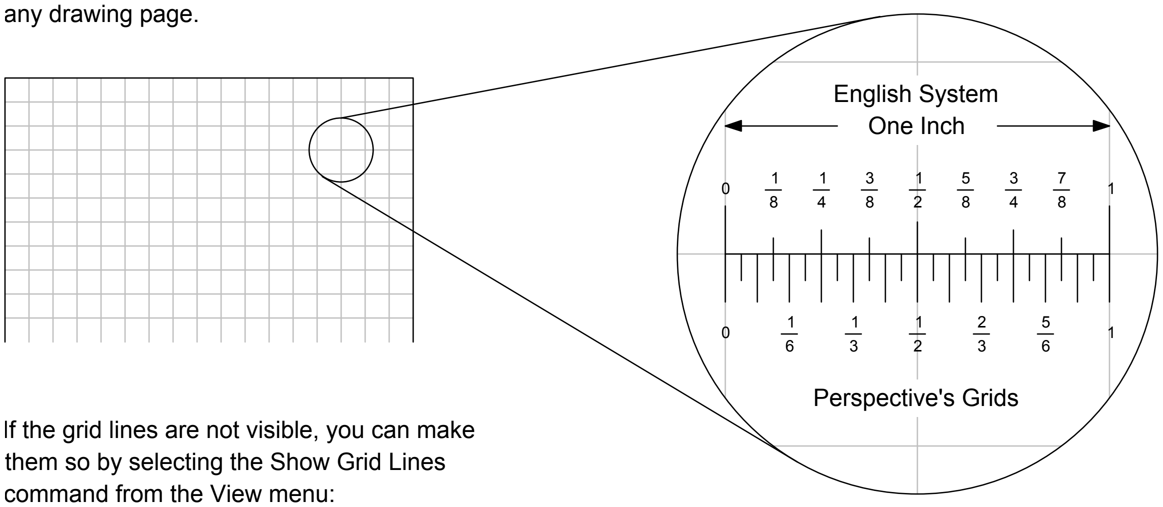
3

DRAWING

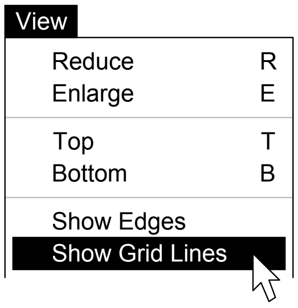
The Grids

The Visible Grid

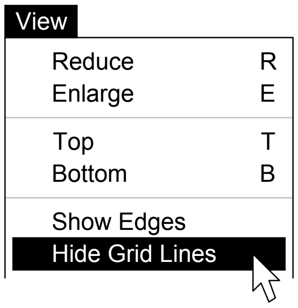
Perspective's coordinate system makes it easy to lay out good-looking pages. Light blue grid lines, spaced 1/2-inch apart, can be displayed on any drawing page.



If the grid lines are not visible, you can make them so by selecting the Show Grid Lines command from the View menu:



The Show Grid Lines command changes to the Hide Grid Lines command whenever the grid is showing, like so:



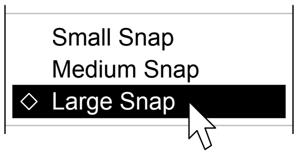
You can use this command to make the grid lines invisible. Grid lines never print.

The Invisible Grids

Perspective further divides each inch so you can align shapes to 1/12 and 1/24-inch boundaries.

This system is similar to the one used by printers for centuries. It lets us divide inches into halves, quarters, and eighths, without leaving the grid. It also allows us to divide inches evenly into thirds, sixths, and twelfths — impossible in the familiar English and Metric systems — making it easy to center shapes of almost any size.

Perspective automatically aligns shapes to the current 'snap' grid. Commands for setting the snap grid are in the middle of the View menu:



The Large and Medium Snaps correspond to the 1/12 and 1/24-inch boundaries discussed above. The Small Snap allows drawing at 1/72 of an inch (normally used when pages are enlarged). The diamond indicates which snap is currently active.

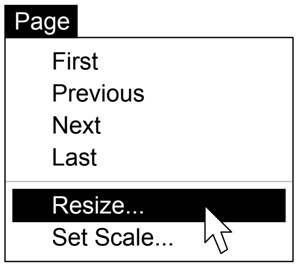
Helpful Hint

• Stick with Large Snap; it makes drawing much easier. Try not to use the Small Snap at all.

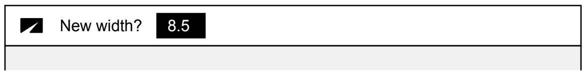
Sizing and Scaling Pages

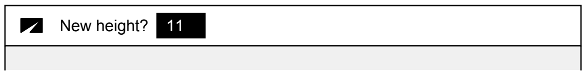
Sizing Pages

New drawing pages are initially 8-1/2 inches wide by 11 inches tall. You can change the size of a drawing, however, with the Resize command:



When you execute this command Perspective will ask for the new width and height (in inches):


New width? 8.5

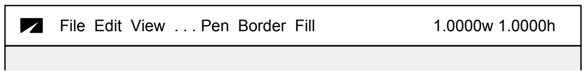

New height? 11

The current values are provided as defaults — just press ENTER if there is no change.

Minimum page size is 2 x 2; maximum page size is 11 x 11. Page sizes are always rounded to the nearest half-inch.

Shape Sizes

When a shape on a page is selected, its size is displayed in the message area of the menu bar:

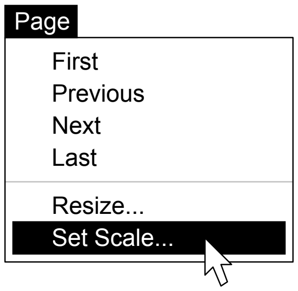

File Edit View . . . Pen Border Fill 1.0000w 1.0000h

The number followed by the 'w' indicates the width of the shape; the 'h' number indicates the height.

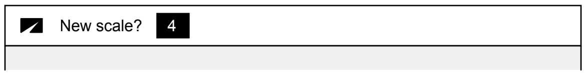
The unit of measure is normally inches, and is always the same for every shape on the page. You can, however, adjust these readings to any arbitrary scale using the following techniques.

Scaling Pages

To set the scale, determine how many units one inch on the page should represent, then execute the Set Scale command on the Page menu:



Perspective will clear the menu bar and ask you for the new scaling factor:


New scale? 4

The default is the current factor for the page. Shape dimensions (in inches) are multiplied by the current scaling factor before being displayed.

Conversions

To convert any scale to a Perspective scaling factor, use the following formula:

$$s = r / d$$

where s is the scale, r is the amount in real-life units, and d is the amount in drawing units.

Example

Say we want to draw a plan for an addition to our house. Standard architectural drawings use a scale where 1/4 inch on the drawing equals 1 foot in real life. We apply the formula as follows:

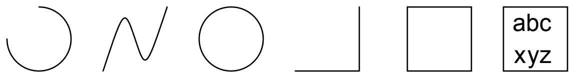
$$s = r / d = 1 / .25 = 4$$

As you can see, r is 1 (the real life measure) and d is .25 (the drawing measure). The result is 4, which is entered in the Set Scale dialog. Readings in the menu bar now represent feet.

Graphic and Text Shapes

Shapes

You can put six kinds of shapes on pages: arcs, curves, ellipses, polygons, rectangles, and text.

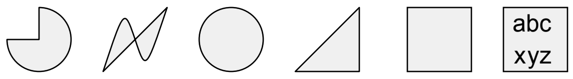


Note that ellipses include ovals and circles of all sizes, and that rectangles include squares.

Polygons include straight lines, angles of all sorts, triangles, parallelograms, pentagons, hexagons, and most other multi-sided figures.

Text shapes are always rectangular and are sometimes called text blocks or fields.

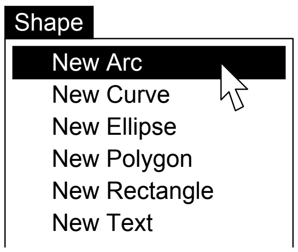
All shapes can be clear or filled. The shapes above are clear; those below are filled.



Note that filled arcs become wedges, and that an extra line is added (if necessary) to connect the endpoints of filled curves and polygons.

Creating Shapes

To add a shape to a page, simply select the appropriate command from the Shape menu.



The new shape will appear in the center of the visible area of the current page, on top of any existing shapes.

All new shapes are initially 1/2-inch square.

Editing Shapes

You can operate on shapes in a variety of ways. Sometimes you select one or more shapes and execute a command from a menu; at other times, you manipulate shapes directly with the mouse.

All Shapes

All shapes can be selected, colored, grouped, layered, moved, sized, and duplicated. These operations are discussed on pages 38-42.

Graphic Shapes

Graphic shapes can also be rotated and flipped. See page 43 for further information.

Arc, Curve, and Polygon Shapes

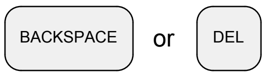
Arcs, curves, and polygons can be reshaped, and can also have arrows added to either or both ends. See pages 44 and 45 to find out how.

Text Shapes

You can choose the face, style, alignment, and size of the text in any text shape. You can also edit the text inside the shape. See page 46.

Deleting Shapes

To remove one or more shapes from a page, all you have to do is select the appropriate shapes and press the BACKSPACE or DELETE key.



If you delete shapes by mistake or accident, you can restore the page to its previous state using the Undo command on the Edit menu. The last ten operations can be undone.

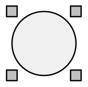
Helpful Hint

• Make a scratch drawing page in a folder and experiment with actual shapes on the screen as you read through the remainder of this chapter.

Selecting Shapes

Working with Shapes

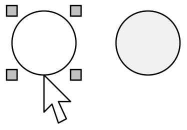
Shapes must be selected before you can work with them. Selected shapes have small, gray 'handles' at their corners, as shown below.



Zero, one, or more shapes can be selected at a time. There are a number of different ways to select shapes.

Selecting a Single Shape

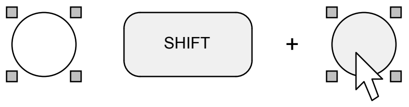
You can select a shape simply by clicking on it. If the shape is filled, you can click anywhere on it. If it is not filled, you must click on its border.



All other shapes are deselected when you click.

Selecting More Shapes

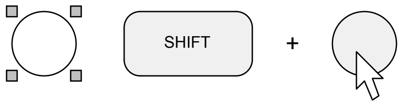
You can add to the current selection by holding the SHIFT key while you click other shapes.



Currently selected shapes remain selected.

Selecting Fewer Shapes

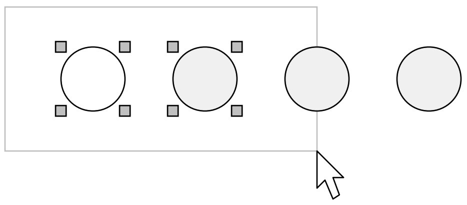
You can also remove shapes from the current selection by SHIFT-clicking them.



Again, other shapes are not affected. If you inadvertently deselect a shape, click it again.

Selecting Many Shapes

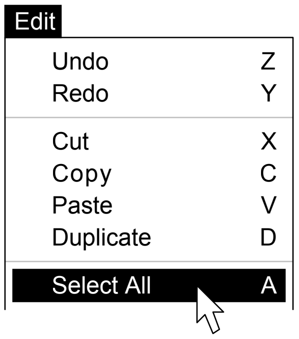
You can select multiple shapes simultaneously by drawing a 'rubber band' around them.



Only shapes that are fully enclosed in the rubber band will be selected.

Selecting All Shapes

You can select all the shapes on a page by choosing Select All from the Edit menu.



Select All selects all shapes, whether they are selected or not prior to issuing the command.

Deselecting

You can deselect everything by clicking on a blank part of a page, or in the gray work area surrounding the page.

Helpful Hints

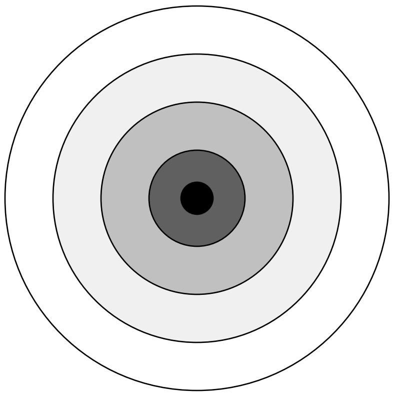
• Sometimes it is easier to select more than you want (either with the rubber band or the Select All command), then deselect the ones you don't want with SHIFT-clicks.

• You can use the rubber band to select clear and very small shapes that are hard to click directly.

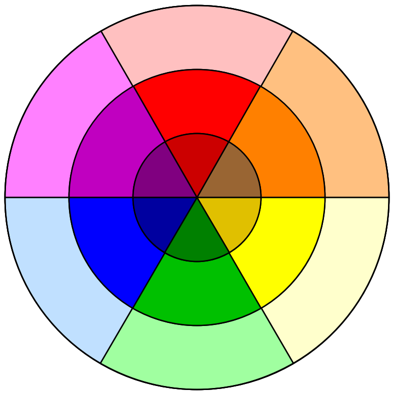
Coloring Shapes

The Color Palettes

The Perspective color palettes include clear, white, black, and three shades of gray.



The palettes also include the six colors of the rainbow in light, normal, and dark shades.



Colors can be applied to the edges of shapes, the interior of shapes, and to text.

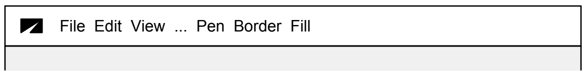
Helpful Hints

• Note that dark orange is brown, and that light orange is in the tan family. Light yellow is almost ivory and is very pretty; dark yellow is not.

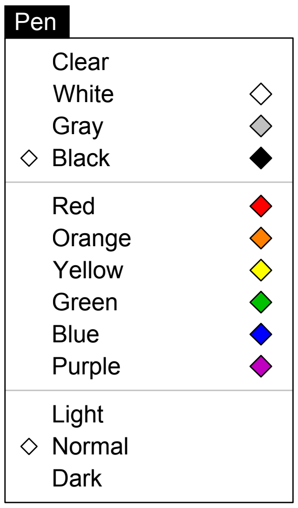
• Note also that colors rarely look the same when they are printed as they do on the screen. Many printers let you adjust their color handling from the Windows environment. Turn all options OFF for the fastest (and frequently the best) color.

The Color Menus

The Pen, Border, and Fill menus are used to select colors for text and shapes. All three menus include the same color and shade options.



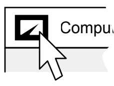
The Pen menu is used for coloring text. The Border and Fill menus are used to color the edges and the interiors of all kinds of shapes, including text shapes.



Select the desired color first, then choose the shade (light or dark).

Helpful Hints

• You can use white curves with white borders to 'cover up' unwanted portions of other shapes. The wavy edge on the right of the diagram below, for example, is made with such a curve:

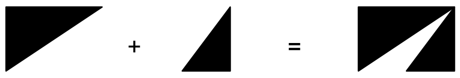


• Text shapes with clear borders align differently than those with visible borders. See the 'Fonts' topic in this chapter for further information.

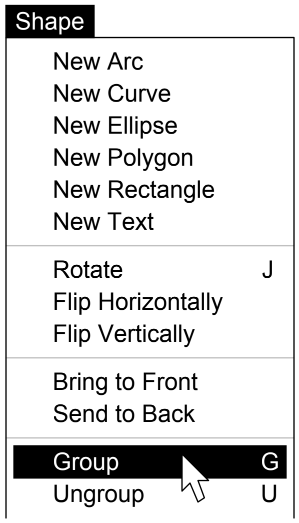
Grouping and Layering Shapes

Grouping Shapes

Sometimes it is desirable to treat several shapes as if they were a single object. For example:



We can accomplish this by selecting the shapes and choosing Group from the Shape menu:



The group now behaves as a single object. It can be selected with a single click; when selected, only one set of handles is displayed; and it can be moved by grabbing any part of it.

Ungrouping Shapes

To ungroup shapes, select the group and execute the Ungroup command from the Shape menu.

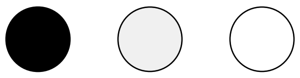
Helpful Hints

• Do not 'overgroup' your shapes. It may seem like a good idea at first, but it makes your shapes difficult to modify later.

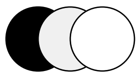
• Groups can include individual shapes, and other groups, as well. There is no limit to the depth you can go. It is good practice, however, to limit your groups to a maximum of two or three levels.

Layering Shapes

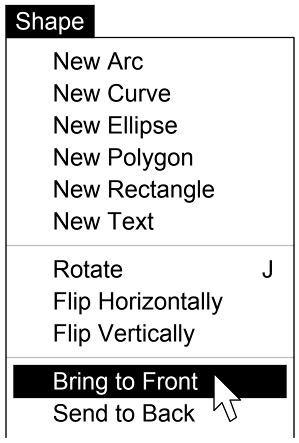
Shapes on a page are drawn from back to front. This has no apparent effect when shapes do not overlap. In the following illustration, for example, it does not matter that the black circle is in back, the gray in the middle, and the white in front:



When we move the circles so they overlap, however, it becomes apparent that back-to-front ordering is critical. Observe:



You can adjust the back-to-front ordering of shapes with the Bring to Front and Send to Back commands on the Shape menu.



When you group shapes, the entire group is brought to the front. When you ungroup, the shapes are inserted at the group's level.

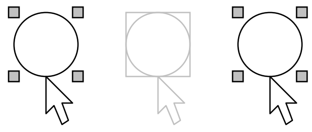
Helpful Hint

• You can do neat things with overlapping shapes. The subtitles on this page, for example, are bold text shapes on top of two long text columns. The technical term is 'upper-middle-class text'.

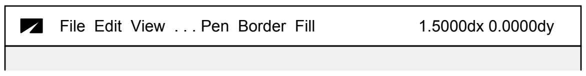
Moving and Sizing Shapes

Moving a Shape

To move a shape on a page, simply grab it with the mouse and drag it to its new location. A filled shape can be grabbed anywhere, but a clear shape must be grabbed on its border.



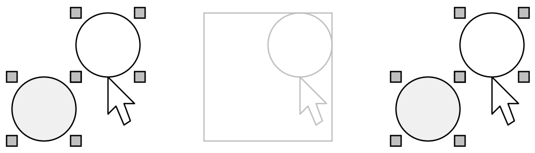
A rectangle enclosing the shape, together with the outline of the selected shape, follows the cursor while the mouse is moving. The distance moved is displayed in the menu bar:



The 'dx' figure indicates horizontal movement; 'dy' indicates vertical movement. Readings are adjusted to the current scale of the page.

Moving Multiple Shapes

To move more than one shape simultaneously, first select all of the desired shapes. Then grab any of the selected shapes and drag.



A rectangle enclosing all of the selected shapes, together with the outline of the shape under the cursor, moves with the mouse.

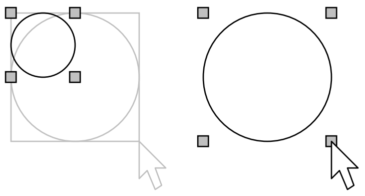
Helpful Hints

• To avoid clicking the handles on a small shape, either move the shape when it is not selected, or enlarge the page first.

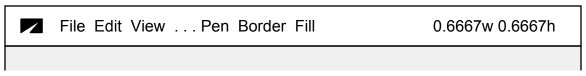
• Aim for the flat parts of curves when dragging and adding vertexes; our code is less than optimal at recognizing clicks on the curvy parts.

Sizing a Shape

To change the size of a shape, grab any of its handles and drag until the desired size is attained.



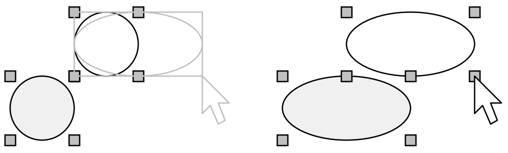
A rectangle enclosing the shape, together with the outline of the selected shape, follows the cursor and indicates the new size of the shape. The new size is also tracked in the menu bar:



The 'w' figure is the width; the 'h' figure is height. Readings are adjusted to reflect the current scale setting of the page.

Sizing Multiple Shapes

To size more than one shape simultaneously, first select all of the desired shapes. Then grab any handle of any shape and drag.



The new size is indicated for the shape whose handle is being dragged; all selected shapes will size accordingly when the mouse is released.

Helpful Hints

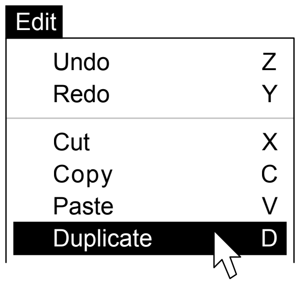
• When ungrouped shapes are sized, sizing is absolute — each shape is enlarged or reduced in each dimension by the same number of units.

• When groups are sized, the boundary sizes absolutely, but the shapes size proportionately: this can move grouped shapes 'off the grid'.

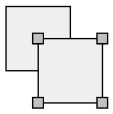
Duplicating Shapes

Duplicating Shapes

In many cases, the easiest way to draw a shape is to make a copy of an existing shape and modify it, either by moving it, sizing it, or both. The Duplicate command allows you to replicate shapes on a page quickly and easily. Just select a shape and execute the command.

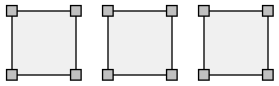


The new shape is selected and is normally placed slightly to the right and below the original.

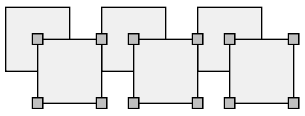


You can move, size, or modify the replicated shape any way you please.

You can also duplicate more than one shape at a time. Just select all of the desired shapes:



And execute the command:



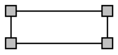
All of the new shapes are selected.

Helpful Hint

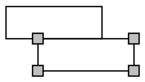
Use the shortcut, ALT-D, when duplicating. It's faster, especially when you want to do some...

Super-Duper Duplicating

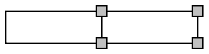
When you move duplicated shapes, the distance moved is remembered. If no actions intervene, this distance is used as the offset for the next Duplicate command. This makes it easy to produce regular drawings, such as tables on forms. Starting with this text shape, for example:



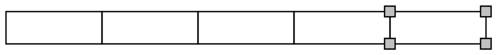
We can duplicate it, like so:



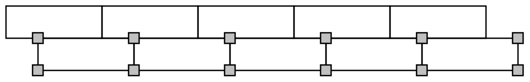
And then we can move it to this location:



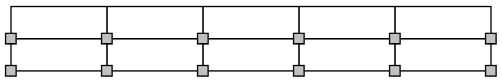
And duplicate it three more times:



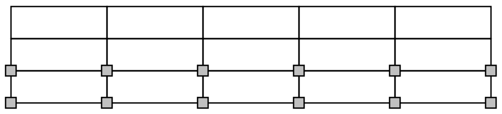
Note that no moves were necessary. Now, if we select and duplicate the whole row:



And move it to here:



We can make as many additional rows as desired simply by pressing ALT-D over and over again.

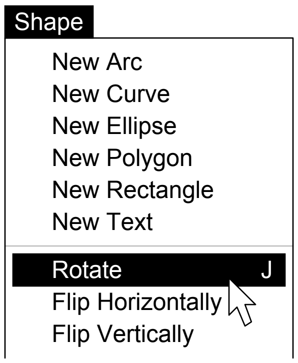


We fill in the headings, and we're done. Nifty.

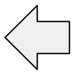
Rotating and Flipping Shapes

Rotating Shapes

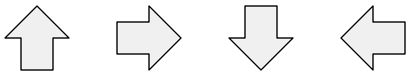
All shapes except text shapes can be rotated clockwise in 90-degree increments. Simply select the desired shape or shapes, and execute the Rotate command on the Shape menu.



This shape, for example,



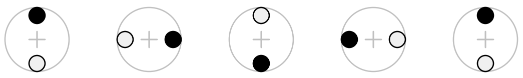
will rotate as shown below:



Note that the fourth Rotate command returns the shape to its original orientation.

Helpful Hint

• Shapes rotate around their individual center points, unless they are part of a group, in which case the center point of the group is used. For example, no change is seen if circles are rotated individually (because they are symmetrical); but when circle shapes are rotated as a group...



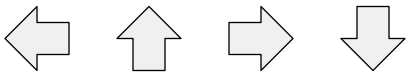
the effect is obvious. Here, both circles rotate around the center of the group (the gray cross).

Flipping Shapes

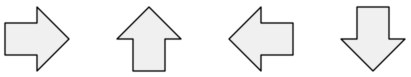
All shapes except text shapes can be flipped horizontally and vertically. Simply select the desired shape or shapes, and execute the appropriate Flip command on the Shape menu.

Flipping Horizontally

These shapes, for example,



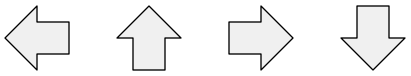
when flipped horizontally, look like this:



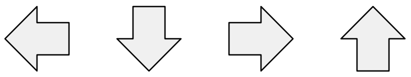
No change is apparent in the second and fourth shapes because they are left-right symmetrical.

Flipping Vertically

These shapes, for example,



when flipped vertically, look like this:



No change is apparent in the first and third shapes; they are symmetrical top and bottom.

Helpful Hint

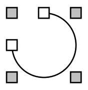
• Shapes flip around their individual center points, unless they are part of a group. Flipping therefore has no apparent effect when flipping symmetrical shapes, unless they are grouped.

See the Helpful Hint at the left for additional information regarding the center points of groups.

Reshaping Shapes

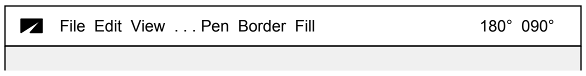
Arcs

Selected arcs have both handles (for sizing) and vertexes (for reshaping).

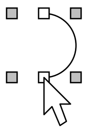


Vertexes look like handles except that they are white, and appear at the ends of the arc, rather than the corners of the shape.

The location of each vertex is displayed in the menu bar when an arc is selected, like so:



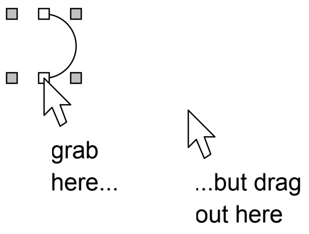
You can reshape an arc simply by dragging its vertexes to new locations.



The location of the vertexes is tracked in the menu bar while reshaping is in progress.

Helpful Hints

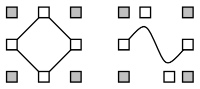
• The easiest way to reshape an arc is to grab a vertex and then move the mouse away from the arc; circle the arc with a wide sweep, watching the message to determine when to let go:



• Resize small arcs (to a larger size) and reshape them while they are large and easier to work with. Squish them back down when you are done.

Polygons and Curves

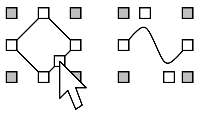
Polygons and curves also have both handles (for sizing) and vertexes (for reshaping).



You can reshape either one by dragging the vertexes to new locations.

Adding and Deleting Vertexes

You can add new vertexes to polygons and curves by double-clicking on an edge.



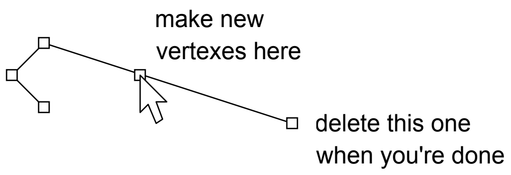
You can delete a vertex by double-clicking it.

Helpful Hints

• Note that a line is a polygon with two vertexes.

• Some polygons have vertexes that cover the handles, making it difficult or impossible to size them. You can group such a shape (all by itself), size the group, then ungroup it.

• When first fashioning a polygon, give yourself a long string to work with by dragging a vertex at one end of the line off to the side. Make new vertexes on this line and place them in the appropriate spots. Delete the extra vertex.

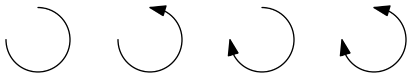


• Aim for the flat parts of curves when dragging and adding vertexes; our code is a bit less than optimal at recognizing clicks on the curvy parts.

Arrows

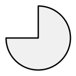
Arrows on Arcs

You can add arrows to the ends of arcs like so:

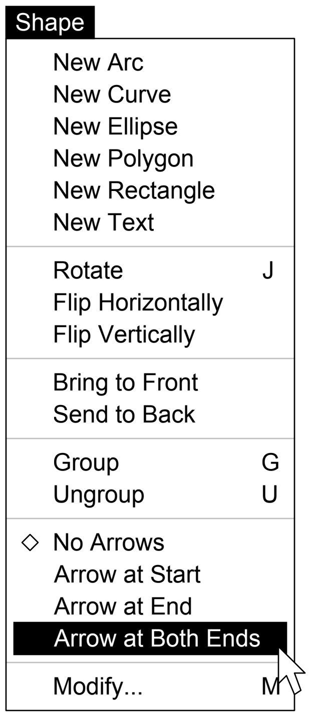


Note that arrows can be at one end of an arc, at the other end of the arc, or at both ends.

You cannot, however, put arrows on filled arcs, because filled arcs are wedges — additional lines are drawn connecting their end points:



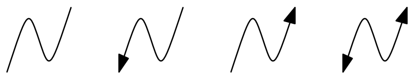
To put arrows on an arc, just select the arc and choose the appropriate command from near the bottom of the Shape menu:



The tiny diamond at the left side of the menu indicates the number and placement of arrows currently on the shape.

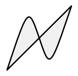
Arrows on Curves

You can also add arrows to the ends of curves:



Note that the arrows can be positioned at either or both ends of the curve.

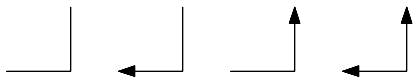
You cannot put arrows on filled curves, because an additional line is automatically drawn to connect the end points of the curve, like this:



The Arrow commands discussed earlier are also used to place arrows on curves.

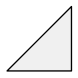
Arrows on Polygons

You can also add arrows to the ends of polygons:



Arrows can be positioned at either or both ends.

Filled polygons cannot have arrows, because an additional line connects the end points:



The same Arrow commands that work on arcs and curves also work on polygons.

Helpful Hints

• If arrows won't show up on straight lines, make sure the selected shape's fill is clear.

• You can also rotate and flip shapes to change the orientation of arrows.

Fonts

Faces

Perspective supports three different text faces. Each text shape has a single face.

Arial
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

Courier
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

Times
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

Note that Arial and Times are proportionally spaced (letters vary in width), while Courier is a mono-spaced face (letters are the same width).

Styles

Three different text styles are supported:

This text is plain.
This text is bold.
This text is italic.

Bold styles are typically used for headings and titles; italic is normally used for emphasis. Each text shape can have only a single style.

Alignments

Text shapes can be aligned in five different ways, as shown below.

This text is aligned to the left and has a ragged right margin.

This text is aligned to the right and has a ragged left margin.

This text is centered and is ragged on both sides.

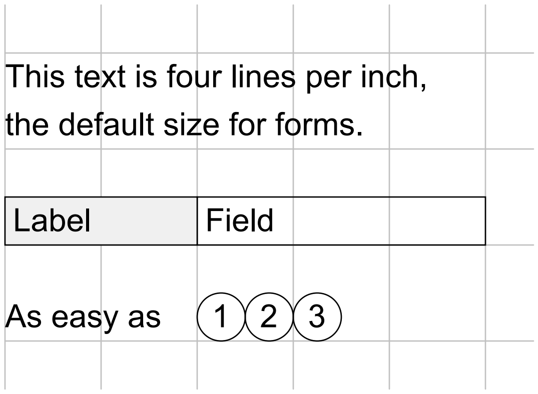
This text is justified so that both left and right margins line up.

T H I S   T E X T   I S   S P R E A D

All text in a shape is aligned in the same way.

Sizes

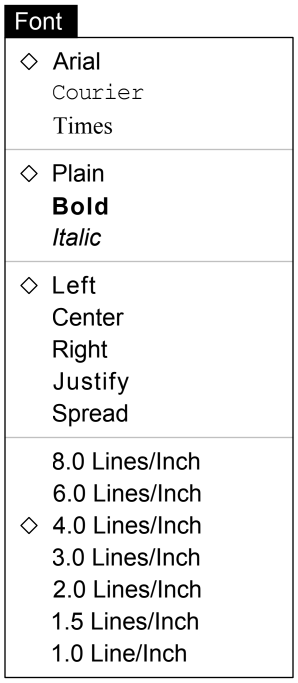
Perspective's seven font sizes are measured in lines per inch to make it easy to align text shapes with the grid and adjacent graphics on a page.


This text is four lines per inch,
the default size for forms.
Label
Field
As easy as
1
2
3

Note that bordered text also aligns with the grid, and is auto-indented on the left and right.

The Font Menu

You select the face, style, alignment, and size of text shapes from the Font menu.



Edges

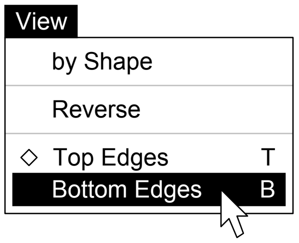
Page Edges

The top and bottom 1/4-inch of each page is reserved for edge shapes — shapes that will appear on the edges of pages in Folder View. Only text shapes are allowed on page edges. Shapes of other kinds appear only in Page View.

Three text shapes are automatically placed on the bottom edge of every page: the type (drawing, form, or function), an identifier, and the page number. Up to seven additional text shapes can be put on the top edge of any drawing or form (see the next chapter for more info about forms).

Edges in Folder View

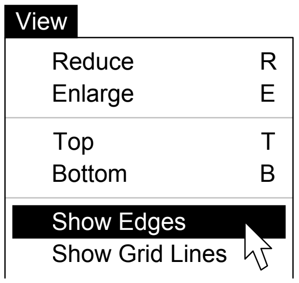
In Folder View, either the top or the bottom edges of pages can be viewed, using the Top Edges and Bottom Edges commands on the View menu:



The tiny diamond at the left of the menu indicates which edges are currently showing.

Edges in Page View

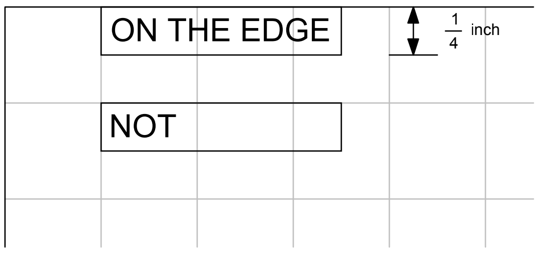
In Page View, edges only show on drawing pages. Use the Show Edges command to expose them:



This command then becomes Hide Edges and can be used to make them invisible again.

Making Edge Shapes

To make an edge shape, just place any text shape within 1/4-inch of the top of a page.



Edge shapes can have any face, style, size, alignment, and color, but only their position and alignment matter in Folder View (where plain, black, borderless, 4-lpi Courier is always used).

Helpful Hints

• Use shapes that are exactly 1/4-inch tall. This is three large grids, or one-half of a light-blue grid square. It is the height of text at 4 lines per inch.

• If you plan to print your edges from Folder View, position your shapes at least 1/4-inch in from the left and right edges of the page. Most printers will not print closer to the edge than this.

• Try to align the edge shapes on different kinds of pages in similar ways, especially if you plan to keep different kinds of pages in the same folder. This makes your edges look better, and makes them easier to search and sort in Folder View.

• Edges in Folder View are always 8-1/2 inches wide, regardless of actual page size. This means you can fit extra edge shapes on narrow pages by making the page wider, adding the shapes, then reducing the page to its original size. Shapes right of 8-1/2 inches never appear in Folder View.

• If edges are hidden, shapes will seem to disappear when dragged or pasted onto the edge.

• Use top edges to indicate the contents of your page — names, dates, states, zip codes, phone numbers, totals, etc. Do not modify the bottom edges except to assign identifiers to your pages.

4

FORM MAKING

Forms

Overview

A form is a printable page that contains both editable and non-editable shapes. The data on the top edge is usually extracted from the body of the page. Forms start out as drawings; they are converted to forms when the design is complete. To convert a drawing into a form, use these:

CTRL or ALT + SHIFT + F

Press the same keys again to convert it back.

Slices

All the text shapes on a drawing can be 'sliced' to reveal four additional layers of information:

Update Slice

no	no	no	no
yes	yes	yes	no

Code Slice

			qty*price

Name Slice

qty	desc	price	

Format Slice

z9		z,zz9.00	$z,zz9.99

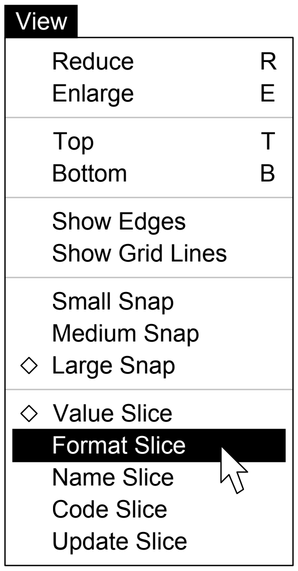
Value Slice

QTY	DESC	PRICE	TOTAL
3	Widget	500.00	$1,500.00

The Value slice is the one we normally see; the Format slice tells Perspective how to display the values. We give names to text shapes in the Name slice, so we can refer to them in formulas in the Code slice. The Update slice is used for pop-up menus and to make background shapes, such as titles and calculated values, non-editable.

Switching Slices

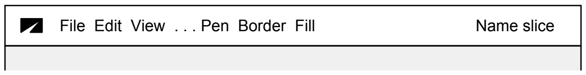
In Page View, the Value slice is the one normally displayed on the screen. You can switch to any of the other slices, however, by simply selecting the appropriate command from the View menu (the current slice is marked with a tiny diamond):



You can also 'flip' through the slices on a page with the CTRL or ALT and ARROW keys:

CTRL or ALT + ↑ or ↓

The next (or previous) slice is displayed, and a message indicates which one is current:



Name slice

Helpful Hints

• Edit each slice, in the order shown on the menu, when designing a new page. Don't jump around.

• Return to the Value slice immediately after editing the others. It is easy to put values in the wrong slice if you forget where you are.

The Value Slice

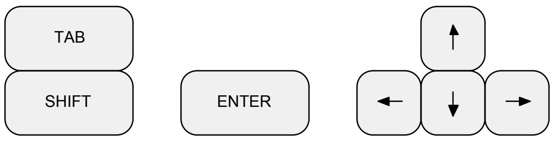
Overview

On drawing pages, you can move, size, color, and manipulate shapes any way you like. You can also enter information into any of the five slices of a text shape. You can even work with edge shapes, if they are showing.

On forms, however, graphic objects are not editable in any way, and only the Value slice of text shapes is visible. Furthermore, only editable text shapes can be changed (see 'The Update Slice' for further information). Handles, vertexes, and edge shapes never show on forms.

Tabbing Around

You can, however, easily enter data on forms because all of the background shapes are skipped when you move the selection with the TAB, ENTER, ARROW, and SHIFT keys.



The TAB key moves the selection to the next editable text shape to the right, if there is one. If there is not, it moves the selection to the first editable text shape on the next line down.

You can use the SHIFT key to reverse the effect of the TAB key, causing it to move left and/or up instead of right and/or down.

The ENTER key moves the selection to the next text shape in the current group, or to the first text shape in the next group. Groups are defined by the layout of the form; touching shapes are considered groups. Movement is top-to-bottom, then left-to-right, within and across groups.

You can use the SHIFT key to make the ENTER key move the selection to the previous, rather than the next logical text shape.

The ARROW keys move the selection left, right, up, or down in an intuitive fashion.

Dragging Around

You can also select one or more text shapes on a form using the mouse. Just click, SHIFT-click, or drag. Only editable shapes are affected.

Cutting, Copying, and Pasting

You can use the Cut, Copy, and Paste commands on the Edit menu to move values around on forms. Values are cut, copied, and pasted in strict left-to-right, top-to-bottom order. Only selected shapes are affected.

Filling Right and Down

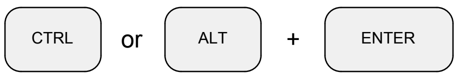
You can also use the Fill Right and Fill Down commands to insert values into editable text shapes on forms. Just select the desired shapes and execute the appropriate command.

Fill Right copies the first selected value in each row to all the other selected shapes in the same row. Fill Down does the same, column by column.

Helpful Hints

• As stated earlier, the ENTER key moves the selection within groups of text shapes — that is, text shapes whose borders touch one another. If your forms are laid out in a reasonable fashion, this will normally make the selection move where you want it to move. You can, however, alter the sequence followed by the ENTER key and force non-contiguous text shapes to act as a group by drawing a rectangle around them; make sure the rectangle fully encloses them. You can give the rectangle a clear border so it doesn't show.

• The value in a selected text shape is completely replaced when you start typing. To edit rather than replace the value, click with the mouse in the desired location. Or press:



and the insertion point will be positioned at the very end of the value in the shape.

The Format Slice

Overview

The Format slice determines how values in text shapes are displayed. Formats are applied immediately following each change in the Value slice. Blank and invalid formats are ignored.

Boolean Formats

You can format text shapes to display boolean values in a variety of ways, using these formats:

```
1 | 0
y | n
t | f
ok | not
on | off
yes | no
true | false
```

Upper, lower, or mixed case letters can be used. Leading and trailing spaces are ignored, but the vertical bar separators are required. Note that the TRUE value is specified first.

Any value entered in a shape with a boolean format is converted to a TRUE or FALSE value, then displayed as indicated in the format.

Date Formats

You can format dates in a variety of ways by combining the following components:

```
m
mm
mmm
mmmm
d
dd
yy
yyyy
```

with dashes, slashes, commas, and/or spaces as separators. All formats must specify month first, day next, and year last.

Use mmm for three-letter month abbreviations; use mmmm for full month names. Month names are always displayed in mixed case.

Time Formats

You can format times in a variety of ways using these components:

```
h
hh
m
mm
s
ss
am
pm
```

with either colons or spaces as separators.

All time formats must specify hours first, followed by minutes. Seconds are optional, but must follow minutes. If no am or pm indicator is included (either will do), a 24-hour clock is used.

Number Formats

Numeric values can be formatted using these characters:

```
$
z
9
```

Commas and a decimal point are also allowed.

The z component can be either upper or lower case, and indicates that specified digits should be blank when zero; 9s always produce a digit.

Numeric values are truncated on the right only.

Examples

VALUE	FORMAT	RESULT
0	true | false	false
true	yes | no	yes
01/31/98	mmmm dd, yyyy	January 31, 1998
01/31/01	mm/dd/yyyy	01/31/2001
3:15 pm	hh:mm:ss am	03:15:00 pm
3:15 pm	hh:mm:ss	15:15:00
1234	9	1234
1234	$z,zzz,zz9.99	$1,234.00

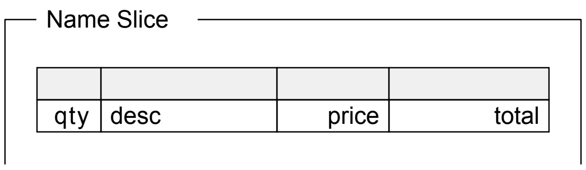
The Name Slice

Overview

The Name slice is used to assign identifiers to text shapes on pages, so they can be referenced in the Code slice and, later, in programs you write using the built-in programming language.

Naming Shapes

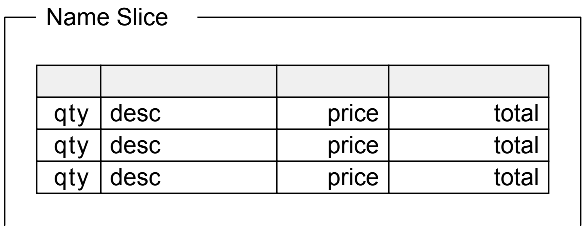
To assign a name to a shape, you simply enter the name in the Name slice, like so:



Shape names can be up to 40 characters long. They must begin with a letter and must not contain spaces. Shorter names are preferred.

Shape Arrays

The same name can be assigned to more than one shape, resulting in an 'array' of shapes:



You can refer to an element of a shape array by including a 'subscript' in square brackets after the shape name. The quantity shapes in the above illustration, for example, would be referenced as:

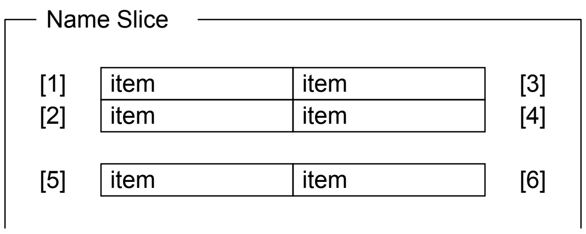
```
qty[1]
qty[2]
qty[3]
```

Shapes are numbered from top to bottom.

Similar names would be used to reference the price, description, and total shapes.

Multi-Column Shape Arrays

Multi-column arrays are numbered top to bottom, left to right; touching shapes take precedence:

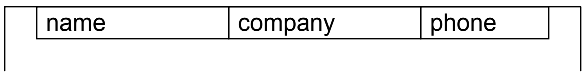


You can draw clear rectangles around related shapes to change array order, and you can 'tab around' with the ENTER key to see the sequence.

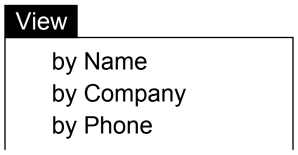
Edge Shapes

Shapes on the top or bottom edges of pages can also be named. Arrays are not allowed on edges.

The names given to edge shapes will appear in Folder View on the View menu as sorting options.



The edge above, for example, will produce a View menu in Folder View that includes these items:



The names are capitalized to look like standard items. Abbreviated names should be avoided.

Reserved Names

TYPE, IDENTIFIER, and NUMBER are special names given to shapes on the bottom edges of pages. Do not use the first two for other shapes. Use NUMBER as a name only when you want the page number assigned to that shape.

The Code Slice

Overview

The Code slice is used when you want shapes on a page to contain calculated values. Values are updated when a page is saved, closed, or printed.

Edge Values

You can use the Code slice to automatically update edge shapes. Say, for example, we had a page like this one (Value slice showing):

NAME	John Doe
ADDRESS	123 Main Street
PHONE	555-1212

And we wanted an edge that looked like this:

John Doe	555-1212

We would simply add two shapes to the edge, and name the shapes on the edge and in the body of the page like so (Name slice showing):

name	phone

	name
	phone

This makes it possible for us to sort the edges in Folder View by name and phone. It also allows us to extract the name and phone number from the body of the page like this (Code slice showing):

name	phone

The values in the name and phone number shapes in the body of the page will automatically be copied to the shapes on the edge of the page whenever the page is saved, closed, or printed.

Expressions

You can also put arithmetic expressions in the Code slice of a page. Consider, for instance, this page (Name slice showing):

qty	desc	price	total
qty	desc	price	total
qty	desc	price	total

	grandtotal

Now consider the expressions we need to add to calculate the total for each line (Code slice here):

			qty[1]*price[1]
			qty[2]*price[2]
			qty[3]*price[3]

Note that subscripts must be specified in the Code slice. You can use the Fill Down command to simplify entry of these expressions.

Functions

You can also use Perspective's statistical functions (COUNT, SUM, MAX, MIN, and AVG) in the Code slice to operate on shape arrays:

			qty[1]*price[1]
			qty[2]*price[2]
			qty[3]*price[3]

	sum(total)

Note the use of the SUM function. It operates on an entire array and does not require subscripts. See the 'Function Reference' chapter for more information regarding statistical functions.

The Update Slice

Overview

The Update slice of text shapes is used, primarily, to specify which text shapes on a form are editable, and which ones are not; non-editable text shapes — and all graphic shapes — are relegated to the background of the form.

The Update slice can also be used to create pop-up menus for text shapes, when the values allowed in those shapes are few and well-defined.

Editable and Non-Editable Shapes

All text shapes are editable, by default. To make them non-editable, we must adjust their Update slice. Consider, for example, this partial form:

Value Slice

QTY	DESC	PRICE	TOTAL
3	Widget	500.00	$1,500.00

Say that we want the quantity, description, and price to be editable, but the shaded headings and the calculated total to be non-editable.

To make this so, we switch to the Update slice:

Update Slice

yes	yes	yes	yes
yes	yes	yes	yes

Note that all text shapes are initially flagged as editable. Now, we modify the values like this:

Update Slice

no	no	no	no
yes	yes	yes	no

And we're done.

Shapes marked with a 'yes' in the Update slice will be editable when the drawing is converted to a form; those marked with a 'no' will not be.

Pop-Ups

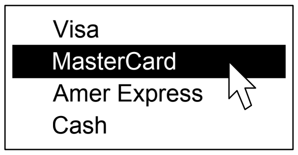
Sometimes you want to associate a pre-defined list of words or phrases with a text shape. You can accomplish this by putting a pop-up menu definition in the shape's Update slice.

A pop-up menu definition is simply a list of items separated by vertical bars (|), like so:

```
Visa | MasterCard | Amer Express | Cash
```

Up to 25 items are allowed, and any character (except the vertical bar) can appear in any item.

Perspective will display the appropriate pop-up menu when any such shape on a form is selected and then pressed with the mouse. In this case, the menu would appear over the shape like this:



The selected item, if any, will be inserted into the Value slice of the text shape when the mouse is released. If no item is selected with the mouse, the original value will be retained.

Helpful Hints

• Keep pop-up menu items short. It is difficult to scan lengthy names quickly, and long values frequently don't fit on forms and reports.

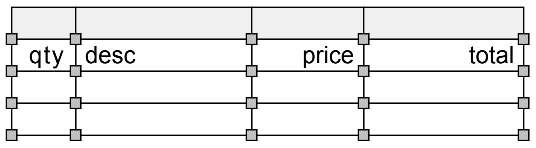
• Use pop-up menus with discretion. If only two values are allowed in a shape (such as 'yes' and 'no') you may want to use a boolean format instead of a pop-up menu. If more than ten different values are displayed on a pop-up, the menu will probably be too large to be handy.

• Design your forms a slice at a time, not shape by shape. Draw the entire form with the Value slice showing, then move on to the Format slice. When you're done with the Format slice for the whole form, move on to the Name slice, etc.

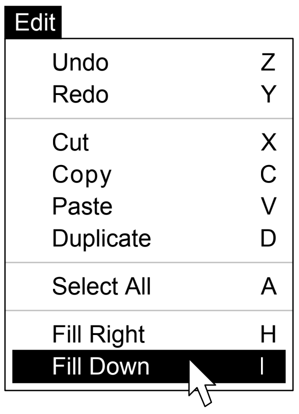
Easier Editing

Filling

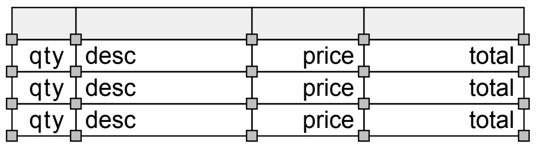
You can fill text shapes with values from nearby text shapes using Fill Down and Fill Right. Just select the appropriate shapes:



Then execute the command:



The Fill Down command copies the value from the topmost shape in each column into each of the other selected shapes in that column:



Fill Right copies the leftmost value in each row into the other selected shapes in the same row.

Helpful Hints

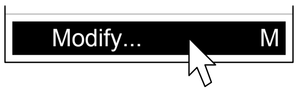
• Keep your shapes on the grid. Text shapes you want to fill do not have to touch, but their tops and left sides must be exactly aligned.

• Fill Down will automatically 'bump' subscripts in the Code slice. Just enter the appropriate values in the top row, then select and Fill Down.

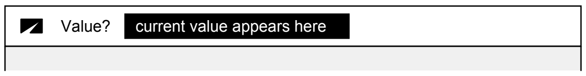
Modifying

Text shapes on pages are normally sized so that the largest expected value fits in the Value slice. This often results in shapes that are too narrow for identifiers in the Name slice or complex formulas in the Code slice. The Modify command is provided for editing values in narrow shapes.

When you select a text shape and execute the Modify command from the very bottom of the Shape menu, like so:



A dialog, similar to this one, appears:



The prompt varies depending on the slice that is currently displayed.

The default value is the first line of the current slice of the selected shape.

After modifying the value with the usual editing techniques, press ENTER to update the shape, or exit the dialog to cancel.

Helpful Hints

• Do not use the Modify command on multi-line text shapes; if you do, lines other than the first line will be lost.

• Use short names for shapes on pages, except for edge shapes whose names will appear on the View menu as sort options.

• Design the graphic layout of your pages first, filling in sample data as defaults and to check shape widths. Then fill in the Format, Name, Code, and Update slices.

• Use the Duplicate, Copy, Paste, Fill Right, and Fill Down commands to minimize typing.

5

PROGRAMMING

Introduction

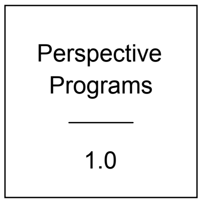
Overview

You can extend the functionality of Perspective by designing forms that can be added to the Page menu in Folder View. You can also write your own functions that can be added to the File and Special menus and run as if they were commands native to Perspective.

You can also execute the functions you write by typing their names into the Compute dialog. This is a handy way to run conversion, end-of-year, and other specialty programs that you don't want cluttering up the menus.

Program Drawers

Programs are kept in a drawer like the one shown below. The group must be 'Perspective' and the name must be 'Programs'.



The version can be blank or any value of your choosing. The program drawer with the most recent version is used.

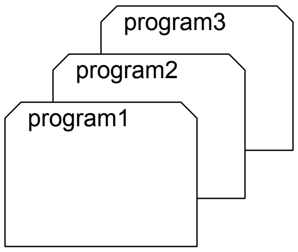
The most recent version is the one that sorts highest. A drawer with a '2.0' version, for example, will supersede one with a '1.9' version. But note that a '2.0' version will also supersede a '10.0' version because the leading digit is larger.

It is best to keep only one program drawer in a cabinet. If you must keep multiple versions, pad your versions with zeros on the left to make them all the same length so they will sort properly.

Note that programs are accessible only when the cabinet containing the appropriate program drawer is open. This allows you to group your programs into 'systems', cabinet by cabinet, and also lets you have up to three systems active at any time (one on each tab).

Program Folders

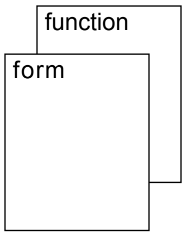
A program is a collection of forms and/or functions that performs a service. Each folder in a program drawer should be thought of as a separate and distinct program.



You can have any number of program folders. When two or more program folders in a drawer have the same name, the latest version is used.

Program Pages

Program folders typically contain two kinds of pages: forms and functions, as shown here:



Form pages are like records in a traditional programming language. They contain named text shapes that can be referenced in your programs, and are used as models to make actual pages. A typical program might include two forms: one for Orders and another for Mailing Labels.

Function pages contain executable statements that operate on pages made from form pages. A typical program might include a function that makes Mailing Label pages from the names and addresses found on Order pages.

You can have any number of form and function pages in a program folder.

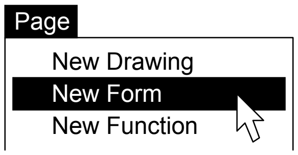
Forms

Overview

When programming, you should think of forms in two ways: as templates from which other pages are made, and as pages with named text shapes that you can access in the functions you write.

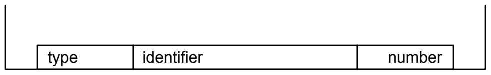
Creating a Form

You can make new forms in a program folder with the New Form command on the Page menu:



In program folders, forms always behave like drawing pages so you can modify their design without converting them to drawing pages. In all other folders, forms act like forms.

Every form you make will have three standard shapes on its bottom edge (Name slice showing):



The first shape is the TYPE. This shape tells Perspective what kind of page this is. Its value is the word 'form' and should not be modified.

The second shape is the IDENTIFIER. Enter the name that you want the form to have in the Value slice of this shape. Form names must start with a letter, cannot contain spaces, and must be 40 or less characters long. All form names in a program folder must be unique. The full name of a form is the folder name, a colon, and the form name:

```
foldername:formname
```

You can use this syntax in the rare case that you need to refer to a form in another program folder.

The third shape is the NUMBER and is used only if and when you renumber the pages in the folder.

Graphics

Forms can contain any number of graphic shapes arranged in whatever way suits your purpose. You can create and edit graphic shapes on forms just like you do on drawing pages.

Note that when a new page is created from a form, however, the graphics are locked in the background and are no longer editable.

Note also that graphic shapes cannot be manipulated with the programming language.

Text

Forms typically contain one or more text shapes, in the body of the page or on its edges. The slices of these shapes operate as usual, but the following considerations should be noted:

The Value slice can be used to specify default values for each shape which will appear when a new page — based on the form — is created.

The Format slice is used by Perspective to determine the data type of each shape.

The Name slice determines which shapes can be programmatically manipulated, since only named shapes can be referenced in functions you write.

The Code slice is normally used to extract edge values and to calculate on-page totals. Note that calculated values may appear on the edge as well as in the body of forms.

The Update slice determines the overall behavior of the form, including any pop-up menus.

Helpful Hints

• Use the 4 lpi fonts in the body of your forms. They box nicely, shade nicely, and sit on the large blue grid. This also leaves larger and smaller fonts available for titles and 'the fine print'.

• Remember that ENTER-key tabbing is defined by the layout of your text shapes. Keep related shapes together and in logical order.

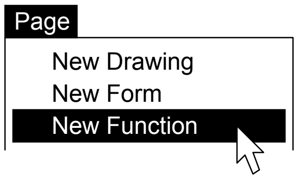
Functions

Overview

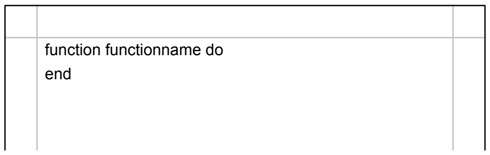
Function pages contain executable code. Each one contains a single function that can be executed from a menu, run from the Compute dialog, and/or called from other functions.

Creating a Function

You can make new function pages only when you are in a program folder. To do so, select the New Function command from the Page menu:



Each function page consists of a single, large text shape in the center of the page:



```
function functionname do
end
```

This is where you enter your code. The grid and the first and last lines of the function are automatically provided for your convenience.

The function name is entered on the first line immediately following the word 'function'. Function names can be up to 40 characters in length, must begin with a letter, and cannot contain spaces. They must be unique within their program folder. The full name of a function is:

```
foldername:functionname
```

The bottom edge of each function page contains three shapes that are visible only in Folder View. The first is the TYPE (function). The second is the IDENTIFIER (the function name is extracted automatically from the code and placed here). The third is the NUMBER and is used only if and when you renumber the pages in the folder.

Function Syntax

The general syntax of a function is:

```
FUNCTION functionname DO
  { statements go here }
END
```

The capitalized words are keywords. You can type them in upper, lower, or mixed case letters.

Comments

Comments must be enclosed in { curly braces }. Multi-line comments are not supported.

Helpful Hints

• Functions automatically indent and do not wrap. Keywords display in red; comments in blue.

• Use all lower case letters (it's simpler and faster) and don't abbreviate words in function names (it makes them harder to read).

• When commenting more than one line of code, start at the bottom to avoid the annoying side effects of the auto-indention feature.

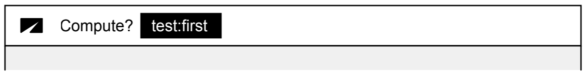
Example

And now it is time to write your first function.

• Make a program folder called 'test'.
• Open the folder and make a function page.
• Enter the code shown here:

```
function first do
  warning('Hello, World!')
end
```

Compile the program using the Compile command on the Special menu, then Run the program from the Compute dialog, using its full name, like this:



Compute? test:first

Ta da! You are now a Perspective programmer.

Events

Overview

Events are functions that Perspective executes automatically at certain times. There are four kinds of events, as described below.

You can include any or all of these events in any program folder, as required.

Initialization Events

When first starting up, and whenever the Compile command is executed, Perspective executes all the INITIALIZEEVENT functions that are found in any program folder.

The syntax is:

```
function initializeevent do
  { initialization code goes here }
end
```

This function is typically used only to define and initialize global variables.

See the 'Variables' topic, later in this chapter, for more information on global variables.

Menu Events

All MENUEVENT functions are executed when the user clicks in the menu bar.

The syntax is:

```
function menuevent do
  { menu code goes here }
end
```

You can use this function to add items to the File, Special, and Page menus. Normally, the current view, the current drawer, and the current folder are checked to determine what action, if any, should be taken.

See the next page, and the 'Interface Functions' topic of the Function Reference chapter, for further information on modifying menus.

New Page Events

The appropriate NEWPAGEEVENT function is executed whenever a new page is created, just before it is displayed on the screen.

The syntax is:

```
function newpageevent(p) do
  { new page code goes here }
end
```

Note that the page is passed to the function so you can operate upon it. You should check the type and identifier of the page and take any appropriate action at this time.

This function is normally used to initialize dates, serial numbers, and other text shapes on a page.

Finalization Events

FINALIZEEVENT functions are executed when the Compile command is executed, a cabinet is closed, and just before Perspective shuts down in response to a Quit command.

The syntax is:

```
function finalizeevent do
  { finalization code goes here }
end
```

This function is normally used to close and/or dispose of any global variables allocated during the INITIALIZEEVENT.

Helpful Hints

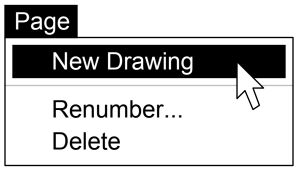
• The order in which Perspective processes program folders — and, therefore, the event functions within them — cannot be predicted. Take care, therefore, to make sure that there are no inter-dependencies between program folders.

• Program folders should be independent of one another, in any case, so that programs can be moved easily from one system to another.

Menus

Adding Forms

New pages are created with the 'New...' commands on the Page menu in Folder View:

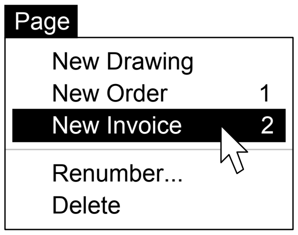


You can add your own 'New...' commands to the Page menu like this:

```
function menuevent do
  additem(pagemenu,'New Order','1',order)
  additem(pagemenu,'New Invoice','2',inv)
end
```

where 'New Order' is the text to display on the menu, '1' is the shortcut key, and 'order' is the identifier on the bottom edge of the form, etc. Use the full form name if the form is in a different folder than the MENUEVENT function.

After compiling, the Page menu in Folder View will have an additional item in the 'New' section:



When the command is executed, a copy of the form will be added to the current folder.

If the current folder is not a program folder, the page will behave as a form: all graphic shapes will be fixed in the background, and only updatable and pop-up text shapes will be editable.

Helpful Hint

• 0 through 9 are available for shortcut keys.

Adding Functions

You can add your own functions to the File and Special menus. Normally, only printing functions are added to the File menu; any kind of function can appear on the Special menu. For example:

```
function menuevent do
  additem(filemenu,'Print
Form','',&print)
  additem(specialmenu,'Hello','',&first)
```

Here, 'Print Form' and 'Hello' are the items as they are to appear on the menus; the empty string indicates that no shortcut keys are desired; and 'print' and 'first' are the names of the functions to be executed when the commands are selected from the menus. The full name of the function must be used if it is not in the same program folder as the MENUEVENT function.

Note the '&' symbol preceding the function names. This is required; it tells Perspective to save the name of the function, rather than execute it immediately.

Items are added at the bottom of the 'Print' section of the File menu, and to the very bottom of the Special menu.

Helpful Hints

• You can avoid cluttering up the menus by adding items only when you are in certain views, drawers, or folders. See page 85 for an example.

• You can also avoid menu clutter by running test, conversion, and other one-time programs from the Compute dialog. Don't forget to use the function's full name (foldername:functionname).

• The ADDSEPARATOR function puts gray lines between items you add to a menu. See the Function Reference chapter for more information.

• See 'Conditional Execution' in this chapter, and 'Interface Functions' in the next chapter, for more information regarding modifying menus, adding separators, and for various ways of determining the current view, drawer, and folder.

The Programming Language

Overview

Perspective's built-in programming language is similar to traditional programming languages, such as FORTRAN, BASIC, COBOL, PASCAL, and C.

It is more consistent than FORTRAN, however, and more powerful than BASIC. It is much less wordy than COBOL, slightly more elegant than PASCAL, and significantly more intuitive than C. And it is smaller than any or all of these, making it decidedly easier to learn and use.

The remainder of this chapter describes the language in detail. Here's a synopsis:

Literals, Constants, Variables

This topic covers the basic and extended data types supported by Perspective. It tells you how to define literals, constants, and local, global, and on-page variables of each type.

Expressions and Assignments

Under this heading, expressions in general (and boolean, numeric, and string expressions in particular) are discussed. The assignment statement is also covered in excruciating detail.

Conditional Execution

Statements that cause other statements to be executed conditionally are discussed in this topic. The various forms of IF and CASE statements are defined and described.

Flow Control and Iteration

The four different ways loops can be written in Perspective are discussed under this heading. Generic LOOPs, as well as REPEAT, WHILE, and FOR loops — including a special form of the latter for file system traversal — are covered.

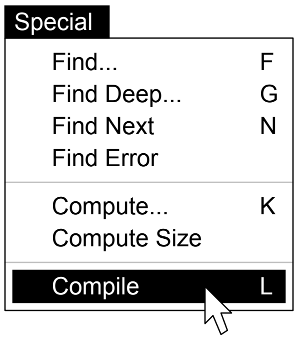
Function Calls

Function calls and related topics (including parameter passing, return values, and recursion) are discussed last.

Compiling

Normally, Perspective compiles functions on demand. The first time a function is called it is loaded from disk and compiled into memory.

It is frequently necessary during development, however, to force recompilation, like so:

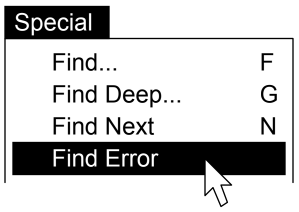


This command saves the current page, resets all global variables, refreshes the available function list, and tells Perspective that all new and existing functions should be recompiled when called.

Debugging

Perspective includes all the debugging tools used by true professionals everywhere.

For example, the Find Error command (available in program drawers only) positions you on the offending line of code following any error:



You can examine global variables using the Compute command, and you can display local variables with the MESSAGE, WARNING, and ERROR functions described in the next chapter. You can also insert BEEP, BLINK, and PAUSE functions at critical places in your code to trace program flow within and across routines.

Literals, Constants, and Variables

Data Types

Perspective recognizes and processes the five basic data types shown in the chart below.

TYPE	DESCRIPTION
boolean	true or false
number	±922337203685477.5807
string	text, up to 2 billion characters
date	01/01/1900 thru 12/31/9999
time	00:00:00 thru 23:59:59

The language also includes operators to process the four extended data types shown here:

TYPE	DESCRIPTION
page	a collection of shapes
folder	a collection of pages
drawer	a collection of folders
cabinet	a collection of drawers

Literals

Use TRUE or FALSE, upper or lower case, for boolean literals. Combine the digits 0 through 9 and a single, optional decimal point for numerics. Enclose strings in single quotes, using two successive quotes for quotes within strings.

Constants

A list of predefined constants is located in the Appendixes. You can use global variables, discussed below, to define your own constants.

Variables

Three kinds of variables are supported: LOCAL (within a function), GLOBAL (within a program folder), and ON-PAGE (named text shapes on the edge or body of a page).

Locals and globals are untyped until a value is assigned; they then remain that type. On-page variable types are deduced from their formats.

Variable names are case-insensitive, must start with a letter, and may not contain spaces.

Local Variables

Local variables can be defined at the top of any function using the LOCAL statement:

```
function second do
  local x,y,z
end
```

Local variables can be accessed only within the function in which they are defined.

Global Variables

Global variables are normally defined in the INITIALIZEEVENT of a program, like so:

```
function initializeevent do
  global pi
  pi:=3.1415
end
```

Global variables are shared by all the functions in a program folder.

On-Page Variables

On-page variables are defined on forms when a name is given to a text shape. They can be accessed like this:

```
p.shapename
p.shapename[subscript]
```

where P is a page, and SHAPENAME is the value entered in the Name slice of the shape on the page. SUBSCRIPTs apply to shape arrays only.

When a page is closed, only edge shapes are accessible. When a page is open, body shapes can be referenced.

Helpful Hints

• Keep your variable names short like the pros do.

• You can reference edge shapes on open pages by specifying zero as the subscript. This is rarely necessary because edges usually contain values copied from shapes on the body of a page.

Expressions and Assignments

Expressions

Expressions are phrases that can be reduced to a value of some type. Perspective uses 'short circuit' evaluation on all expressions, considering only as much as necessary to deduce the result.

Boolean Expressions

Boolean expressions reduce to TRUE or FALSE. They are made of boolean literals, constants, variables, functions, and the following symbols:

SYMBOL	DESCRIPTION
<	less than
<=	less than or equal to
=	equal to
<>	not equal to
>=	greater than or equal to
>	greater than

The keywords AND, OR, XOR, and NOT can also be used, and parentheses can be included to enforce a specific order of evaluation (innermost first). String comparisons are not case-sensitive.

Numeric Expressions

Numeric expressions include literals, constants, variables, functions, and the following symbols:

SYMBOL	DESCRIPTION
+	add
-	subtract
*	multiply
/	real division
^	raise to power

The keywords DIV and MOD can also be used as infix operators in expressions for integer division and remainder of integer division, respectively. Expressions in parentheses are evaluated first.

String Expressions

String expressions include literals, constants, variables, and functions that reduce to strings. The '+' symbol is used to concatenate strings.

Assignments

Values are assigned to variables like this:

```
b:=true
n:=3.1415
s:='This is it'
d:=ctd('01/01/1998')
t:=ctt('12:01:01 am')
p:=createpage(myfrm)
```

In all cases, the value on the right is assigned to the variable on the left.

Note that the assignment operator is ':=', not a mere '='. This may seem inconvenient at first, but it removes ambiguity from the language and allows statements like:

```
x:=y:=z:=1
```

and

```
b:=x=y
```

and even

```
if b:=x=y do message(b) end
```

to be interpreted and executed correctly.

Note also that conversion functions are required to convert strings into date and time values, since date and time literals are not supported.

Note that you can assign variables to other variables, in the same manner:

```
q:=p
```

assuming that the receiving variable (q in this case) is either untyped at the time, or the same type as the source (p).

The source variable (p) is not affected.

Helpful Hint

• Think of ':=' as an arrow pointing left. We know it doesn't look like one, but that's what it does.

Conditional Execution

Overview

Perspective's programming language includes two kinds of conditional statements: IF and CASE.

If Statements

The syntax for IF statements is:

```
if expr do
  { statements go here }
else
  { statements go here }
end
```

EXPR is a boolean variable or an expression yielding a boolean result.

If EXPR is true, the first group of statements is executed; if EXPR is false, the second group of statements is executed. In either case, execution eventually continues with the statement following the IF statement.

The ELSE clause (and the second group of statements) are optional and may be omitted.

The END is required in all cases.

Helpful Hints

• Avoid ELSE clauses except in trivial 'this or that' cases. They make code ugly and more difficult to read and understand.

• Avoid nested IFs. They are hard to read and difficult to debug. This is not a common position, we know, but we truly believe that in all but a few exception cases, single-level IFs are preferable. As our chief programmer so eloquently puts it: 'Two deep is too deep'. Now if you don't think that programming like this is possible, consider this: in all of Perspective's 25,000 lines of source code, there are only seven nested IF statements.

• Handle exception situations first, and use the EXIT statement to leave the function once a particular situation has been handled. This keeps program flow top-to-bottom while making normal processing unconditional and easy to debug.

Case Statements

The syntax for CASE statements is:

```
case expr1 do
  when expr2 do { statements here } end
  when expr3 do { statements here } end
  ...
  when exprN do { statements here } end
else
   { statements go here }
end
```

EXPR1 is any boolean, number, date, time, or string variable, or any expression yielding one of these basic types.

EXPR2 thru EXPRn are variables or expressions of the same type as EXPR1.

If EXPR2 equals EXPR1, the statements in the associated DO block are executed. Execution then continues with the statement following the entire CASE statement.

If EXPR3 equals EXPR1, the statements in the associated DO block are executed. Execution then continues with the statement following the entire CASE statement.

If none of the expressions match, the statements following the ELSE clause are executed. The ELSE clause (and associated statements) are optional and can be omitted.

Helpful Hints

• Try to keep your WHEN clauses on a single line, because deep CASE statements are hard to read and difficult to debug. Create functions, as necessary, and call them from the WHEN clauses to achieve this end.

• A series of IF or trivial IF-ELSE statements often requires fewer lines and provides a more elegant solution than a CASE statement — even though a CASE statement may be the 'proper' construct to use. CASE statements work best when you are simply 'dispatching' control to one of several alternate routines.

Flow Control and Iteration

Overview

Perspective's programming language includes three statements for flow control: BREAK, CONTINUE, and EXIT. Four statements for iteration are also supported: generic LOOPs, REPEAT loops, WHILE loops, and FOR loops.

Flow Control

You can use BREAK, CONTINUE, and EXIT statements to change the normal sequence of execution in any loop or function.

The BREAK statement terminates a loop immediately. Program execution continues with the statement following the end of the loop.

The CONTINUE statement skips the remainder of the current iteration of the loop, resuming execution with the next iteration of the loop.

The EXIT statement ends execution of a function and returns control to the caller. Any and all loops in progress are terminated.

Generic Loops

The syntax of a generic loop is:

```
LOOP
  { statements go here }
END
```

The loop will execute indefinitely until a BREAK or EXIT statement is encountered.

Repeat Loops

The syntax for a repeating loop is:

```
REPEAT
  { statements go here }
UNTIL condition
```

REPEAT loops iterate until the CONDITION is TRUE, and always execute at least once.

The CONDITION can be any boolean variable or any expression that yields a boolean result.

While Loops

The syntax here is:

```
WHILE condition DO
  { statements go here }
END
```

WHILE loops iterate until the CONDITION is FALSE; the enclosed statements may execute zero, one, or more times.

The CONDITION can be any boolean variable or any expression that yields a boolean result.

For Loops

Perspective supports two forms of FOR loops:

```
FOR var:=num1 TO num2 BACKWARDS DO
  { statements go here }
END
```

This is a standard FOR loop. VAR must be a local variable; NUM1 and NUM2 can be any numeric variable or expression yielding a numeric result; the BACKWARDS keyword is optional.

The second form of a FOR loop is:

```
FOR var IN expr BACKWARDS DO
  { statements go here }
END
```

This form is used to iterate through the file system. VAR must be a local page, folder, or drawer variable; EXPR is a corresponding folder, drawer, or cabinet variable, or an expression that yields one. BACKWARDS is optional.

Helpful Hints

• You can terminate infinite loops with:

CTRL or ALT + END

• See the next chapter for additional information on manipulating the file system.

Function Calls

Defining Functions

As you have seen, functions take the form:

```
FUNCTION functionname DO
  { statements go here }
END
```

This is the simplest case. Functions can also take in parameters and return a result.

Functions with Parameters

Functions can accept any number of parameters. The names of parameters (as they are known to the function) are listed in parentheses, like so:

```
FUNCTION functionname(p1,p2,p3) DO
END
```

where P1, P2, and P3 represent parameters of any type to be passed to the function. Any name that is not used as a local variable can be used.

Parameters are passed by reference; that is, a pointer to the original variable is used. Changing the value of a parameter, therefore, changes the value of the original variable.

Functions with Return Values

Functions that return a value are defined thus:

```
FUNCTION functionname RETURNS result DO
END
```

RESULT represents a variable of any type to be returned by the function. Any valid name that is not used as a parameter or local variable name can be used.

The type and value of the result is undefined until an assignment is made to it within the function.

Only a single return value is allowed.

Helpful Hint

• Use short but readable names for parameters. Call all results the same thing. 'Result' works well.

Calling Functions

You can call a function like this one:

```
function first do
  warning('Hello, World!')
end
```

simply by referring to its name:

```
function second do
  first
end
```

If the called function accepts parameters, like this one does:

```
function third(w) do
  warning(w)
end
```

you must call it with appropriate parameters:

```
function fourth do
  third('Hello, World!')
end
```

If the called function returns a value, like this:

```
function fifth returns result do
  result:='Hello, World'
end
```

you can either save the result, pass it to another routine, or discard it:

```
function sixth do
  local m
  m:=fifth          { save result in m }
  warning(fifth)    { pass result }
  fifth           { discard result }
end
```

In the last case, no meaningful action is taken.

Helpful Hint

• Perspective is fully recursive, so you can write routines that call themselves. You probably won't need to do this, but it's nice to know you can.

Helpful Hints

Design Guidelines

When designing a system with Perspective, keep the following in mind:

• Get back to basics. Design your application as if it was a completely manual system with real cabinets, drawers, folders, and pages. This will make your system easy to think about, easy to explain to others, and very easy to implement.

• Divide and conquer. Keep the number of pages per folder reasonably small by grouping things in natural and obvious ways. Some information is best stored alphabetically, with one folder for A, another for B, and so forth. Other data can be divided up geographically: a Michigan folder, a Texas folder, etc. You can also break things up time-wise, using separate drawers or folders for each year, month, week, or day. Always keep in mind that any folder can — and just might — be printed in its entirety.

• Think hierarchically. You can manage large numbers of pages by keeping detail batches small and summarizing at intermediate levels. A system that handles 50,000 orders per year, for example, can be configured with weekly and monthly summaries of daily detail folders that contain only 200 pages each.

• A small success is better than a big failure. Build systems incrementally. Develop a drawing, print it off, and send it to somebody. Put up a couple of forms, and actually use them for a few days. Write a program or two, and see if you like the results. Remember that great systems are usually grown, not manufactured.

• Plan on doing everything at least three times. You can afford to experiment with systems in Perspective because development is so fast. Prototype everything and assume that everything is subject to change. Never say, 'Done'. Always say, 'It gets better every time we touch it'.

• Keep it simple. Perfection is attained when there is nothing left to remove. Solve problems, whenever you can, by deleting something rather than adding something.

Programming Guidelines

When implementing a system with Perspective, keep the following in mind:

• Put your forms in the left work area, your functions in the middle, and run your tests in the right one. This makes it easy to check names on forms while you are writing code, and also lets you find errors quickly without opening and closing a lot of drawers, folders, and pages.

• Add a menu item called 'Test' to the bottom of the Special menu, with '0' as the shortcut key; make it refer to the routine you are currently working on. This lets you compile and run with a quick ALT-L followed by an ALT-0:

ALT + L then ALT + 0

• Design your forms first, a slice at a time. Don't jump around between the slices, and always return to the Value slice when you are finished.

• Do as much processing as possible right on the form — put defaults in the Value slice, formats in the Format slice, and formulas (for both body and edge shapes) in the Code slice. You can use your own functions in the Code slice, but do so with care: it makes your forms less transportable.

• Name every shape you might want to reference in a program. Keep names short but meaningful.

• Test your system continuously. Don't write more than a few lines of code without testing. Build complicated routines a step at a time, and test after each step.

• Include status messages at the beginning and end of all functions, and in the middle of all long-running ones. A few seconds seems like a long time when nothing appears to be happening.

• Look in the Perspective Programs drawer for a wide variety of sample programs, and in the Perspective Samples drawer for additional forms and drawings that may be of use to you.

Helpful Hints

System-Assigned Keys

Many systems automatically assign identifying numbers to forms such as orders, invoices, etc. You can do this with Perspective by storing the last-used number in your program folder on a form you create expressly for that purpose.

Say, for example, you had a form called ORDER with a shape on it called ORDERNUMBER. Say you also created a form to hold your last order number called KEYFORM with a shape called LASTORDER on it. You could then install a NEWPAGEEVENT like the one below to assign a unique order number when the page is created.

```
function newpageevent(p) do
  lock('c')
  openpage(keyform)
  inc(keyform.lastorder)
  p.ordernumber:=keyform.lastorder
  savepage(keyform)
  closepage(keyform)
  unlock('c')
end
```

This function begins a transaction by locking the cabinet. It then opens the page with the last-used key on it. After incrementing the key by one, it assigns the new value to the appropriate shape on the newly created page (passed to the function as 'p'). The page with the last-used key is then saved (for future use) and closed (to free memory). The new page will be saved in the normal course of events. Then the cabinet is unlocked, and the transaction ends.

Note that forms in program folders can be accessed as if they were global variables. You simply have to open them up, using the identifier on the bottom edge as a variable name. Closing such pages when you are through with them frees memory for other purposes.

Note also the use of the LOCK and UNLOCK functions in this example. These guarantee that you will have exclusive access to the page (actually, to the whole cabinet) during the entire transaction. See 'File System Functions' in the next chapter for more information on locking.

Memory Leaks

If you have open cabinets, drawers, folders, or pages when an error occurs, a small amount of memory will not be properly freed for later use.

Normally, this is not a problem, because the amount lost is typically insignificant compared to the amount of virtual memory available. In any case, you can simply restart Perspective (not the whole machine) to recover all lost memory.

You can check for 'memory leaks' in your programs by holding the SHIFT key when you quit. A dialog will appear, indicating the number of memory blocks that were not properly freed.

If you start Perspective fresh, encounter no errors, and close every cabinet, drawer, folder, and page that you open in your code, you should have zero leakage.

Handy Key Combinations

To convert drawings to forms (and vice-versa):

CTRL or ALT + SHIFT + F

To abort any function:

CTRL or ALT + END

To check for memory leaks:

SHIFT when quitting

To display elapsed time for a function:

SHIFT when running a function from a menu

Note that this last combination does not work when a function is run from the Compute dialog.

6

FUNCTION REFERENCE

Introduction

Overview

This chapter describes Perspective's built-in functions. They are presented in logical order, with File System and Interface functions first. Date and Time functions are next, with String, Number, and Statistical functions following. A handful of Text File and Miscellaneous functions are grouped together at the end.

A complete, alphabetical list of all the functions (with their parameters and return values) can be found in the Appendixes.

File System Functions

The function library includes a variety of functions for working with cabinets, drawers, folders, and pages, much as you do on the screen. Functions to open, close, sort, count, find, copy, create, rename, and delete file system objects are provided.

Additional functions to update, save, and print pages are also included, together with functions for working with 'loose' pages in memory.

All file system functions provide the necessary feedback regarding errors that may occur.

Interface Functions

Perspective's interface is designed to be complete enough so that extensive 'interactive' programming is not necessary. Functions to manipulate key interface components are, however, provided in the library.

These include routines to beep and blink, to display status, warning, and error messages, and to conduct dialogs with a user. Other functions allow you to refresh the menu bar and/or the work area to keep the screen up-to-date.

Interface functions are also provided so you can determine the current view and gain access to the current cabinet, drawer, folder, and page. You can use these functions to examine and set the current view and selection, as well as add items to the File, Special, and Page menus.

Date and Time Functions

Functions for getting the current date and time, and for extracting the components thereof, are provided. Routines for converting dates and times into formatted strings are included, as well.

String Functions

All of the usual string manipulation routines can be found in the library. You can convert from ASCII to characters, and back. You can get the length and insert, delete, and extract characters from the left, middle, or right of a string. You can convert case, trim leading or trailing characters, and search for a substring. You can also convert dates, times, and numbers into strings.

Number Functions

An ample supply of numeric functions, including routines to round, bump, and compare can be found here. Logarithms and trigonometry are also supported, with all the usual conversion routines.

Statistical Functions

Five special 'set-level' functions are provided for calculating counts, sums, maximums, minimums, and averages of arrays on pages.

Text File Functions

Communication between Perspective and other systems is accomplished via standard text files. You can import and export data with routines to create, delete, read, write, rename, remove unwanted characters from, and parse text files.

Miscellaneous Functions

Various boolean, timer, random number, and utility functions complete the library.

Playing Around

You can experiment with many of Perspective's functions by executing them in the Compute dialog. Just select the Compute command from the Special menu and play away.

Functions by Category

File System

closecabinet
closedrawer
closefolder
closepage
copypage
createpage
deletedrawer
deletefolder
deletepage
destroypage
duplicatedrawer
duplicatefolder
duplicatepage
filepage
finddrawer
findfolder
findpage
getdrawercount
getfoldercount
getpagecount
lock
newdrawer
newfolder
newpage
opencabinet
opendrawer
openfolder
openpage
printpage
renamedrawer
renamefolder
savepage
sortdrawers
sortfolders
sortpages
unfilepage
unlock
unlockall
updatepage
xcx

Interface

additem
addseparator
beep
blink
dialog
drawmenubar
drawworkarea
error
getfirstselecteddrawer
getfirstselectedfolder
getfirstselectedpage
getlastselecteddrawer
getlastselectedfolder
getlastselectedpage
getselecteddrawercount
getselectedfoldercount
getselectedpagecount
message
selectdrawer
selectdrawers
selectfolder
selectfolders
selectpage
selectpages
viewcabinet
viewdrawer
viewfolder
viewpage
warning
xcc
xcd
xcf
xcp
xcv

Date and Time

ctd
ctt
formatdate
formattime
getdate
getday
getdayofweek
gethours
getminutes
getmonth
getseconds
gettime
getyear

String

chr
cts
delete
fillleft
fillright
insert
lcase
left
len
mid
ord
pos
removeleading
removetrailing
right
ucase

Text File

deletefile
gettoken
gnt
gpt
newfile
readfile
removeunwanted
renamefile
writefile

Number

abs
arccos
arcsin
arctan
cos
ctn
dec
degreetoradian
formatnumber
frac
inc
log
max
min
radiantodegree
round
rounddown
roundup
sin
sqrt
tan
trunc

Statistical

avg
count
max
min
sum

Miscellaneous

ctb
formatboolean
getticks
mcm
pause
random
randomize
upperbound

File System Functions

Overview

You can manipulate cabinets, drawers, folders, and pages in programs, much like you do on the screen, using the functions described here.

File system errors that occur when a program is running are not displayed. Use this syntax:

```
result:=XCX
```

to retrieve a full error message after an error.

The Big Picture

You can loop through an entire cabinet — looking at every drawer, folder, and page — like so:

```
function scan do
  local c,d,f,p
  c:=opencabinet('c')
  for d in c do
    opendrawer(d)
    for f in d do
      openfolder(f)
      for p in f do
        openpage(p)
          { process page here }
        closepage(p)
      end
      closefolder(f)
    end
    closedrawer(d)
  end
  closecabinet(c)
end
```

This is the general case. Your programs will normally require only a subset of this loop.

Attributes

You can look at the name of an open cabinet using the syntax 'c.name' (no quotes). All groups, names, and versions can be accessed similarly. Named shapes on page edges can also be examined, and you can check any object's 'sel' flag to see if it is selected. All these attributes are read-only. Body shapes are read-write and can be accessed when a page is open.

Opening

You open a cabinet using this function:

```
result:=OPENCABINET(cabinetname)
```

The CABINETNAME parameter is usually a letter, like 'A' or 'C', but can be any name defined in the configuration file. It must be enclosed in quotes. The result is either NIL or a cabinet variable.

Drawers, folders, and pages are opened using:

```
result:=OPENDRAWER(drawer)
result:=OPENFOLDER(folder)
result:=OPENPAGE(page)
```

These functions return either TRUE or FALSE to indicate success or failure, respectively.

Closing

When you are done with a cabinet, drawer, folder, or page, you must close it with one of these:

```
result:=CLOSECABINET(cabinet)
result:=CLOSEDRAWER(drawer)
result:=CLOSEFOLDER(folder)
result:=CLOSEPAGE(page)
```

The result is a boolean and is usually discarded.

Sorting

Drawers, folders, and pages can be sorted before processing using the functions below.

```
SORTDRAWERS(cabinet,sequence)
SORTFOLDERS(drawer,sequence)
SORTPAGES(folder,sequence)
```

The specified CABINET, DRAWER, or FOLDER must be open before sorting.

SEQUENCE is a case-insensitive string naming the item you want to sort by. Valid values are 'group', 'name', and 'version' for drawers; 'name' and 'version' for folders; and the name of any edge shape for pages. You can sort one item within another by sorting the minor item first.

File System Functions

Counting

You can get drawer, folder, and page counts using these functions:

```
result:=GETDRAWERCOUNT(cabinet)
result:=GETFOLDERCOUNT(drawer)
result:=GETPAGECOUNT(folder)
```

The specified CABINET, DRAWER, or FOLDER must be open before executing the function.

Finding

You can locate drawers, folders, and pages without looping with the functions below:

```
result:=FINDDRAWER(cabinet,g,n)
result:=FINDDRAWER(cabinet,g,n,v)
result:=FINDFOLDER(drawer,n)
result:=FINDFOLDER(drawer,n,v)
result:=FINDPAGE(folder,shapename,value)
```

The specified CABINET, DRAWER, or FOLDER must be open before executing the function.

G, N, and V are short for GROUP, NAME, and VERSION and must be enclosed in quotes. If V is omitted, the one that sorts highest is used.

SHAPENAME must be enclosed in quotes.

The result is either NIL or an unopened drawer, folder, or page, depending on the function called.

Copying

You can make copies of drawers, folders, and pages on disk using the functions listed here:

```
result:=DUPLICATEDRAWER(cabinet,drawer)
result:=DUPLICATEFOLDER(drawer,folder)
result:=DUPLICATEPAGE(folder,page)
```

You can specify where to place the copy like so:

```
result:=DUPLICATEPAGE(folder,page,after)
```

The new item will be placed following the item represented by the AFTER parameter.

Creating

You can make new drawers, folders, and pages with the functions shown here:

```
result:=NEWDRAWER(cabinet,g,n,v)
result:=NEWFOLDER(drawer,n,v)
result:=NEWPAGE(folder,form)
```

The specified CABINET, DRAWER, or FOLDER must be open. FORM is the name of a form.

You can also specify which existing item to place the new item after:

```
result:=NEWDRAWER(cabinet,g,n,v,after)
```

The result is NIL or a drawer, folder, or page.

Renaming

You can rename drawers and folders like this:

```
result:=RENAMEDRAWER(drawer,g,n,v)
result:=RENAMEFOLDER(folder,n,v)
```

The specified DRAWER or FOLDER can be open or closed.

G, N, and V are short for GROUP, NAME, and VERSION and should contain the new values.

The result is a boolean indicating whether or not the operation was successful.

Deleting

You can delete drawers, folders, and pages using the functions described below:

```
result:=DELETEDRAWER(drawer)
result:=DELETEFOLDER(folder)
result:=DELETEPAGE(page)
```

The specified DRAWER, FOLDER, or PAGE can be open or closed.

The result is a boolean indicating whether or not the operation was successful. If successful, the variable will contain NIL after execution.

File System Functions

Page Functions

The file system includes a number of special functions that apply only to pages. They are all described on this page.

Updating

Calculated values on pages are usually evaluated only when a page is saved or printed. You can force a 'run' of the page's Code slice like this:

```
result:=UPDATEPAGE(page)
```

The result is a boolean value that indicates success or failure, and is usually discarded.

Saving

If you make changes to a page in a program, you must save the page using the following function:

```
result:=SAVEPAGE(page)
```

Programmatic changes — unlike the changes made by users — are not automatically saved.

The result is boolean. Failure may indicate that the page has been updated by another user.

Printing

You can print a page in this way:

```
result:=PRINTPAGE(printer,page,auto)
```

The PRINTER parameter is a string identifying the target printer. It can be any name defined in the configuration file. Use 'default', in quotes, to print on the default printer of the current machine.

The PAGE is a variable containing the page you want to print. The page must be open.

The AUTO parameter indicates how the paper will be fed to the printer. Specify TRUE for autofeed, FALSE for manual feed.

The result is a boolean value that indicates success or failure, and is usually discarded.

Loose Pages

Pages normally exist only within folders (which exist only within drawers, and which, in turn, exist only within cabinets). It is sometimes desirable, however, to create a 'loose' page in memory, that is not associated with a cabinet, drawer, or folder. You can do this in three ways. The first is:

```
result:=CREATEPAGE(form)
```

FORM is the name of a form. The result is either NIL or a new, open page created from the specified form. The page exists only in memory.

You can also create a loose page by replicating an existing page with this function:

```
result:=COPYPAGE(page)
```

The result is a loose copy of the page (or NIL if the function fails). Do not copy closed pages.

The third method for creating a loose page is to remove an existing page from its folder like so:

```
result:=UNFILEPAGE(page)
```

In this case, the result is boolean, indicating success or failure. The PAGE variable now exists in memory only. This function opens the page.

Before your program ends, you must properly dispose of your loose pages. You can do this by either filing or destroying them. File like this:

```
result:=FILEPAGE(folder,page)
result:=FILEPAGE(folder,page,after)
```

The specified FOLDER must be open.

If the AFTER parameter is omitted, the page is placed at the very end of the folder. The result is boolean, indicating success or failure.

Destroy like this:

```
DESTROYPAGE(page)
```

This function does not return a result.

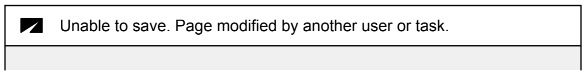
File System Functions

Locking Functions

Perspective automatically serializes disk activity to insure that no data is lost in a multi-user environment, using a technique called 'optimistic locking' at the visible page level. This technique is based on the premise that two or more users may want to view a particular page at the same time, but it is rare that more than one user will want to update the same page at the same time.

Perspective 'locks' each cabinet for the duration of each built-in transaction. This insures that consistent data will be read from, and written to, the disk. The duration of these locks is normally a few milliseconds, and should be virtually undetectable in systems with 100 or fewer users.

Perspective also maintains a modification level for each page. This level is incremented each time a modified page is written to disk. If the modification level of the page to be written does not agree with the level on the disk, the following message is displayed,



Unable to save. Page modified by another user or task.

and the write is aborted. You can then copy the information on the screen, re-open that page to see its current state, and paste the copied information, if it is still appropriate to do so.

Automatic Locking

All of the locking operations described above are automatic, and are usually of no concern to users or programmers of Perspective systems. Simply go about your business, and let Perspective take care of all the multi-user considerations.

Manual Locking

There are instances, however, when manual locking is required (usually when an update involves more than one page). Manual locking is accomplished by grouping the necessary operations into a transaction that begins with a LOCK and ends with an UNLOCK function.

Locking

You can gain exclusive use of a cabinet using the function shown here:

```
result:=LOCK(cabinetname)
```

If the cabinet is already locked by another user, the function retries for 90 seconds before failing.

The result is a boolean indicating whether or not the lock was granted.

Unlocking

You can release an exclusive lock on a cabinet using this function:

```
UNLOCK(cabinetname)
```

This function executes immediately and does not return a result. No action is taken if you attempt to unlock a cabinet that is not locked.

Unlocking All

You can release all your locks with this:

```
UNLOCKALL
```

This function also executes immediately and does not return a result.

Helpful Hints

• Keep your locks short by minimizing the number of operations performed between the LOCK and UNLOCK functions.

• Do not issue a WARNING, an ERROR, or conduct a DIALOG within a lock — the cabinet will be inaccessible to all until the user responds.

• Remember that locks accumulate. If you LOCK a cabinet three times, you must UNLOCK it three times. Use UNLOCKALL if you're such a sloppy programmer that you can't keep track.

• Perspective releases all locks when a program terminates, either normally or abnormally.

Interface Functions

Overview

Access to the user interface from Perspective's programming language is intentionally limited, both to simplify the language and to keep the interface consistent from system to system.

It is rare that a Perspective programmer needs to write an 'interactive' program, anyway, because almost all of the real-time operations needed by users are built into the system.

The functions in this section are provided, however, since even 'batch' routines sometimes require parameters as input, and may also need to display status, completion, and error messages to a user in an interactive fashion.

Beeping and Blinking

You can produce a short click on the computer's speaker using either of these functions:

```
BEEP
BLINK
```

The latter function also flashes the LOGO at the left of the menu bar. Note that neither function returns a value.

Messages, Warnings, and Errors

You can display status messages like this:

```
MESSAGE(values)
```

The VALUES parameter is any list of simple variables (booleans, dates, times, numbers, and strings) separated by commas.

You can display warnings and errors similarly:

```
WARNING(values)
ERROR(values)
```

VALUES are as described above. The menu bar is not redrawn after a warning is dismissed by the user (allowing you to continue a dialog without an annoying flash); the menu bar is redrawn when the user dismisses an error.

Dialogs

You can conduct a standard dialog with a user using this function:

```
result:=DIALOG(question,d,t,n,f,answer)
```

The QUESTION is the prompt that appears just to the right of the LOGO.

D is for DEFAULT and is the value that is initially displayed in the editable portion of the dialog.

The T, N, and F flags are included to control the behavior of the dialog, as described below.

T is short for TRIM. Set this flag to TRUE if you want leading and trailing blanks removed from the user's response; set it to FALSE otherwise.

N stands for NULLS. Set this flag to TRUE if you want blank entries returned. Setting this flag to FALSE causes a blank entry to cancel the dialog.

F is the FINAL flag. Set this flag to TRUE if this is the last question you plan to ask (and the menu bar will be redrawn automatically after the user responds). Set it to FALSE if you have other questions (to avoid flicker between them).

The ANSWER contains the user's response. It is a string. You can use the CTB, CTD, CTN, or CTT functions if you need to convert it to a boolean, date, number, or time.

The function returns FALSE if the dialog was canceled; it returns TRUE otherwise. ANSWER is valid only if the function returns TRUE.

Refreshing the Sceen

You can refresh the menu bar, the current work area, or both using these functions:

```
DRAWMENUBAR
DRAWWORKAREA
```

You should normally not need to call these functions since Perspective refreshes the screen automatically at the end of each command.

Interface Functions

The Current View

You can determine the current view like this:

```
result:=XCV
```

The result will be one of the following constants:

```
AREAVIEW
CABINETVIEW
DRAWERVIEW
FOLDERVIEW
PAGEVIEW
```

The Current Object

You can access the current cabinet, drawer, folder, or page using the functions below:

```
result:=XCC
result:=XCD
result:=XCF
result:=XCP
```

These functions return NIL if they are called at inappropriate times, as shown in this chart:

XCV	XCC	XCD	XCF	XCP
areaview	nil	nil	nil	nil
cabinetview	valid	nil	nil	nil
drawerview	valid	valid	nil	nil
folderview	valid	valid	valid	nil
pageview	valid	valid	valid	valid

Note that while you can use these functions (in a round-about sort of way) to see what the current view is, it is better to use XCV.

The Current Page

The XCV, XCC, XCD, XCF, and XCP functions should not be used in the Code slice of pages. To refer to the 'current' page of a Code slice, use:

```
THISPAGE
```

'Current' here means the page that contains the code being executed in the Code slice, which is not necessarily the current page on the screen.

The Current Selection

You can find the first or last selected object:

```
result:=GETFIRSTSELECTEDDRAWER(cabinet)
result:=GETFIRSTSELECTEDFOLDER(drawer)
result:=GETFIRSTSELECTEDPAGE(folder)

result:=GETLASTSELECTEDDRAWER(cabinet)
result:=GETLASTSELECTEDFOLDER(drawer)
result:=GETLASTSELECTEDPAGE(folder)
```

And you can get a count of selected objects:

```
result:=GETSELECTEDDRAWERCOUNT(cabinet)
result:=GETSELECTEDFOLDERCOUNT(drawer)
result:=GETSELECTEDPAGECOUNT(folder)
```

All of the above functions return NIL or zero if called in an inappropriate view.

Changing the Selection

You can modify the selection with these:

```
SELECTDRAWER(drawer,flag)
SELECTFOLDER(folder,flag)
SELECTPAGE(page,flag)

SELECTDRAWERS(cabinet,flag)
SELECTFOLDERS(drawer,flag)
SELECTPAGES(folder,flag)
```

A value of TRUE in the FLAG indicates that you want to select; FALSE says deselect. The latter three functions select or deselect all.

Changing the View

You can 'flip' the view from its current state to another using these very powerful functions:

```
VIEWCABINET(cabinet,drawer)
VIEWDRAWER(drawer,folder)
VIEWFOLDER(folder,page)
VIEWPAGE(page)
```

The current object is closed, the specified object is opened, and the indicated item (if not NIL) is selected and positioned on the screen.

Interface Functions

Menu Functions

You can add items to the File, Special, and Page menus using the functions described below in a MENUEVENT function. Items are added to the 'Print' section of the File menu, to the bottom of the Special menu, and to the 'New Page' section of the Page menu.

Adding Items

The function used to add items to the File and Special menus is:

```
ADDITEM(menu,item,key,&function)
```

The same function is used to add items to the Page menu, but with slightly different parameters:

```
ADDITEM(menu,item,key,form)
```

The MENU parameter is one of the following predefined constants:

```
FILEMENU SPECIALMENU PAGEMENU
```

The ITEM parameter is the name that will appear on the menu, and must be enclosed in quotes.

The KEY parameter is the shortcut key. It should be a character between 0 and 9, enclosed in quotes. Use a null string if there is no shortcut.

The FUNCTION parameter is the name of the function associated with this item. The '&' symbol preceding the function name is required.

The FORM parameter is the name of the page that will be used to create a new page when the user selects the item from the Page menu.

Separating Items

You can add separator lines with this function:

```
ADDSEPARATOR(menu)
```

The menu parameter must be one of the three menu constants defined above. Extra separators are automatically eliminated.

Example

The following routine adds items to the File, Special, and Page menus.

```
function menuevent do
  additem(specialmenu,'Wink','3',&mywink)
  if xcv<>folderview do exit end
  if xcf.name<>'mydata' do exit end
  additem(filemenu,'Print One','',&myprt)
  addseparator(pagemenu)
  additem(pagemenu,'New Form1','',myfm1)
  additem(pagemenu,'New Form2','',myfm2)
  additem(pagemenu,'New Form3','',myfm3)
  addseparator(pagemenu)
  additem(pagemenu,'New FormA','',myfmA)
  additem(pagemenu,'New FormB','',myfmB)
  additem(pagemenu,'New FormC','',myfmC)
  additem(pagemenu,'New FormD','',myfmD)
end
```

Note that the 'Wink' item is always added to the Special menu, while the remaining items appear only in Folder View when the current folder's name is 'mydata' (case-insensitive compare).

Note that the 'Wink' command can be executed from the keyboard by typing CTRL-3 or ALT-3.

Note also that functions named 'mywink' and 'myprt' must exist in the folder containing this MENUEVENT function, and that forms named 'myfm1', 'myfm2', 'myfm3', 'myfmA', and so forth, must also be stored in this folder.

Now consider the separators on the Page menu. The first will immediately follow the standard 'New Drawing' item at the top of the menu. The second will appear between 'Form3' and 'FormA'.

Separators should be used sparingly, if at all. Too many separators make menus harder, not easier, to read. Try to keep the number of items in each group small, and vary the number of items in each group to make them more visually distinct.

More about Events

If you missed the 'Events' topic in the previous chapter, you should go back and read it now.

Date and Time Functions

Overview

Perspective stores dates and times internally as sequential numbers. The function library includes a wide variety of functions for working with dates and times in more familiar forms.

Getting the Date and Time

You can get the current date and time using the functions shown below:

```
result:=GETDATE
result:=GETTIME
```

The date and time values are retrieved from the system clock on the active computer.

Separating Dates and Times

You can extract the component parts of dates and times using the following functions:

```
result:=GETYEAR(date)
result:=GETMONTH(date)
result:=GETDAY(date)

result:=GETHOURS(time)
result:=GETMINUTES(time)
result:=GETSECONDS(time)
```

Years are numbers between 1900 and 9999; months are numbers between 1 and 12; days are numbers from 1 to 31, depending on the month.

Hours are numbers between 0 (midnight) and 23. Minutes and seconds range from 0 to 59.

Getting the Day of Week

A special function is included that returns the day of the week for any valid date. The syntax is:

```
result:=GETDAYOFWEEK(date)
```

The result is a number between 1 and 7, where 1=Sunday, 2=Monday, 3=Tuesday, etc.

This function is handy for labeling documents and for special processing at end-of-week, etc.

Converting Strings to Dates

You can convert string representations of dates into Perspective's internal date format using the convert-to-date function:

```
result:=CTD(string)
```

The string can be in any of the following forms:

```
'1/1/98'
'1/1/1998'
'Jan 1, 1998'
'January 1, 1998'
```

Invalid dates are stored as zero and displayed as the string 'Invalid Date'.

Converting Strings to Times

You can convert string representations of times into Perspective's internal time format using the convert-to-time function:

```
result:=CTT(string)
```

The string can be in any of the following forms:

```
'23:00'
'23:00 am'
'23:59:59'
'12:00:00 am'
```

Invalid times are stored as 999999 and displayed as the string 'Invalid Time'.

Converting Dates and Times to String

You can convert a date or time to a string, with or without formatting, using these functions:

```
result:=CTS(date)
result:=CTS(time)
result:=FORMATDATE(date,format)
result:=FORMATTIME(time,format)
```

The convert-to-string function performs no special formatting. Valid formats for the latter functions are discussed in the Form Making chapter under the topic 'The Format Slice'.

String Functions

Overview

A number of general-purpose string manipulation functions are included in Perspective's library.

Working with Characters

You can convert a single character to its numeric equivalent, and vice-versa, with these functions:

```
result:=ORD(character)
result:=CHR(number)
```

Numeric values are standard ASCII.

Getting the Length

You can get a count of the number of characters in a string using this function:

```
result:=LEN(string)
```

The minimum length is zero. The maximum is approximately two billion characters.

Inserting and Deleting Characters

You can insert and delete characters in a string with these functions:

```
result:=INSERT(string,insertstring,pos)
result:=DELETE(string,pos,count)
```

Note that the modified string is returned; the original is not changed.

Extracting Characters

The functions below can be used to extract characters from the left, middle, or right of a string, respectively.

```
result:=LEFT(string,count)
result:=MID(string,pos,count)
result:=RIGHT(string,count)
```

If the count exceeds the available number of characters, the system explodes. Just kidding. All available characters are returned. The original string is not modified.

Adjusting the Case

You can convert a string to all upper or all lower case characters using the following functions:

```
result:=UCASE(string)
result:=LCASE(string)
```

The names are abbreviated for easy nesting.

Trimming Strings

You can 'pad' strings on the left or right with a specific character using the two functions below:

```
result:=FILLLEFT(string,character,count)
result:=FILLRIGHT(string,character,count)
```

And you can remove padding with these:

```
result:=REMOVELEADING(string,chars)
result:=REMOVETRAILING(string,chars)
```

The CHARS are one or more single-character parameters, separated by commas.

Searching Strings

You can locate the position of a substring within a string using the function below:

```
result:=POS(substring,string)
```

If the substring is found, the result is the position of the substring in the string (the first character is one). If not found, the result is zero.

Converting Strings to Other Types

Strings can be converted to other basic types using conversion functions described elsewhere in this chapter. String formats are discussed in detail in the Programming chapter.

Converting Other Types to Strings

All basic data types can be converted to strings, with or without formatting, using the appropriate convert-to-string and formatting functions described under other headings in this chapter.

Number Functions

Overview

Perspective's function library includes a complete set of general-purpose mathematical routines.

Rounding Numbers

You can round numbers up, down, or in the usual way (.5 or less goes down) with these functions:

```
result:=ROUNDUP(number)
result:=ROUNDDOWN(number)
result:=ROUND(number)
```

You can get just the integer or just the decimal portion of a number like this:

```
result:=TRUNC(number)
result:=FRAC(number)
```

And you can get the absolute value of a number (its distance from zero) using this function:

```
result:=ABS(number)
```

The absolute value is always positive.

Adjusting Numbers

You can conveniently increment or decrement a number by one or any other amount like this:

```
INC(number)
INC(number,amount)
DEC(number)
DEC(number,amount)
```

These functions are faster and more concise than their equivalents. Compare inc(x) with x:=x+1.

Comparing Numbers

You can find the larger or smaller of two numbers quickly and easily using these functions:

```
result:=MAX(number,number)
result:=MIN(number,number)
```

See 'Statistical Functions' in this chapter for an alternate form of these functions.

Roots and Logarithms

You can find the square root of any number and its logarithm using these functions:

```
result:=SQRT(number)
result:=LOG(number,base)
```

Natural logarithms can be derived using 2.7182 as the value for the BASE parameter.

Trigonometric Functions

The six basic trigonometric functions and two related conversion functions are also included in the library, as shown here:

```
result:=SIN(number)
result:=COS(number)
result:=TAN(number)
result:=ARCSIN(number)
result:=ARCCOS(number)
result:=ARCTAN(number)

result:=DEGREETORADIAN(number)
result:=RADIANTODEGREE(number)
```

Converting Strings to Numbers

You can convert strings to numbers with the convert-to-number function, like so:

```
result:=CTN(string)
```

The string can contain a dollar sign, a plus or minus sign, digits, and a decimal point. All invalid characters are dropped before conversion.

Converting Numbers to Strings

You can convert numbers to strings, with or without formatting, using these functions:

```
result:=CTS(number)
result:=FORMATNUMBER(number,format)
```

The convert-to-string function translates the number without formatting. Number formats are discussed in the Form Making chapter under the heading 'The Format Slice'.

Statistical Functions

Overview

Statistical functions are used to calculate totals, averages, and other figures from arrays of shapes on pages.

These functions are typically found in the Code slice of pages, though they can also be called directly in programs that you write.

Counting Shapes

You can determine the number of non-NIL values in an array of shapes on a page like so:

```
result:=COUNT(shapename)
```

The result is the number of shapes with the specified name whose Value slice is not NIL.

Summing Shapes

You can determine the total value in an array of shapes on a page using this function:

```
result:=SUM(shapename)
```

Shapes with blank values are skipped. If all shapes are blank, the result is NIL.

Comparing Shapes

You can determine the largest or smallest value in an array of shapes on a page like this:

```
result:=MAX(shapename)
result:=MIN(shapename)
```

Shapes with blank values are skipped. If all shapes are blank, the result is NIL.

Averaging Shapes

You can determine the average value in an array of shapes on a page using this function:

```
result:=AVG(shapename)
```

Shapes with blank values are skipped. If all shapes are blank, the result is NIL.

Example

First, we draw a page with twelve text shapes, as shown below. The top two shapes are headings.

INPUT	OUTPUT

Then, we use the Name slice and to make an array (assigning the same name to each shape):

input	
input	
input	
input	
input	

Next, we switch to the Code slice and enter five functions in the right column, as shown below:

	count(input)
	sum(input)
	max(input)
	min(input)
	avg(input)

Finally, we return to the Value slice and enter the data on the left. When we save the page, the calculated values appear in the OUTPUT column:

INPUT	OUTPUT
100	4
200	1000
	400
300	100
400	250

Note that the missing data on the third row was excluded from all calculations: the minimum is 100 (not 0) and the average is 250 (not 200).

Text File Functions

Overview

Perspective includes a small but sufficient set of functions for working with text files. These functions are normally used for importing data from, or exporting data to, another system.

Return Values

All of the functions in this column return TRUE if they are successful, FALSE if they are not.

Creating and Deleting Text Files

You can create and delete text files like this:

```
result:=NEWFILE(name)
result:=DELETEFILE(name)
```

The NAME parameter must be enclosed in single quotes and include the drive, directory, filename, and extension of the file.

Reading and Writing Text Files

You can read and write text files in this way:

```
result:=READFILE(name,string)
result:=WRITEFILE(name,string)
```

The NAME parameters are as described above. When writing, a new file is created, if necessary. Any existing file with the specified name is completely overwritten.

The STRING parameters are local or global variables that hold the contents of the file. When reading, previous contents of the string, if any, are lost. When writing, the values in the string parameters are not changed.

Renaming Text Files

You can rename text files using this function:

```
result:=RENAMEFILE(oldname,newname)
```

Both names must be fully qualified. If the drive and/or directory part of the NEWNAME do not match the OLDNAME, the file is moved.

Cleaning Text Files

You can eliminate unwanted characters from a text file using the following function:

```
result:=REMOVEUNWANTED(string)
```

All control characters except for carriage return, linefeed, and tab are removed from the input string and the resulting string is returned.

Processing Text Files

You can extract data from a text file string with the get-next and get-previous-token functions:

```
result:=GNT(string,pos,flag,delims)
result:=GPT(string,pos,flag,delims)
```

The STRING parameter is a string containing the text to be parsed, typically read from a file.

The POS parameter is the current position in the file. The first character is one.

The FLAG parameter indicates whether or not delimiters should be returned as tokens. Specify TRUE for yes, FALSE for no.

The DELIMS are one or more single-character parameters separated by commas. Common delimiters are spaces, tabs, carriage returns, and linefeeds. The predefined constants XSP, XTB, XCR, and XLF can be used here.

The function returns all characters in the string from the current POS up to, but not including, the next DELIM. The POS is adjusted appropriately.

You can also use this function:

```
result:=GETTOKEN(string,num,flag,delims)
```

which behaves like the preceding two except that the NUM parameter indicates a specific token to retrieve by position within the string, counting from the beginning. The first token is one.

See 'Importing and Exporting Data' on page 97 for an example using these text file functions.

Miscellaneous Functions

Overview

This topic covers a handful of miscellaneous but useful functions built into Perspective.

Boolean Functions

The convert-to-boolean function accepts a number or a string as input and attempts to convert it to a boolean value:

```
result:=CTB(input)
```

Positive numbers and upper or lower case strings beginning with '1', 'y', 't', 'ok', 'on', 'yes', and 'true' are converted to TRUE. Zero, negative numbers, and all other values are converted to FALSE.

Boolean values can be converted to strings, with or without formatting, using these functions:

```
result:=CTS(boolean)
result:=FORMATBOOLEAN(boolean,format)
```

The convert-to-string function translates the boolean into 'true' or 'false'. Boolean formats are discussed in the Form Making chapter under the heading 'The Format Slice'.

Timer Functions

You can get the system tick count like this:

```
result:=GETTICKS
```

The result is the number of seconds (accurate to four decimal places) that have passed since the computer was restarted.

You can pause execution of a program for a specified period of time in this way:

```
PAUSE(seconds)
```

Up to four decimal places can be included. Note that this function does not have a return value.

You can display the time required to execute any function you have written by holding the SHIFT key when you execute the function from a menu.

Random Numbers

You can get a random integer using this function:

```
result:=RANDOM(min,max)
```

The MIN and MAX parameters specify the inclusive lower and upper bounds, respectively. Only the integer portion is considered, and only integers are returned from this function.

You can also reset the random number generator:

```
RANDOMIZE
RANDOMIZE(seed)
```

In the first case, the random number generator is initialized with a system-defined seed, resulting in a different set of random numbers in each run. In the second case, the specified SEED number is used, which will generate the same 'random' number sequence each time.

Upper Bounds

You can determine the number of shapes with a given name on a page using this function:

```
result:=UPPERBOUND(shapename)
```

The result is the maximum subscript and can be used as a terminating value in loops.

Count Messages

Status and completion messages often contain a number followed by a noun that may or may not need to be plural, like this:

```
0 pages
1 page
2 pages
```

These messages are a nuisance to formulate without the make-count-message function:

```
result:=MCM(number,ss,ps)
```

SS is a string appended to the number in the singular case; PS is appended otherwise.

7

APPENDIXES

Technical Specifications

A summary of Perspective's hardware and software requirements, capacities, and limitations is provided in the chart below. Good performance can be realized on any compatible computer if the recommended values are followed (rather than the theoretical maximums). Less stringent limits apply to larger and faster machines.

GROUP	DESCRIPTION	MINIMUM	MAXIMUM	RECOMMENDED	UNITS
Hardware and Software	CPU	DX4	unlimited	Pentium or better	chip
	Memory	8	unlimited	16 or more	megabytes
	Disk	2	unlimited	100 or more	megabytes
	CD-Rom	1	unlimited	1	drive (1)
	Operating System	n/a	n/a	Windows 95	n/a
General	Lexicon	0	unlimited	approx 65,000	words
	Menus	0	25	3-12	items per menu
Cabinets	Number of	1	26	up to 26	cabinets
	Number of (with config file)	1	unlimited	unlimited	cabinets
	Name length	0	20	3-12	characters
Drawers	Number of	0	unlimited	up to 1000	per cabinet
	Group length	0	20	3-12	characters
	Name length	0	20	3-12	characters
	Version length	0	20	3-12	characters
Folders	Number of	0	unlimited	up to 1000	per drawer
	Name length	0	60	3-30	characters
	Version length	0	20	3-12	characters
Pages	Number of	0	unlimited	up to 250	per folder
	Width	2	11	8.5	inches
	Height	2	11	11	inches
	Type length	0	20	3-12	characters
	Identifier length	0	40	3-30	characters
	Edge shapes (top edge)	0	7	up to 7	per page
	Edge shapes (bottom edge)	3	3	3	per page
	Edge values	0	40	up to 40	characters
Shapes	Number of	0	unlimited	up to 500	per page
	Name length	0	unlimited	1-20	characters
	Curve vertexes	3	30	up to 30	per curve
	Polygon vertexes	2	30	up to 30	per polygon
	Text lines	0	250	n/a	per text shape
Data Values	Booleans	0	1	either	n/a
		n	y	either	n/a
		f	t	either	n/a
		not	ok	either	n/a
		off	on	either	n/a
		no	yes	either	n/a
		false	true	either	n/a
	Dates	01/01/1900	12/31/9999	any	days
	Times	00:00:00	23:59:59	any	seconds
	Numbers	-9.223E14	+9.223E14	any	units
	Strings	0	2 billion	any	characters

(1) CD-Rom drive required for installation only; cannot be used for storage of drawers, folders, and/or pages.

The Configuration File

The Configuration File

Perspective's configuration file allows you to give names to all of the cabinets and printers on a Perspective network. It can be used in a single-user system, though it normally is not. This file resides in the PERSPECTIVE folder on the machine where the executable file is located. Do not change the name of this file.

Editing

The configuration file is a standard text file. You can edit this file using NOTEPAD or any other text-only editor. You can also edit it using a word processor, but you must save it as text only.

Statements

The configuration file contains two kinds of statements: CABINET statements and PRINTER statements. Statements can be entered in any order, but each one must be on a single line.

Comments

Comments in the configuration file must be enclosed in { curly braces }. If the ending brace is omitted, it is assumed at the end of the current line. Multi-line comments are not supported.

Defining Cabinets

The CABINET statement is used to assign names to disks and directories that are to be considered cabinets by Perspective. The syntax is:

```
CABINET name path
```

CABINET is a keyword that must be entered exactly as shown (upper or lower case).

NAME is the title by which the cabinet will be known within the Perspective environment.

PATH is the fully-qualified location of the disk or folder (backslash on end is optional). Examples:

```
A:\
\\mycomputer\mycabinets\sharon
```

Defining Printers

The PRINTER statement is used to name shared printers on a network, and to define several of their characteristics that cannot be determined automatically with any degree of consistency. The syntax is:

```
PRINTER name device p1 p2 p3 p4
```

PRINTER is a keyword that must be entered exactly as shown (upper or lower case).

NAME is the title by which the printer will be known within the Perspective environment.

DEVICE is the name of the printer as it is known to Windows. Enclose this value in single quotes if the name contains spaces.

P1 indicates whether the printer is a postscript printer or not. Valid values for this parameter are POSTSCRIPT and NOPOSTSCRIPT.

P2 indicates whether the printer's manual feed slot centers or left-justifies the page. Valid values are CENTERFEED and NOCENTERFEED.

P3 indicates whether pages should be printed in forward or reverse order. Valid values are REVERSE and NOREVERSE.

P4 indicates the amount of rotation, in degrees, required for landscape pages. Valid values are 0, 90, and 270. You may have to experiment to determine the proper setting for this parameter. Most laser printers use 90; most others use 270.

Examples

A typical configuration file might begin...

```
{ my configuration file }
cabinet a a:\
cabinet server \\ourserver\c
printer local 'BJC 4200' nopostscript...
printer remote 'HP 5L' nopostscript...
```

Additional samples can be found in the configuration file supplied with Perspective.

Importing and Exporting Data

Overview

You can import and export data from and to other non-Perspective systems using text files as the transport medium. Here's how.

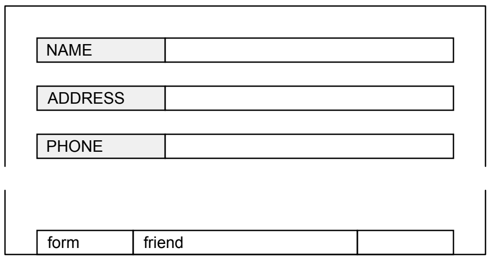
Defining the Form

Say, for example, we wanted to import a text file from another system, with the following format:

```
name <tb> address <tb> phone <cr> <lf>
```

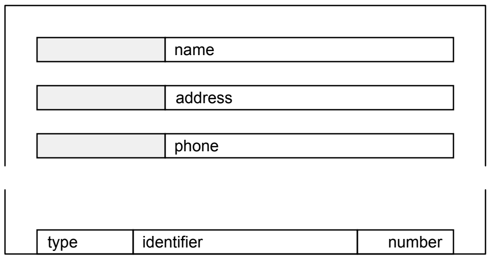
where <tb> represents a tab character, <cr> is a carriage return, and <lf> indicates a line feed.

We would first design a form, like this one (Value slice showing):



Note the name of the form entered into the identifier shape on the bottom edge of the form.

Then we would name the shapes in the body of the form, like so (Name slice showing):



Note the names assigned to the three data shapes on the form.

The names of the edge shapes were assigned when the form was created.

Importing the Data

To import the data, we would use code like this:

```
function import do
  local c,d,f,p,t,tpos
  c:=opencabinet('c')
  d:=newdrawer(c,'My','Friends','1998')
  f:=newfolder(d,'Imported','')
  readfile('c:\data',t)
  tpos:=1
  while tpos<len(t) do
    p:=createpage(friend)
    p.name:=gnt(t,tpos,false,xtb,xcr,xlf)
    p.address:=gnt(t,tpos,false,xtb,xcr,xlf)
    p.phone:=gnt(t,tpos,false,xtb,xcr,xlf)
    filepage(f,p)
    closepage(p)
  end
  closefolder(f)
  closedrawer(d)
  closecabinet(c)
end
```

Note that this code does not allow missing fields.

Exporting the Data

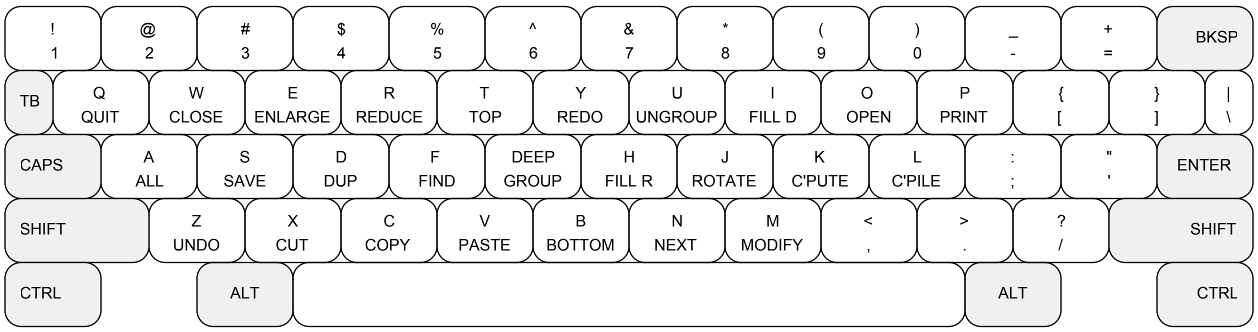
To export our friends, we use code like this:

```
function export do
  local c,d,f,p,t
  c:=opencabinet('c')
  d:=finddrawer(c,'My','Friends','1998')
  opendrawer(d)
  f:=findfolder(d,'Imported','')
  openfolder(f)
  t:=''
  for p in f do
    openpage(p)
    t:=t+p.name+xtb+p.addr+xtb+p.phone+xcrlf
    closepage(p)
  end
  writefile('c:\data',t)
  closefolder(f)
  closedrawer(d)
  closecabinet(c)
end
```

And that's all there is to it.

Keyboard Summary

A summary of Perspective's shortcut keys, in pictorial form, appears below. Note that the 'G' key is a shortcut for both 'Find Deep' and 'Group' (depending on the current view), and that '0' through '9' are available for user-defined functions. A summary of the various functions assigned to other keys is also provided.



ESC	Closes the selected drawer, folder, or page; ends edit of a text shape.	HOME	Moves selection to the top or left; shows first page in Page View
TAB	Moves selection right and/or down; with SHIFT, moves left and/or up	END	Moves selection to the bottom or right; shows last page in Page View
ENTER	Opens selected item; moves to next editable text shape on forms	PAGE UP	Moves display one screen's worth up; shows previous page in Page View
SHIFT	Selects more or less; speeds scrolling; reverses effect of TAB and ENTER	PAGE DOWN	Moves display one screen's worth down; shows next page in Page View
CTRL	Used with character keys to execute commands from the keyboard	↑	Moves up; selects with SHIFT; changes slice with CTRL or ALT
ALT	Used with character keys to execute commands from the keyboard	↓	Moves down; selects with SHIFT; changes slice with CTRL or ALT
BKSP	Deletes selected objects; deletes text to the left of the insertion point	←	Moves selection or insertion point left; with SHIFT, selects characters
DEL	Deletes selected objects; deletes text to the right of the insertion point	→	Moves selection or insertion point right; with SHIFT, selects characters

Keyboard Summary

The chart below shows the special characters for each of the three fonts supported by Perspective. You can enter these characters in any text shape by holding the ALT key and typing the indicated four-digit number. The CHR and ORD functions can also be used to convert these numbers to characters, and vice-versa.

ARIAL					
0130	‚	0172	¬	0214	Ö
0131	ƒ	0173	-	0215	×
0132	„	0174	®	0216	Ø
0133	…	0175	¯	0217	Ù
0134	†	0176	°	0218	Ú
0135	‡	0177	±	0219	Û
0136	ˆ	0178	²	0220	Ü
0137	‰	0179	³	0221	Ý
0138	Š	0180	´	0222	Þ
0139	‹	0181	µ	0223	ß
0140	Œ	0182	¶	0224	à
0141	□	0183	·	0225	á
0142	Ž	0184	¸	0226	â
0143	□	0185	¹	0227	ã
0144	□	0186	º	0228	ä
0145	‘	0187	»	0229	å
0146	’	0188	¼	0230	æ
0147	“	0189	½	0231	ç
0148	”	0190	¾	0232	è
0149	•	0191	¿	0233	é
0150	–	0192	À	0234	ê
0151	—	0193	Á	0235	ë
0152	˜	0194	Â	0236	ì
0153	™	0195	Ã	0237	í
0154	š	0196	Ä	0238	î
0155	›	0197	Å	0239	ï
0156	œ	0198	Æ	0240	ð
0157	□	0199	Ç	0241	ñ
0158	ž	0200	È	0242	ò
0159	Ÿ	0201	É	0243	ó
0160		0202	Ê	0244	ô
0161	¡	0203	Ë	0245	õ
0162	¢	0204	Ì	0246	ö
0163	£	0205	Í	0247	÷
0164	¤	0206	Î	0248	ø
0165	¥	0207	Ï	0249	ù
0166	¦	0208	Ð	0250	ú
0167	§	0209	Ñ	0251	û
0168	¨	0210	Ò	0252	ü
0169	©	0211	Ó	0253	ý
0170	ª	0212	Ô	0254	þ
0171	«	0213	Õ	0255	ÿ

COURIER					
0130	‚	0172	¬	0214	Ö
0131	ƒ	0173	-	0215	×
0132	„	0174	®	0216	Ø
0133	…	0175	¯	0217	Ù
0134	†	0176	°	0218	Ú
0135	‡	0177	±	0219	Û
0136	ˆ	0178	²	0220	Ü
0137	‰	0179	³	0221	Ý
0138	Š	0180	´	0222	Þ
0139	‹	0181	µ	0223	ß
0140	Œ	0182	¶	0224	à
0141	□	0183	·	0225	á
0142	Ž	0184	¸	0226	â
0143	□	0185	¹	0227	ã
0144	□	0186	º	0228	ä
0145	‘	0187	»	0229	å
0146	’	0188	¼	0230	æ
0147	“	0189	½	0231	ç
0148	”	0190	¾	0232	è
0149	•	0191	¿	0233	é
0150	–	0192	À	0234	ê
0151	—	0193	Á	0235	ë
0152	˜	0194	Â	0236	ì
0153	™	0195	Ã	0237	í
0154	š	0196	Ä	0238	î
0155	›	0197	Å	0239	ï
0156	œ	0198	Æ	0240	ð
0157	□	0199	Ç	0241	ñ
0158	ž	0200	È	0242	ò
0159	Ÿ	0201	É	0243	ó
0160		0202	Ê	0244	ô
0161	¡	0203	Ë	0245	õ
0162	¢	0204	Ì	0246	ö
0163	£	0205	Í	0247	÷
0164	¤	0206	Î	0248	ø
0165	¥	0207	Ï	0249	ù
0166	¦	0208	Ð	0250	ú
0167	§	0209	Ñ	0251	û
0168	¨	0210	Ò	0252	ü
0169	©	0211	Ó	0253	ý
0170	ª	0212	Ô	0254	þ
0171	«	0213	Õ	0255	ÿ

TIMES					
0130	‚	0172	¬	0214	Ö
0131	ƒ	0173	-	0215	×
0132	„	0174	®	0216	Ø
0133	…	0175	¯	0217	Ù
0134	†	0176	°	0218	Ú
0135	‡	0177	±	0219	Û
0136	ˆ	0178	²	0220	Ü
0137	‰	0179	³	0221	Ý
0138	Š	0180	´	0222	Þ
0139	‹	0181	µ	0223	ß
0140	Œ	0182	¶	0224	à
0141	□	0183	·	0225	á
0142	Ž	0184	¸	0226	â
0143	□	0185	¹	0227	ã
0144	□	0186	º	0228	ä
0145	‘	0187	»	0229	å
0146	’	0188	¼	0230	æ
0147	“	0189	½	0231	ç
0148	”	0190	¾	0232	è
0149	•	0191	¿	0233	é
0150	–	0192	À	0234	ê
0151	—	0193	Á	0235	ë
0152	˜	0194	Â	0236	ì
0153	™	0195	Ã	0237	í
0154	š	0196	Ä	0238	î
0155	›	0197	Å	0239	ï
0156	œ	0198	Æ	0240	ð
0157	□	0199	Ç	0241	ñ
0158	ž	0200	È	0242	ò
0159	Ÿ	0201	É	0243	ó
0160		0202	Ê	0244	ô
0161	¡	0203	Ë	0245	õ
0162	¢	0204	Ì	0246	ö
0163	£	0205	Í	0247	÷
0164	¤	0206	Î	0248	ø
0165	¥	0207	Ï	0249	ù
0166	¦	0208	Ð	0250	ú
0167	§	0209	Ñ	0251	û
0168	¨	0210	Ò	0252	ü
0169	©	0211	Ó	0253	ý
0170	ª	0212	Ô	0254	þ
0171	«	0213	Õ	0255	ÿ

Menu Summary

A summary of Perspective's menus is provided below. Note that not all commands are available in all views, and that certain commands apply only to certain page types (drawings, forms, or functions). Note also that user-defined commands can be added to the File, Special, and Page menus at the indicated places.

File

Open
Save
Close

Print...
Print Special..
Print (user)

Sleep
Quit

Edit

Undo
Redo

Cut
Copy
Paste
Duplicate

Select All

Fill Right
Fill Down

Suggestions

View

by Field

Reverse

Reduce
Enlarge

Top
Bottom

Show Edges
Show Grid Lir

Small Grid
Medium Grid
◇ Large Grid

◇ Value Slice
Format Slice
Name Slice
Code Slice
Update Slice

Special

Find...
Find Deep...
Find Misspelli
Find Next
Find Error

Compute... K
Compute Size

Add Word...
Delete Word...

Compile L

Other (user)

Drawer

New...

Rename...
Delete

Folder

New...

Backup...

Rename...
Delete

Page

New Drawing
New Form
New Function
New (user)

Renumber...
Delete

First
Previous
Next
Last

Resize...
Set Scale...

Shape

New Arc
New Curve
New Ellipse
New Polygon
New Rectang
New Text

Rotate
Flip Horizonta
Flip Vertically

Bring to Front
Send to Back

Group
Ungroup

◇ No Arrows
Arrow at Start
Arrow at End
Arrow at Both

Modify... M

Font

◇ Arial
Courier
Times

◇ Plain
Bold
Italic

◇ Left
Center
Right
Justify
Spread

8.0 Lines/Inch
6.0 Lines/Inch
◇ 4.0 Lines/Inch
3.0 Lines/Inch
2.0 Lines/Inch
1.5 Lines/Inch
1.0 Line/Inch

Pen

Clear
White
Gray
◇ Black

Red
Orange
Yellow
Green
Blue
Purple

Light
◇ Normal
Dark

Border

Clear
White
Gray
◇ Black

Red
Orange
Yellow
Green
Blue
Purple

Light
◇ Normal
Dark

Fill

Clear
White ◇
Gray ◈
◇ Black ◆

Red 🟥
Orange 🟧
Yellow 🟨
Green 🟩
Blue 🟦
Purple 🟪

Light
◇ Normal
Dark

Language Summary

The syntax of all of Perspective's programming language statements appears below. Items in uppercase are keywords (which can be entered in upper or lower case). Items in lower case are user-supplied values (with plurals indicating that more than one entry is valid). Square brackets indicate optional clauses.

STMT	SYNTAX
Assign	`variable := expression`
Break	`BREAK`
Call	`functionname [ ( parmlist ) ]`
Case	`CASE expression DO whenclauses [ ELSE statements ] END`
Continue	`CONTINUE`
Exit	`EXIT`
Function	`FUNCTION functionname [ ( parmlist ) ] [ RETURNS variable ] DO statements END`
For	`FOR variable := expression TO expression [ BACKWARDS ] DO statements END`
	`FOR variable IN expression [ BACKWARDS ] DO statements END`
Global	`GLOBAL variablelist`
If	`IF expression DO statements [ ELSE statements ] END`
Local	`LOCAL variablelist`
Loop	`LOOP statements END`
Repeat	`REPEAT statements UNTIL expression`
When	`WHEN expression DO statements END`
While	`WHILE expression DO statements END`

KEYWORDS								
AND	CASE	DO	EXIT	GLOBAL	LOCAL	NOT	RETURNS	WHEN
BACKWARDS	CONTINUE	ELSE	FOR	IF	LOOP	OR	TO	WHILE
BREAK	DIV	END	FUNCTION	IN	MOD	REPEAT	UNTIL	XOR

GROUP	VARIABLE	DESCRIPTION
Character	X00	Null
	XCR	Carriage return
	XCRLF	XCR + linefeed
	XLF	Linefeed
	XSP	Space
	XSQ	Single quote
	XTB	Tab
Menu	FILEMENU	File menu
	PAGEMENU	Page menu
	SPECIALMENU	Special menu
View	AREAVIEW	Area View
	CABINETVIEW	Cabinet View
	DRAWERVIEW	Drawer View
	FOLDERVIEW	Folder View
	PAGEVIEW	Page View
Other	NIL	Value assigned to all undefined variables
	THISPAGE	Page containing the code slice of the current shape

VARIABLE	ATTRIBUTES	DESCRIPTION
Cabinet	NAME	Cabinet name
	FIRST, LAST	First, last drawer
Drawer	GROUP	Drawer group
	NAME	Drawer name
	VERSION	Drawer version
	SEL	Selection flag
	CABINET	Containing cabinet
	FIRST, LAST	First, last folder
	NEXT, PREV	Next, prev drawer
Folder	NAME	Folder name
	VERSION	Folder version
	SEL	Selection flag
	DRAWER	Containing drawer
	FIRST, LAST	First, last page
	NEXT, PREV	Next, prev folder
Page	shapename	Edge or body shape
	SEL	Selection flag
	FOLDER	Containing folder
	NEXT, PREV	Next, prev page
Shape	SEL	Selection flag

Function Summary

A

```
ABS(number) -> number
ADDITEM(menu,item,key,&function)
ADDITEM(menu,item,key,form)
ADDSEPARATOR(menu)
ARCCOS(number) -> number
ARCSIN(number) -> number
ARCTAN(number) -> number
AVG(shapename) -> number
```

B

```
BEEP
BLINK
```

C

```
CHR(number) -> char
CLOSECABINET(cabinet) -> boolean
CLOSEDRAWER(drawer) -> boolean
CLOSEFOLDER(folder) -> boolean
CLOSEPAGE(page) -> boolean
COPYPAGE(page) -> page
COS(number) -> number
COUNT(shapename) -> number
CREATEPAGE(form) -> page
CTB(simpletype) -> boolean
CTD(string) -> date
CTN(string) -> number
CTS(simpletype) -> string
CTT(string) -> time
```

D

```
DEC(number[,amount])
DEGREETORADIAN(number) -> number
DELETE(string,pos,count) -> string
DELETEDRAWER(drawer) -> boolean
DELETEFILE(name) -> boolean
DELETEFOLDER(folder) -> boolean
DELETEPAGE(page) -> boolean
DESTROYPAGE(page)
DIALOG(question,d,t,n,f,answer) -> boolean
DRAWMENUBAR
DRAWWORKAREA
DUPLICATEDRAWER(c,d[,after]) -> drawer
DUPLICATEFOLDER(d,f[,after]) -> folder
DUPLICATEPAGE(f,p[,after]) -> page
```

E

```
ERROR(values)
```

F

```
FILEPAGE(folder,page[,after]) -> boolean
FILLLEFT(string,char,count) -> string
FILLRIGHT(string,char,count) -> string
FINDDRAWER(cabinet,g,n[,v]) -> drawer
FINDFOLDER(drawer,n[,v]) -> folder
FINDPAGE(folder,shapename,value) -> page
FORMATBOOLEAN(boolean,format) -> string
FORMATDATE(date,format) -> string
FORMATNUMBER(number,format) -> string
FORMATTIME(time,format) -> string
FRAC(number) -> number
```

G

```
GETDATE -> date
GETDAY(date) -> number
GETDAYOFWEEK(date) -> number
GETDRAWERCOUNT(cabinet) -> number
GETFIRSTSELECTEDDRAWER(cabinet) -> drawer
GETFIRSTSELECTEDFOLDER(drawer) -> folder
GETFIRSTSELECTEDPAGE(folder) -> page
GETFOLDERCOUNT(drawer) -> number
GETHOURS(time) -> number
GETLASTSELECTEDDRAWER(cabinet) -> drawer
GETLASTSELECTEDFOLDER(drawer) -> folder
GETLASTSELECTEDPAGE(folder) -> page
GETMINUTES(time) -> number
GETMONTH(date) -> number
GETPAGECOUNT(folder) -> number
GETSECONDS(time) -> number
GETSELECTEDDRAWERCOUNT(cabinet) -> number
GETSELECTEDFOLDERCOUNT(drawer) -> number
GETSELECTEDPAGECOUNT(folder) -> number
GETTICKS -> seconds
GETTIME -> time
GETTOKEN(string,num,flag,delims) -> string
GETYEAR(date) -> number
GNT(string,pos,flag,delims) -> string
GPT(string,pos,flag,delims) -> string
```

I

```
INC(number[,amount])
INSERT(string,insertstring,pos) -> string
```

Function Summary

LMNO

```
LCASE(string) -> string
LEFT(string,count) -> string
LEN(string) -> number
LOCK(cabinetname) -> boolean
LOG(number,base) -> number

MAX(number,number) -> number
MAX(shapename) -> number
MCM(number,ss,ps) -> string
MESSAGE(values)
MID(string,pos,count) -> string
MIN(number,number) -> number
MIN(shapename) -> number

NEWDRAWER(cabinet,g,n,v[,after]) -> drawer
NEWFILE(name) -> boolean
NEWFOLDER(drawer,n,v[,after]) -> folder
NEWPAGE(folder,form[,after]) -> page

OPENCABINET(cabinetname) -> cabinet
OPENDRAWER(drawer) -> boolean
OPENFOLDER(folder) -> boolean
OPENPAGE(page) -> boolean
ORD(character) -> number
```

P

```
PAUSE(seconds)
POS(substring,string) -> number
PRINTPAGE(printer,page,auto) -> boolean
```

R

```
RADIANTODEGREE(number) -> number
RANDOM(min,max) -> number
RANDOMIZE[(seed)]
READFILE(name,string) -> boolean
REMOVELEADING(string,chars) -> string
REMOVETRAILING(string,chars) -> string
REMOVEUNWANTED(string) -> string
RENAMEDRAWER(drawer,g,n,v) -> boolean
RENAMEFILE(oldname,newname) -> boolean
RENAMEFOLDER(folder,n,v) -> boolean
RIGHT(string,count) -> string
ROUND(number) -> number
ROUNDDOWN(number) -> number
ROUNDUP(number) RETURNS number
```

S

```
SAVEPAGE(page) -> boolean
SELECTDRAWER(drawer,flag)
SELECTDRAWERS(cabinet,flag)
SELECTFOLDER(folder,flag)
SELECTFOLDERS(drawer,flag)
SELECTPAGE(page,flag)
SELECTPAGES(folder,flag)
SIN(number) -> number
SORTDRAWERS(cabinet,sequence)
SORTFOLDERS(drawer,sequence)
SORTPAGES(folder,sequence)
SQRT(number) -> number
SUM(shapename) -> number
```

T

```
TAN(number) -> number
TRUNC(number) -> number
```

U

```
UCASE(string) -> string
UNFILEPAGE(page) -> boolean
UNLOCK(cabinetname)
UNLOCKALL
UPDATEPAGE(page) -> boolean
UPPERBOUND(shapename) -> number
```

V

```
VIEWCABINET(cabinet,drawer)
VIEWDRAWER(drawer,folder)
VIEWFOLDER(folder,page)
VIEWPAGE(page)
```

W

```
WARNING(values)
WRITEFILE(name,string) -> boolean
```

X

```
XCC -> cabinet
XCD -> drawer
XCF -> folder
XCP -> page
XCV -> view
XCX -> string
```

Index