Nota Bene

Customization & Programming Guide

by

Tony Woozley

Revised for Nota Bene 8.0

by

Mary Bernard

This revised edition is dedicated
to Tony Woozley, mentor and friend,
and to the memory of Dorothy Day

Preface to the Revised Edition of the CPG

The Customization and Programming Guide is not the work of one person or even of two people. Some of it has been around since Nota Bene 1, such as the examples using 'WEBER.DOC' in Chapter 4. Some of it may have come from a XyWrite manual. XyWrite is the word-processing program from which Nota Bene derived. Like NB it crossed the DOS barrier and became a Windows program; unlike NB, it is no longer commercially available. (Nota Bene and XyWrite are extremely similar; they both use the XyWrite Programming Language—XPL. NB uses it less in Windows than in DOS versions, but there are current examples in the *.AUX files in the main NB program folder. All but a few XyWrite codes (mostly DOS-related) work in NBWin.)

The first three versions of Nota Bene had a terse XPL programming section, the Customization and Programming Guide (CPG). NB 4, released in 1993, had a superb manual, the Big Black Book, but no programming guide. However, the company asked Tony Woozley, a retired professor of philosophy at the University of Virginia, to revise and update the old Guide. Tony rewrote and expanded it. In his hands it became lucid, elegant and informative, a true guide for the beginner as well as a work of reference for more advanced XPL programmers.

Nota Bene became a Windows program in 1998, with version 5.0 of the program. There was still no programming section in the manual, now called Help, and online rather than printed. XPL still worked (and works) in NBWin, though some user programs written for NB DOS needed minor revision. Tony's version of the CPG needed revising, too. Almost all the information about XPL was still valid, but a few older codes weren't, and it had whole chapters on NB DOS features that didn't work in NBWin, such as function OV.

Eight years late, here is a revision of the CPG manual for NBWin, up to date for version 8. It is still largely Tony's Guide; I have tried to match the tone and spirit of his version, insofar as possible.

I want to thank Carl Distefano for letting me include excerpts from his end of our email correspondence as Chapter 10, a 'Miscellany of XPL Information,' and Robert Holmgren for letting me include his 'Compendium of XyWrite/NB Variables' as Chapter 9. I owe them both thanks for their illuminating answers to my XPL questions over the past several years; and I especially want to thank Robert for debugging and hastily editing this file at the last moment.

I am grateful to the many members of the Nota Bene Users' list whose answers to my questions have helped me to make the Guide as accurate as possible. Particular thanks are due to Jukka-Pekka Takala, Joel Lidov, Michael Norman Jannik Lindquist and Rick Penticoff; and

to Steve Siebert, creator and chief programmer of Nota Bene, who answered questions about codes while getting NB8 out of beta. I'd like to thank him and Anne Putnam, president of Nota Bene, for letting me add information about customization and programming, and links to this Guide and other XPL resources, to NB8 online Help.

Printing the CPG

The Guide is carefully formatted for printing on both European A4 and American 8½" x 11" paper. Each page therefore ends with a hard page break. If you change them, pages and tables may break in awkward places.

If you have downloaded only the PDF version of the CPG, you may want to download CPG.NB as well. A few pages contain actual XPL code, which you can view in CPG.NB by changing to Codes View—which of course you can't do with the PDF.

Notes on the text of the CPG

I have not changed Tony Woozley's prefatory note to the DOS version except to update his email address and add a footnote about files.

Italicised notes in the text.
These are brief supplements to information that has survived otherwise unchanged from the DOS version of the Guide.

Faux program codes in the text
Since the CPG is formatted to be printable, function codes are represented by uppercase character in boldface (e.g., **BX**) and command brackets by Euroquotes (e.g., «SV01,Y»). You cannot copy and paste these characters into a file and run the file as a program. You can copy/paste them, but you must then change to Show Codes View, eliminate the strings that look (for instance) like: «MB+BO»BX«MDNM», and enter a BX code by doing 'pfunc bx'.
You can then do a global change of all « and » characters to command brackets (on Ctrl+, [Ctrl + comma] and Ctrl+. [Ctrl + period/full stop]). It is actually much easier simply to rewrite the program in real code, using the pfunc key and the keys that enter true command brackets.

Updates to the newGuide
With luck, the information in this version will be valid for a good while.

I maintain an unofficial list of all the keyboard and programming codes (or all that I can find) that are valid in Nota Bene for Windows. As of July 2006, the entire list is included in Chapter 8 of the CPG. I shall update the list whenever an update of NB includes a significant amount of new code, send it to Rick Penticoff for posting on his NB Users' website, and notify the NB List (see below) that it is available. You will be able to download it from: http://www.penticoff.com/nb/programming/nb-codes.zip. I shall list new codes at the head of the file as well as incorporating them into the alphabetical ordering, so that you can quickly check what is new.

Troubleshooting

Errors in the CPG

Please send corrections to mary.bernard@taffa.eu

Back Up Before Customizing

Before you do *any* customization whatever, back up. Back up your data files before running programs on them; back up your keyboard file before editing it. This is of paramount importance. It is very easy to make a slip while customizing a keyboard. NB may then load an ancient default keyboard with important keys in bewildering places. If you have a backup, you can open Windows Explorer and restore your original keyboard file to the c:\nbwin\users\default folder.

It is even more important to back up your data files before running a user program, whether you've written it yourself or downloaded it from another user or the users' website. Programs that work on one system don't always work on another. Programs you write yourself almost always have mistakes at first. Usually they simply don't run. But some mistakes can swallow data, such as the file on which you want the user program to work, or possibly some other file or files. And mistakes often cause NB to lock up. So save and back up before you begin trying out programs.

Disclaimer

These notes are provided by me, not by Nota Bene Associates, Inc. They have kindly allowed me to include them in the Help file, but Nota Bene Technical support cannot help with user programs or keyboard definitions that aren't working.

If you try your hand at programming, you should have enough experience with computers not to be too fazed by program and/or computer crashes. They will almost certainly happen as you write and test XPL programs—they happen to everyone who tinkers with programming.

If you are stymied, ask for help on the users' list (for how to join, see p. iv).

If you ask for help

It is hard to read someone else's XPL program if it is not broken up into lines and commented—it is even hard to read one's own programs a while after writing them. If you send a program to the NB user list in hopes of getting help with it, you *must* comment it—in the program, not just in your message. You need to include the purpose of the program, what is going wrong with it. You should break it up into lines, and precede each with a description of what you intend the line of code to do, e.g.,

```
    ;*; Label 3. Move cursor Left one character. Backdelete previous character. Insert paragraph marker
«lb3»«CL BD ↵
    ;*; Go to Label E
      «glLE»
```

See p 90 for information on how to comment a program.

Online Customization and Programming Resources

Rick Penticoff's NB Users' website
This is a major source of tips, manuals, user programs and useful links. It is at: http://www.penticoff.com/nb/index.htm

NBKEY.KEY—the Keyboard Table
This table shows the key assignments of all keys in all the shift states (Unshifted, Shift, Ctrl, etc.) in NB.KBD as delivered. You should print it out if you are doing keyboard customizing. It is in C:\NBWIN\DOCUMENT\SAMPLES.

Greg Polly's Tutorial
Greg Polly has written a basic tutorial on how to write a program in Nota Bene for Windows. It's at: http://www.penticoff.com/nb/help/howtorun.htm. You should use the codes lists in Chapters 2 and 8 rather than his appendix of command codes, which are taken from the DOS edition of the CPG.

XYWWWEB.U2
This is an extraordinary compendium of XPL programs, written and maintained by Robert Holmgren and Carl Distefano. They came to NB by way of XyWrite IV for DOS, Nota Bene's parent program, which is now maintained by them. It is safe to say that they know more about XPL programming than anyone except, perhaps, NB's own programmers.

The compendium is called XYWWWEB.U2, and you can download it from http://www.serve.com/xywwweb/. It contains hundreds of XPL programs. Some of them are only for XyWrite (noted in the documentation), but Holmgren and Distefano have worked hard at making much of U2 compatible with NB. You install U2 by unzipping it, copying its files into your main NB program directory, and copying a string to an empty key definition in your keyboard file: a key which will thereafter be your U2 help key. You can then run U2 programs by typing a mnemonic on the command line and striking your U2 help key.

The XyWrite Programming Language User's Guide
On the XyWWWeb site you can find a very detailed programming guide: http://www.serve.com/xywwweb/XPL.ZIP. It was written for XyWrite IV for DOS, which uses a version of XPL that is the same as NB's XPL. (The original CPG described an earlier and much less powerful version of XPL.) It has the disadvantage that you have to know how XyWrite differs from NB: for instance that F5, not F9, goes to the command line, and that Phrase keys are called Save/Gets. But it is a useful resource for anyone who gets serious about XPL programming.

The Nota Bene users' list
This is a helpful and friendly group of NB users, from complete novices to people who have been using the program for many years, have done a good deal of NB customization and programming, and are happy to share their experience. There is no such thing as a stupid question on the NB list, and you won't be told to Read The Manual.

For more information, or to subscribe, go to:
http://wnk.hamline.edu/mailman/listinfo/notabene

You can browse the List archive at: http://www.penticoff.com/nb/help/howtorun.htm

Jukka-Pekka Takala's website
Jukka-Pekka Takala is a long-time user of Nota Bene. His website, http://www.helsinki.fi/~jtakala/notabene.html, has some useful programs, especially unbal.run, which finds unbalanced command brackets better than NB's own 'Go, Illegal Format Code' dialog. (His BACKUPDIR.RN5 is also very good, especially if you want to back up each changed version as you save it (I have '31=bx,r,u,n, ,C,:,\,n,b,w,i,n,\,X,P,L,\,B,A,C,K,U,P,D,I,R,.,R,N,5,q2' on Ctrl+S instead of NB's own Save definition).

The Nota Bene Dragonfly

Nota Bene Associates was called Dragonfly Software for the first ten years of its life. A dragonfly was part of the logo and appeared on NB's stationery; users became attached to it. With version 8.0 the dragonfly has returned to hover on the program's icon, and on the cover sheet of this Guide.

The typeface of the dedication, and of 'Nota Bene' on the title page, is NB Daylight, a font commissioned by Nota Bene in memory of Dorothy Day, a long-standing and much-missed user of the program and member of the Nota Bene List.

Mary Bernard
mary.bernard@taffa.eu
12th April, 2006

Prefatory Note

by Tony Woozley 24 December 1994
cad2m@cms.mail.virginia.edu

This collection needs a little explanation. It is more about myself than I would like, but I cannot avoid that. In early 1992 I was invited by Deborah Reumann, on behalf of **Nota Bene**, to revise and expand the **Programming** chapter of their **Customization & Programming Guide**; I had misgivings about my competence to undertake the task, but for reasons that will appear below I accepted, and began work immediately. At a much later stage Steve Siebert asked me also to revise the preceding three chapters of the Guide. I agreed to tackle the first two chapters (**General Customization**, and **Keyboard Customization**), but declared myself unfit for **Printer Customization**, as there was far too much that I did not know about too many printers and printer drivers. Later, Tony St. Quintin undertook that task, but I do not know whether he had begun it before his health required him to give up his job as **Nota Bene**'s UK agent; I have failed in attempts to contact him.
More than a year ago I sent to Steve Siebert files covering the **Programming** chapter; and six months ago I sent him a revision of those, plus files covering the **General Customization** and **Keyboard Customization** chapters, with in addition a new chapter, on **Help File Customization**. I regarded my job as finished, except for making alterations and corrections, as required or recommended by Steve. In fact, as far as I know, nothing has since been done; and I do not know whether the files I sent to **Nota Bene** have even been read by anybody there. It would be easy to criticize them for that, but I am disinclined to, because I know something of the problems that they have had in re-establishing themselves in New York, and because a book of the kind that I have produced has to be low on their list of priorities: interest in, and a desire to learn something about, XPL programming, are bound to be confined to a small minority of NB users; and on a cost-benefit calculation the book would now, and for an indefinite period ahead, rate very low. On the other hand, I have good reason for wanting to get it off my shoulders.

So, I have decided to deposit all the files into our SimTel archives, where they will be available to any users interested in seeing them; and that will not preclude **Nota Bene** from publishing them, or something based on them, later.

The reason why I undertook the job in the first place was that I was bothered by the knowledge that there were so many NB commands that were undocumented, and unknown to most users. I wanted to do what little I could to repair that. For me it all began with a talk that Steve Siebert gave at the University of Virginia in April 1991. In the course of informal conversation afterwards he enthusiastically introduced a new string of XPL code, called Extract String, and also Parsing String, which had originated in XyWrite, and was now available in NB. I don't think any of us followed his brief exposition of it, but I took away from the meeting a scrap of paper on which he had written it down. This was all that it was:

«xs00,01,02,03,04»

I couldn't make head or tail of it, but much later, with the help of something written about XyWrite, I managed to work it out. It is one of the most ingenious and versatile codes in the whole XPL collection, and has been invaluable to me ever since, especially when used recursively, as in

«xs00,01,02,03,04»«xs04,01,05,03,06»....

I passed it on to our then-guru, Itamar Even-Zohar; and that was the first and only time that I have ever taught him anything about NB. A description of it can be found in XPLCALLS.DOC in this collection, and an example of its use in XPLSAMPL.DOC. There is also a full account of it in XPL.DOC (contained in NB-XPL01.ZIP) in our SimTel archives.

The point of that story is that it illustrates how much there is in NB, that is unknown to almost all its users, and that will remain unknown until documentation is made available to them. In what I have written I have tried to supply some of that. In that particular case four years have passed since the string of coding was introduced; but it is totally unknown to almost everybody not on this list, and probably also to a good many who are.

Contents, and Conventions

The file CONTENTS.TBL[1] shows the list of contents that the book is intended to have, with in the third column the names of my files that match the proposed chapters of the book.

It should be noted that, as in the original **Customization & Programming Guide**, what appears to be XPL coding in the text is not actual XPL coding, but a textual representation of it: the apparent functions are just the appropriate mnemonic characters in Boldface type, and the apparent opening and closing command brackets are European quotes. That makes reading of the files on screen easier, and makes printing of them possible. But, in any instance, the apparent XPL coding can be converted to actual XPL coding by running on it the program TXT2XPL which is included here. To use it on any pseudo-XPL string or program in any of the files, first copy the string or program to a new file, and then run TXT2XPL on that. If you run it on one of the actual files in this collection, the program will convert, not just the pseudo-XPL string or program that you want converted, but all those that follow in the file as well.

For those who do not want the trouble of converting pseudo-XPL to actual XPL programs there is one other file in this collection that is not mentioned in CONTENTS.TBL, viz. XPLSAMPL.XPL. This contains the sample programs in XPLSAMPL.DOC, in their actual XPL format. They will not need transcribing before they can be run as programs.'

Also included is the file XPL2TXT, which converts XPL coding to textual representation of it. The same precautions in using it apply.

In a collection of this length, all of it written by a single not particularly skilled person, there are bound to be many errors. It would be appreciated if users will call them to my attention, so that they may be corrected. I wish to end by expressing my thanks to my friend and frequent collaborator Jukka-Pekka Takala, who has already shown me the need for a number of changes. Without his help and initiative I doubt this collection would be finding its way into the archive at all.

---

[1] *There is not now a file CONTENTS.TBL in the CPG; nor does it include the DOS programs TXT2XPL,XPLSAMPL.XPL or XPL2TXT.—MB*

Table of Contents

Chapter 2: Keyboard Customization

Chapter 3: Introduction to XPL Programming and Functions

Chapter 4: XPL Program Calls

Chapter 5: Programming: Sample Programs

Chapter 6: Programming: Writing Programs

Chapter 7: Running XPL Programs

Chapter 8: Codes that work in Nota Bene for Windows

Chapter 9: Compendium of XyWrite/NBWin Variables by R.J. Holmgren . 131

Chapter 10: Miscellany of XPL Information, chiefly by Carl Distefano

Chapter 11: Codes Probably Out-of-Date or for XyWrite Only

Appendix I: NB DOS XPL Error Messages; XyWrite Error Messages

Index

General Introduction to Customization

Introduction to Customization

Nota Bene uses four files to implement certain commands and default settings during the loading process. They were either selected or created when you installed the program, based on the type of system you have and on the options you selected. Although you might never need to modify any of them, it is a good idea to become familiar with their function and structure. That way you will learn how they can be modified to provide greater flexibility to certain operations of the program, and you will be better able to detect problems that might be related.

The four files are:

1. NBSTART.INT can contain commands and programs that the user wishes to have executed whenever **Nota Bene** is loaded
2. / 3. NB.DFL and NB.INI determine default settings
4. NB.KBD determines what the computer does when any given key or key-combination on the keyboard is struck. The user can edit the KeyBoarD file, or create new KBD files for special purposes. Only one KBD file may be loaded at a time.

(There are other files that can be customized and loaded on startup; they include a personalized spell check/ auto-replace dictionary; phrase libraries; and user programs. They will be described later.)

The four files are ordinary text files that can be called to the screen, modified, and stored with the file-handling and editing commands. But:
NB: You must not edit NB.INI directly. Nota Bene writes to it during and at the end of a session, and you will get error messages, and possibly compromise NB.INI, if you edit and save it. You may well never need to edit it: most customizations can more safely be made through the menu dialogs 'Tools, Preferences' and 'View, Interface Options'. If, however, you decide to edit it directly, call it under a new name with:

F9 ne new.ini,nb.ini F10

(where 'new.ini' stands for a temporary filename of your choice). Make your changes; save the file; close Nota Bene; copy the current NB.INI to a safe place, such as a new folder named 'TEMP' under the C:\NBWIN\USERS\DEFAULT folder. Now rename NEW.INI to NB.INI; and open Nota Bene. If all is well, you can after a time delete the version in the temp folder.

If you have NB.DFL onscreen, do not try to change a default using the Tools, Preferences menu. You will get an error message when you click OK. (And be sure to back up NB.DFL to a safe place if you edit it.)

Remember that storing a modified settings file does not immediately implement the modifications. To do that, you must "reload" the table(s) with the corresponding run or load command(s) (see "Using the load Command" section).

This chapter starts with a brief explanation of the system path. It is not unique to **Nota Bene**, but it can be usefully modified to include **Nota Bene** paths.

It then covers the following **Nota Bene** table files:

NBSTART.INT—contains any commands that you wish to execute whenever **Nota Bene** is loaded.
NB.DFL and NB.INI—implement the program's default settings (and your modifications , made either through 'Tools, Preferences' and 'View, Interface Options' or directly.

NB.KBD is described in the "Keyboard Customization" chapter.

Other general customization instructions are also included in this chapter.

SYSTEM PATH in Windows 2000 / XP

The Windows NT4 / 2000 / XP System PATH is now managed by entries in the Windows Registry, although 16-bit programs like NB can still consult an AUTOEXEC file (now called "autoexec.nt", and located—unless you specify an alternative autoexec file—in the system32 folder). Autoexec.nt may be used to supply configuration *additional* to that found in the Registry.

It can be very useful to add one or two NB folders to the system path, principally the program folder itself and the folder where you keep your XPL programs. Then, wherever you are in Nota Bene (perhaps in a folder named C:\NBWIN\CPG), you can run user programs from the command line without specifying the full path:

```
run filename.run
```

instead of (for instance):

```
run c:\nbwin\xpl\filename.run
```

If your Nota Bene installation is on C:, and you have not modified your program folder name, the program folder will be C:\NBWIN. It is a very good idea to keep user XPL programs in their own folder: C:\NBWIN\XPL.

The disadvantage of adding to the System PATH is that it slow down your system noticeably if you add more than five or six folders.
You can add to the path either through My Computer or by using a freeware tool.

To change the path through My Computer, follow these steps.
1. From the desktop, right click My Computer and click Properties.
2. In the System Properties window, click on the Advanced tab.
3. In the Advanced section, click the Environment Variables button.
4. In the Environment Variables window, highlight the path variable in the Systems Variable section (the lower pane) and click Edit.
5. Go to the end of the Variable value line (do not erase what is already there) and add the Nota Bene path(s). Each different directory is separated with a semicolon as shown below.
C:\Windows\System32;C:\Windows;C:\Program Files;

You must specify the full path, e.g., C:\NBWIN\XPL. Remember to put a semicolon at the end of the existing line, before typing your addition. If you know what functions are performed by the various folders in the Path, you may reorder them to speed up access to certain folders; but in no case should any folder precede the operating system root folder (usually WINDOWS or WINNT) or the system folder (usually system[32]).

The maximum length of the combined system and user-defined path variables is 1,023 characters. This does not includes the "path=" portion of either.

Typing the path in the Environment Variables window is a bit fiddly and annoying, because you cannot see the whole of the existing path at once. An alternative is to use a freeware tool, such as System Path Commander, (http://www.softpedia.com/get/System/System-Miscellaneous/System-Path-Commander.shtml)
To add to the path using System Path Commander, you run the program, right click on the window, choose 'Add', and either type the path or navigate to it with the usual Browse button.

NBSTART.INT

The NBSTART.INT file ("INiTialization") is a table that provides you with the opportunity to implement settings of your choice whenever you load **Nota Bene**. Initially it contains no commands, because the necessary configuration, including loading of the keyboard table and default-settings tables, is now performed by NB.INI *before* any commands that you might add to NBSTART.INT (examples are shown below) would take effect.

Do not rename NBSTART.INT! (You can use alternative names for alternate versions, but if you wish NB to look for this file and auto-load commands that you have entered into it, it **must** bear the name NBSTART.INT.)

The NBSTART.INT table is a "program file" that contains "program functions" and is executed with the run command. To learn more about program files, see Chapters 3 and 7.

It should be either in the program folder (typically C:\NBWIN) or in C:\NBWIN\USERS\DEFAULT, where there is already an empty NBSTART.INT file for you to use. If you keep your NBSTART.INT in the program folder, you should delete the one in C:\NBWIN\USERS\DEFAULT, in case the command to run NBSTART.INT finds the empty file rather than the one you have modified.

Calling to Screen

If you want to check or modify the NBSTART.INT table, call it to the screen as an ordinary file:
F9 call nbstart.int **F10**
Specify C:\NBWIN\before the filename if you are not in the C:\NBWIN folder.

Rules for Modifying File

When modifying the NBSTART.INT table, be sure to follow these rules:

Each command except the first line must begin with a "BX" (see below) and must end with a "Q2". It is convenient to put each command on a separate line, in which case you must end each line with a comment string (;*;).

Make sure you are in Show Codes View, so that print mode commands will not be inserted into the file.

You can use lowercase or uppercase.

BX

BX is a "program function" that "blanks" the command line so that a new command can be executed (this is an over-simplification; see p 165 for details). **Q2** executes the command.

The **BX** and apparent space (which is really ASCII 0) are one unit, so the cursor jumps over the "X" and space.

Saving File

When you have finished modifying the table:

F9 sa **F10** (Or do Ctrl+S.)

Reimplementing

To reimplement the modified NBSTART.INT table:

F9 run nbstart.int **F10**

If you deleted items from the table, you must exit **Nota Bene** and reload the program. If you just rerun NBSTART.INT, some of the former settings might remain in effect if there are no new commands to override them.

NBSTART.INT is less useful than it used to be in earlier versions of NB for Windows. In NB 8, settings made via the menus, and saved in NB.DFL and NB.INI, usually override commands in NBSTART.INT. For instance, the directory sort order set through Tools, Preferences, Directories (Command Line) will not be overridden by the line:

BX order d,r**Q2**;*;

in NBSTART.INT. The chief uses of NBSTART.INT now are to load the XYWWWEB.U2 program compendium (see p iv) and to load programs on ampersand phrases (see pp 41, 99).

Programs

If you have programs that you want either loaded (say, to an ampersand phrase) or run whenever you load **Nota Bene**, you can do that here, by using:

BX ldpm x:[filename].run,&y**Q2**;*; for loading a program on an ampersand phrase
BX run x:[filename].run**Q2**;*; for running a program

Displaying Directory

If desired, you can have a directory displayed as the last step. There are two ways to do so:

BX dir**Q2**;*;
BC call

Include *one* space after the call command, so that the cursor will be positioned for typing the filename.

BX dir**Q2**;*;
BC call **CC**;*;

By adding the CC function as shown, you can have the cursor move down into the directory so you can position it on the file you want to call.

Choose the method you most frequently use for calling files.

Calling file(s)

If you want a particular file or files to open whenever you open NB, you can do it with lines of this type in NBSTART.INT:

BX ca [path\file1].nb**Q2**;*;
BX ca [path\file2].nb**Q2**;*;

Using the LOAD Command

These commands were more generally useful in NB DOS, where there were a number of files that could be **load***ed. But it is still useful to know how to load the few remaining files that can be loaded from the command line.*

There are two types of commands used to load table and other files into memory: the generic **load** command, which can load any type of table file that is properly identified; and **ld** commands, which are used to load specific kinds of files. There are two types of file for which the generic **load** cannot be used, and the appropriate **ld** command must be used instead:

Phrase libraries (filename.**lib**) can be loaded only with the **ldlib** command.

NB: this command does not load the [phraselib] .LIX file containing the phrase-library descriptions, but it can be useful for quick phrase-library changes if you know what is on the keys.

NB: phrase libraries can be saved from the command line with **salib**, but this command does not save the .LIX file. If you significantly modify your phrase library, and want to change the descriptions to match, you need to use the menu on Alt+F3.

Program files (filename.**run**) can be loaded only with the **ldpm** command.

NB: They can be *run* from the command line with the **run** command (and in other ways, see Chapter 7).

LOAD

The **load** command can be used to load one or more tables files at a time. Each table file, however, must start with a special four-character sequence called a **load ID**. These are the load IDs for each kind of table file:

File Type	Filename	Load ID
Keyboard table	filename.kbd	;KB;
Personal dictionary	filename.spl	;SP;
Default settings file	NB.DFL	;PR;

NB: do not load NB.DFL with the **load** command; you might cause a program crash. Instead, quit NB and re-open.

The load ID must be on the first line in a table file, and must be typed exactly as shown, i.e., in the sequence of semicolon, capitalized two-letter code, semicolon. The load ID should be terminated by a paragraph marker (↵).

The **load** command can be used either to load one or more table files, or to load one or more personal dictionary files simultaneously. To load a group of table files, use:

F9 load table1,table2,table3 **F10**

(note separating commas; no spaces between comma and following filename)

To load more than one personal dictionary, use:

F9 load file1.spl+file2.spl+file3.spl **F10**

(note separating plus signs)

If you already have a personal dictionary loaded and want to add another/others to use at the same time as the first, put a plus sign *in front* of the first dictionary file in the list.

NB.DFL

NB.DFL file is a table that implements many of program's default settings whenever you load **Nota Bene**. It is automatically loaded by the program before NBSTART.INT takes effect.

As can be seen when you call the NB.DFL table to the screen (preferably under a new name, as described on p 1) and scroll through it, there are many features of the program that you can customize. Some of these were set based the choices you made when installing the program; others are initially the same for all computer systems. Many of the settings can be changed as desired by modifying NB.DFL. But, until you have become thoroughly familiar with the operations of **Nota Bene**, it is strongly recommended that you make all changes of default settings by means of through the menu dialogs 'Tools, Preferences' and 'View, Interface Options'.

If you do edit NB.DFL, follow the new-name procedure described on p 1.

Effect of Settings

The settings implemented by the NB.DFL table affect all files that are called to the screen or printed unless the files have contravening commands (i.e., deltas) embedded within them.

If you modify the NB.DFL table, remember that display and printing of previously created documents that used the defaults—rather than embedded format commands—might be affected. Therefore it is normally best not to change the defaults frequently, but instead to insert actual commands in your files whenever you do not want to use the default settings. That way your files will always be formatted in accordance with the intended settings.

DF Settings

Most NB.DFL consists of lines setting defaults, along with descriptive comments (on lines beginning with a semicolon) and lines containing only a semicolon, the purpose of which is to make the file more readable by breaking it up.
The default lines are of the form:

```
DF XX=NN
```

For instance, here are lines setting the page width and length:

```
; PW is page width
DF PW=8.5in
;
; FD is form depth (page depth)
DF FD=11in
```

When specified in the NB.DFL table or executed with the default command (see next section), these settings must have an equals sign between the command and the value; when executed in a file, no space is used. For example, the form depth (as for legal-size paper) can be set in these three ways:

—as permanent default by **DF fd**=11IN in NB.DFL
—as temporary default by executing default (or d) fd=11in on command line
—for a specific document by executing fd 11in on the command line to embed the command as code.

Modifying Settings

When modifying the settings:

Always use Show Codes View. Inclusion of a hidden print-mode command or format code on any non-comment line will result in incorrect loading.

Comment lines may be added at any position in the table. Each such line must begin with a semicolon; any line that does not will be interpreted as an actual default setting.

Temporarily Changing Defaults

Defaults can be changed for the remainder of the current **Nota Bene** session—i.e., until you exit or turn off the computer—by using the Menu Line or the default command. Some changes will not become effective until you call the next file to the screen.

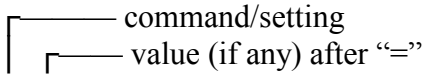
It can be convenient to change a default just for the active NB session. You can do so from the command line, with a command that is almost the same, except that 'DF' becomes 'default' (or 'd':). If you are writing programs, you might want to change the default view for opening files temporarily from Page Layout View to Show Codes View. This would do it:

d dt=0

The change will not become effective until you call the next file to the screen. To return to Page Layout View in mid-session, you would issue 'd dt=4'. Neither of these command-line commands will change NB.DFL.

Default Command

The form of the default command is:



F9 default xx=# **F10** or, **F9 d** xx=# **F10**

Where xx stands for the letters of the default (e.g., DT) and # for the specifying number(s) or letter(s) (e.g., DT=4, or FD=11.7in)
You can issue command-line commands in upper or lower case or a mixture of both.
Changing the page-layout defaults this way will not affect a file that is already on screen.

Can't set value with default — The command specified with the default command was improperly typed or cannot be input as a default. Be sure to use an equals sign (as default fd=11in) rather than a space (as fd 11in) between the command and values.

You can use the default command to test a setting before permanently modifying it in NB.DFL.
NB: Changing a display mode (e.g., 'd dt=0' to change the session default to Show Codes View) does not change the view of currently open files, only of any that you create or open after issuing the command.

List of Available Defaults

Chapter 8 includes all the defaults I have been able to find, with brief descriptions.

NB.INI

NB.INI loads your default printer and Windows fonts. It contains such settings (made through the menus) as your foot/endnote defaults; the folder NB opens in by default; whether or not you have Auto Check and/or Auto Replace turned on, the default sort order for directories; what beeps you have turned on (overstrike, etc.); your default phrase library and keyboard. It is where NB keeps the list of recently opened files that shows at the bottom of the File Menu; and where it lists the format and contents of its toolbars. It is possible to edit it by hand, but inadvisable, unless there is a setting that you cannot make stick through the menus.

Hyphenation Exception Dictionary

When automatic hyphenation is in effect, before breaking a word according to the algorithms, the program first checks the hyphenation exception dictionary to see how to break the word, or whether it should be broken at all.

A hyphenation exception dictionary (such as those supplied with the program) is an ordinary h text file that can be called to the screen and modified, if you want to specify where particular words should be broken. See Nota Bene online Help for full instructions.

Case

It makes no difference whether a word is lowercased or uppercased. However, capitalizing words that normally are capitalized makes it easier for you to find them in the list.

Order

Although the list does not need to be in alphabetical order, you can use the sort command to h alphabetize the list from time to time, so that it will be easier to find words.

Breaking Words

If you want a particular word to break, insert a soft hyphen with **Ctrl+/**

Nonbreaking Words

If you don't want a word to break at any point, insert a soft hyphen just *before* the word with **Ctrl+/**.

Phrase Libraries

This section will make more sense in the context of the XPL programming chapters.

The general topic of using, loading and saving phrase libraries is covered in online Help. This section is designed to supplement that information. It concerns what happens when you click 'Show Options'. The only difference is that there appears, just above the bottom line of buttons, a line with four choices:

Save as Program Insert: Command Set Command Key Function

First click in the type-in box and press Shift+F8 to change to Show Codes View.

Key Function:
Here you type function codes, such as BX, NP. They will immediately appear in the type-in box above.

Insert: Command
In the box beside this you enter embedded commands. Enter them in the form:

pv 01 md +bo [note space in each command]

Set Command
After typing an embedded command in the Insert: Command box, click this button to place the command in the type-in box above, as (in Show Codes View) «PV01», «MD+BO».

Save as Program
Saves the program you have just written in the box (or saved to a phrase key) as a program, rather than as literal text. If you do not check this box, then the program will be inserted into your file, rather than run.

On the other hand, it is easy enough to type the whole program into the type-in box (except the function codes, which you cannot input here except by using the Key Function box). Even in Show Codes View you will get an error beep as you enter command brackets, but if you can put up with that, it is quicker than the Insert Command/Set Command procedure.
Command brackets are also called format brackets, chevrons, double angled brackets or guillemets by long-standing Nota Bene or XyWrite users. You may encounter any of those names in the explanatory matter that accompanies user programs. Likewise, codes enclosed in command brackets may be called deltas.

Quicker still, if you want to save a program on a phrase key, is to write it in NB, highlight it, save it to a phrase key, open the phrase-library dialog with F3, highlight the phrase, click 'Show Options' and tick 'Save as Program

NB: This is not a quick or sensible way to write XPL programs or save them to phrase keys. I include it because (a) the topic is not covered in online Help; (b) if you have saved a program to a phrase key (as described in later chapters), it can sometimes be quicker to modify one small part of it in this dialog than to open the program, modify it, reload it on the phrase key and save the phrase library; and (c) this is the only easy way to modify a program which you have saved to a phrase key and erased from your hard disk.

Personal Spell Checkers and Auto-Replace / Auto-Expand

Again, this topic is well covered in online Help. What follows is a few notes about using auto-replace (or auto-expand).

1. The procedure for adding expansion pairs that is described on the 'Auto Expand' page of Help is slow and cumbersome. Your user spell file is an editable text file. Call it to the screen (you should probably keep it in C:\NBWIN\USERS\DEFAULT), make sure auto-expand/auto-replace is turned off, and type the expansion pair. You can put expansion pairs anywhere in the file. Maintenance is easiest if the pairs are in alphabetical order—you can put them in the right place, or put them at the end of the file and issue a 'sort' command.

You may want to put short-term pairs (e.g., specialised terms that you will use for one project only) at the end of the file, where you can quickly erase them when you are done with them.

The abbreviation must not include spaces, punctuation or formatting codes, but can include numbers.. Put one space between the abbreviation and the expansion string—which can contain spaces, punctuation or print formatting codes:

newb New Brunswick
3gm great-great-great-grandmother
dca *David Cameron's Adventures*

2. Online Help suggests using auto-replace for long names or phrases. For me, it shines as a way of speeding up the typing of the words I use most. I do auto-expand long phrases, but the real time-savers are the commonest words in the language:

o of	w with	tho though
mr more	f for	tre there
i I	bs beside	ev every
y you	bss besides	feb February
ty they	bt between	mond Monday
h he	b but	
s she	u and	

and hundreds of others, including every common contraction (apostrophes slow down typing no end):

dsn doesn't	il I'll	cdn couldn't
dnn don't	hl he'll	hsn hasn't
cnn can't	wv we've	hvn haven't

Of the first six words of the the string, ‘and hundreds of others, including every’ only one was typed in full. What I actually typed was: ‘u hundreds o ots, incl ev’.

NB: I have arranged the above expansion pairs in columns to save space; in a .SPL file each must be on a line of its own.

3. If you use auto-replace in this manner, with single or double-letter abbreviations, you need to be able to turn it off easily. I have it turned on by default, but I’ve changed the key definition of Ctrl+H (which is duplicated on Ctrl+Shift+H) to ‘35=az’, which toggles auto-replace on and off.

4. You can turn off the auto-correct/replace beep. Tools, Preferences, Sounds.

5. You can have different .SPL files for different purposes. Besides my everyday spell file, ABBREV.SPL, I have EMAIL.SPL (identical to ABBREV.SPL, except that the contractions have straight apostrophes, not curly ones); CDS.SPL, for entering conductors, composers, etc. into IbidPlus; XPL.SPL for entering programming-code strings into programs (see 92 below); and BOOK.SPL. This last is a only spell checker, not an abbreviation-expansion file—it contains words like Maliseet and Munsterberg that are in my book but not in the main dictionary, so that I can spell-check the book without being stopped every few paragraphs.

Introduction to Keyboard Customization

The Keyboard is a good place to start customizing. Keyboard customization is easier than XPL programming, and the keyboard table itself contains lots of examples.

Keyboard Table

The "keyboard table" is a file consisting of "keyboard functions" that are loaded into computer memory to tell the computer exactly what to do when each key is pressed—either by itself or when a shift-type or toggle key is used.

Short Glossary of Keyboard Table Terms

Keyboard table—a file with extension .KBD, which defines what the keys and key combinations do in NB in all the keyboard states.

Keyboard state—In the unchanged NB.KBD these are: Unshifted; Shift; Caps; Shift+Caps; Ctrl; Ctrl+Shift; Alt; Alt+Shift; Ctrl+Alt; Ctrl+Alt+Shift.

Shift state—the same as keyboard state.

State table—keyboard state/shift state

Keyboard functions—two-letter mnemonics that stand for XPL functions. They are used within keyboard tables to tell the program what editing or other operation to perform when a key is pressed. For instance, **bc** goes to the command line, removing any text that was on it (it can stand for 'Begin Command' or 'Blank Command') and **xc** executes a command typed on the command line (it stands for eXeCute).

Comments, commenting—Every line in a keyboard table that begins with a semicolon is a comment, and will be ignored by NB. You can add as many comments as you wish.

Key definitions—These start with a number followed by an equals sign, followed by code and text, separated by commas. Each shift state consists of a number of key definitions.

Key assignments—the same as key definitions.

The Keyboard File

Nota Bene's "standard" keyboard table is called NB.KBD; it is in the folder C:\NBWIN\USERS\DEFAULT. You should keep any other keyboards that you make in the same folder. Alternative keyboard tables (which also use the .KBD extension) can be installed. A number of language-specific keyboard tables come with Nota Bene, including British English, German, Dutch, French, Spanish, Italian and quite a few more. A XyWrite

keyboard is also available. You can load them via Tools, Keyboards, Select Active; you can save the active keyboard table as the default via Tools, Keyboards, Select Defaults.

The Nota Bene keyboard controls most functions including text entered on the screen, but the Windows keyboard controls text entered in the dialog boxes. If you are loading one of the Nota Bene foreign language keyboards, you should also load the corresponding Windows keyboard. Keyboards that use the Hebrew, Greek and Cyrillic alphabets are available in Lingua Workstation, but are not included in Scholar's Workstation.

Modifying
You can easily modify the keyboard table to include other editing keys or character assignments—or perhaps to assign existing operations and characters to different keys. You can also create new state tables, thereby considerably increasing the number of keys available to you for redefinition. This chapter explains the structure of the keyboard table and how it can be customized to meet your particular needs.

Backup and/or Save under new name
If you want to modify your keyboard table, you should back it up first. It is very easy to make a slip while customizing a keyboard. NB may then load an ancient default keyboard with important keys in bewildering places. If you have a backup, you can open Windows Explorer and restore your original keyboard file to the c:\nbwin\users\default folder.

Better still, copy NB.KBD under a new name, and make changes to the new keyboard. (You could call it TRYOUT.KBD, or NEW.KBD.) When you are satisfied with it, you can make it the default keyboard table (see above). This is a sensible move, because every update of Nota Bene overwrites your old NB.KBD with the latest version. The existing version does get saved in C:\NBWIN\USERS\DEFAULT\CUSTOM; but if your customized keyboard has a new name, and you have made it the default, NB will honour that, and you will not have to do anything at update time except call the new NB.KBD to screen, alongside your customized kbd file, and use either the file comparison keys or the Proof, File Comparison dialog to find changes to NB.KBD.

The comparison process will stop not only at each new or changed key def in NB.KBD, but also at each of your customizations. This can be a nuisance. I obviate it by saving a copy of each new version of NB.KBD as VANILLA.KBD. After an update, and before copying the new NB.KBD to VANILLA.KBD, I compare the two files, flag all changes, and copy the changed key defs to my customized keyboard file.

Keeping track with keyboard diagrams
It is easy to forget what key customizations you have made. If you do much keyboard customizing, you may want to: make a keyboard diagram with NB's key numbers on each key; print out a diagram for each shift state of the keyboard, and label the appropriate keys with your customizations. Appendix II is diagrams of the American and British standard Windows XP keyboards, labelled with NB's key numbers.

Keyboard Identification

The keyboard table begins with lines that tell the program:

- that the file is a keyboard table
- the total number of keys on the keyboard
- which keys are shift-type or toggle keys, and the names used for them in the state tables (see later section)
- what character, if any, should be displayed on the status line

The keyboard table must contain these statements. The only situations in which you should modify any of them are:

- to change shift-type keys to "single shot" keys to facilitate use by persons with typing disabilities
- if you want or need to make a major customization of the keyboard that requires different key identities

Normal Settings

```
    ┌──────────┐
    |;KB;      |          ─── identifies file as keyboard table
.....................................................
    |KEYS=107  |          ─── total number of keys (see note
below)
    |CTRL=29,99 |          ─┬─ shift-type keys indicated by
    |ALT=6,98   |          ─┤  key number only
    |SHIFT=42,54|          ─┘
    |CAPS=58,T: |          ─── toggle keys indicated by key
    |          |        "T", and character for status-line
indicator
    |MOUSE=105  |
    └──────────┘
```

The "load ID" (;KB;) at the beginning of the file is necessary so that the keyboard table can be properly loaded with the load command.

The total number of keys is set at 104 so that the standard keyboard table can also be used with the IBM enhanced keyboard. The second values for **Ctrl**, **Alt** and **Shift** are for the duplicate keys on the IBM enhanced keyboard. (Any key assignments for keys 85-104 are ignored when not using the enhanced keyboard.)

The spelling must exactly match that used in the "table=" lines in the following keyboard-state tables.

Basic Modification Procedure

To modify a particular key in a given state table (see **State Tables** below)

1. Find the start of that table (table=.....)
2. Find the key in the table (##=.....)
3. Modify/Insert the key's definition
4. Save the modified key table to disk, and reload it with

   F9 save F10

   F9 load <filename>.kbd F10

If you are not sure that you will like a change you are making, save the modified keyboard table under a different name (e.g., NB2.KBD) and load it for testing. If the change is acceptable, then save it under its original name.

Always use Show Codes View when modifying a keyboard table. Inclusion of a hidden print-mode comand or format delta on any line containing a keyboard identification, table identification, or key definition will result in incorrect loading of the keyboard table.

Do not delete the

```
;KB;
```

load ID at the beginning of the table; if you do, you will be unable to load the table (see **Using the LOAD Command** in General Customization" chapter).

If there is an error in the file as a result of the editing that you have done, a message of this form will appear on the status line when you try to load the file:
Bad:——A message indicating that the keyboard table contained an incorrect line.
The message flashes by very quickly; and you may have to repeat the load command several times nefore you can catch it.

Key Numbers
The keyboard table consists of key numbers followed immediately by an equals sign and the key definition. The numbers are determined by the computer hardware. Appendix 2: Keyboards contains diagrams of the standard modern US/UK Windows keyboards, with Nota Bene's key numbers superimposed.

The program compendium XYWWWEB.U2 provides a quick way of finding out a key code, if you have it installed: SCAN + Helpkey will report the scancode of the next key pressed.

The file C:\NBWIN\DOCUMENT\SAMPLES\NBKEY.KEY lists the existing key assignments in the vanilla keyboard table, NB.KBD.

If you do much keyboard customizing, you may want to make a diagram of the keyboard, blank except for the key names (e.g., A, F10) and numbers (e.g., 30, 68); print out one for each shift state of the keyboard; and label the keys with your customizations. It is easy to forget what customizations you have done, and where they are; and it is easier to shuffle through a few sheets of diagrams than to open the keyboard table and trawl through for changes, even if you have commented them.

Moving Definitions and Redefining Keys
There is nothing sacrosanct about NB.KBD's key assignments. With a few exceptions, mostly standard Windows keys (see p 163), you can move or copy any keyboard definition, whether NB's or your own, from virtually any key, to virtually any key.

Some key definitions are duplicated in NB.KBD. For instance, function TS, which toggles program recording mode, appears four times. It makes sense to keep TS on only one of these keys and replace the others with your own key definitions.

NB: Do not put any definitions on the Ctrl, Shift or Alt keys, in any of the keyboard states. You can check their key numbers at the top of the keyboard file.

Available Keys
You will probably start with a renamed version of NB.KBD. Some keys in vanilla NB.KBD are already free for you to use for your own assignments (though they may have assignments in future versions of the program). Any line in the keyboard table that consists only of a semi-colon and the key number; or of the key number defined as

```
##=NO
```

or

```
##=NO,NO
```

is available for the purpose. So are any keys that are not listed by number. For instance, if the key numbers in a particular keyboard state table skip from '34=....' to '36=....', then key 35 is available. This does not apply to keys like 85 and 89, which simply do not exist; and you would do well to steer clear of defining system keys like NumLock (69) or 84 (PrtScn) in any keyboard states, at least until you are experienced in keyboard customizing.

It is fairly easy for Scholar's Workstation users to find empty keys in the keyboard table for their customizations—the Ctrl+Alt table has a number of spaces. It is harder for Lingua users, but there are some spaces; and you can remove any of the accented characters in the Ctrl+Alt and Ctrl+Shift+Alt keyboards that you don't use. With accented characters that you do use, but rarely, you could consider inputting them with the F6 Accents and Modifiers popup, thus gaining the keys they were on for customizations.

A quick way of finding empty keys is to search for the NoOperation function 'NO', which is generally assigned to empty keys.

State Tables

Keyboard States
The keyboard table is divided into separate tables for each of the shifted and toggled "states" of the keyboard. Each "state table" consists of key definitions that create an entirely new "keyboard." Each state table begins with a line identifying the keyboard state:

```
                ┌──── name of shift or toggle key(s) used with
table=                          following key assignments
```

The spelling (but not case) of the name must correspond exactly with that in the keyboard identification at the very beginning of the keyboard table file.

Nota Bene has 10 different keyboard "states" (the maximum possible is 20), as established by the following table-definition lines (shown together here):

```
table=                  — unshifted
table=CAPS
table=SHIFT
table=SHIFT+CAPS
table=CTRL
table=CTRL+SHIFT
table=ALT
table=ALT+SHIFT
table=CTRL+ALT
table=CTRL+ALT+SHIFT
```

General Rules
Do not define the same key twice in the same keyboard-state table.
If you do define a key twice, the second definition is the one that is used.

The order of key numbers within a state table makes no difference. However, it is best to have them in numerical order, so that you won't accidentally duplicate an assignment because the number was out of sequence.

All keys not listed in a particular state table are dead in that state, but most can be used for user key definitions.

The shift and toggle keys (29, 42, 54, 56, and 58) appear in each state table with ASCII 0 (which looks like a space) assigned.

Esc (key 1) should not be modified in any way.

Key Definitions

Basic Format
Within each keyboard state table are the actual definitions of keys. The format for all such assignments is:

```
   ┌─────────── number of key to be used
   │ ┌──────── character and /or keyboard function(s)
   ##=
```

The numbers are determined by the computer hardware.

If a paragraph mark "↵" is immediately after the "=" sign, the keyboard table will not load. It will also not load if a line begins with anything other than:

#=
;
TABLE=

Types of Definitions
The different types of key definitions are:

Character Assignments: The character typed by the key is specified in correct lowercase or uppercase form. Examples: a, A, 5, *, +, !

Editing Operations: A two-character "keyboard function" or a group of functions is assigned to the key to conduct an editing or other operation. These can include moving the cursor to the command line to type and execute a command.
In Nota Bene for Windows, a good many editing operations invoke C:\NBWIN\NBMAIN-#.AUX files, with string such as:

```
14=[U,&X,B,D,U]
28=FF,&X,C,R
83=[U,RC,U]
```

Keys with &+letter cannot be used in Program Recording Mode (see later chapters), but they can be copied and modified within a keyboard file.

Comma
Commas—not spaces—are used between characters and/or keyboard functions. If you want to have a key insert a comma as part of a command or text, you must use the **co** function at the point in the definition where the comma is desired (see **Keyboard Functions** below for example).

End of Definition
All material up to the paragraph marker (↵) is part of the definition. Do not break a key definition into more than one line by pressing the Enter key; the line will automatically wordwrap if too long. (If a long definition includes spaces to be typed on the command line, a line break might occur at such a space.)

Comment Line
Any line beginning with a semicolon is a comment line. You may also use the NBWin comment string—;*;

When making a modification that you are not sure of, copy the line(s) with the existing definition, place a semicolon in front of the original to deactivate it, then modify the copy. If the modification does not work correctly when you save and load the table, you can easily reactivate the original definition by removing the semicolon.

You can also add descriptive comments. Keyboard tables make for slow reading and skimming, so you might want to standardise the layout of your comments. This would be one possibility, with comments indented from the semicolon:

```
;          95 - Delete phrase
95=yd,xd,df,bx,s,e, , ,^,S,^,S,^,O,^,R,^,O,^,T,q2,df,rd
;         100 - Go to end of previous line
100=pl,le
```

Or you could use a distinctive character for comments, such as a string of equals signs:

```
; =====95 - Delete phrase (from NB3.KBD)
95=yd,xd,df,bx,s,e, , ,^,S,^,S,^,O,^,R,^,O,^,T,q2,df,rd
; ====100 - Go to end of previous line
100=pl,le
```

Character Assignments

A key definition that consists just of a character inserts that character into the file. This is true for:

—keys 2-13, 16-27, 30-41, and 43-53 on the **Unshifted, Shift**, and **Caps Lock** keyboards [in the US version of NB.KBD. British and European keyboards have different assignments for, for example, key 40].

—accented characters on the **Ctrl+Alt** and **Ctrl+Shift+Alt** keyboards

—certain other keys (e.g., \ on **Ctrl** keyboard)

Duplicate Assignments

The same character can be assigned to more than one key. For example, many monetary symbols are on the same or different keys on different keyboards. No matter what key is used, the same character is entered into the file.

Adding New Characters

Most of the displayable characters that are part of the ASCII standard and extended character sets are already assigned to keyboards. To assign a character that is not, use **Ctrl+Shift** and the ASCII number on the keypad to enter it into the keyboard table, or use the Compose Key with **F6**.

Modifying Character Assignments

You can modify the character assignments in any way you want. For example, if you use only a few of the letters from the Multilingual keyboards, you can assign them to keys on a different keyboard or remove them. It is a good idea to substitute

```
##=NO,NO
```

for a character if you remove it without substituting a key definition of your own—it is easier to see empty keys defined as NO,NO than to hunt for numbers missing from a shift state.

You can also rearrange your keyboard so that it contains only characters and definitions you use, in positions you find easiest to use and remember. For instance, you could change the unshifted and **Shift** keyboards, as well as the corresponding **Caps** and **Caps Shift** keyboards so that they would have the Dvorak character arrangement rather than the "Qwerty."

Inserting Words

To have a key insert more than a single character, the definition must begin with the "no operation" (**no**) keyboard function or another suitable keyboard function:

```
##=A,r,t,i,c,l,e                    types only "A"
##=no,A,r,t,i,c,l,e                 types "Article" in text area or on command line, depending on where cursor is
##=no,si,A,r,t,i,c,l,e              first changes to Insert mode
##=gt,si,A,r,t,i,c,l,e              inserts word in text even if cursor was on command line
```

Keyboard Functions

"Keyboard functions" are two-letter codes used within keyboard tables to tell the program what editing or other operation to perform when a key is pressed. They occur in keyboard tables in three different forms.

1. The two letters of the code are contiguous in the table. For example, in the unshifted table state:

68=xc executes a command on the command line (This is F10)
91=ti toggles insert/overstrike (This is the Insert key)
102=cd moves cursor one line down (This is the Down Cursor key)

2. But many functions are executed through one of **Nota Bene**'s NBMAIN-#.AUX files. In that case the two letters of the function are separated in the table by a comma, and are preceded by a code such as &X, &G. For example:

81=&X,D,N next page (this is the Ctrl PgDn key)
60=&X,D,F start or finish free-form defining (this is the F2 key)

3 Still others are enclosed in square brackets+U; some of these include &+letter codes, some do not.

14=[U,&X,B,D,U] backdelete character (this is the Backdelete key)
83=[U,RC,U] delete character (this is the Delete key
46=[U,MW,C,P,U] copy to the Clipboard (key Ctrl C)

Keyboard functions can be in lowercase or uppercase within the keyboard table. In this chapter they are shown in lowercase and are boldfaced to distinguish them from immediate commands, in bold-underline.

Be sure to distinguish keyboard functions from commands, default settings and program calls—many of which often consist of the same letters.

Keyboard functions can be included within programs as program functions (see "Programming Functions" chapter).

Available Keyboard Functions

Topical and alphabetical lists of the available keyboard functions are given at the end of this chapter. If you are not sure of the effect of a particular keyboard function, use the function **(func)** command to experiment (see next page).

Including Command

To have a key execute a command, you must:

(1) Specify **bc** or **bx** to position cursor on blank command line
(2) Type the command. Separate the characters (including any spaces) by commas.
(3) Specify **xc** or **q2** to execute the command.

NB: **bc** must be used with **xc** and **bx** with **q2**. You cannot mix them. **bx** with **xc** will not execute.

Examples of commands on keys:

##=bx,r,u,n, ,\,n,b,\,x,p,l,\,l,h,-,q,u,o,t,e,.,r,u,n,q2 runs user program lh-quote.run
##=bc,s,t,o,r,e,xc executes **store** command

Including Comma
To include a comma as an element of a command assigned to a key, use the co keyboard function. Example:

##=bc,i,p,5,i,n,co,5,i,n,xc to execute **ip 5in,5in** command

Paragraph Marker
To have a key enter a paragraph marker into the text, either include the paragraph marker definition on Unshifted 28:

FF,&X,C,R

or open Edit, Find and Replace, and use the red and blue button to the right of 'Find' to insert a Carriage Return (Alone) character in the Find box. From there you can copy and paste it into your key definition. (Do not use the regular paragraph marker; it will not work.)

The Carriage Return (Alone) character or FF,&X,C,R string starts a new paragraph or executes a command, depending on whether the cursor is on the command line or in the text at that point in the key definition. **F10 (xc)** always executes a command.

Tab
To include a tab within a key definition, simply insert a tab character with Unshifted [tab key].

Command Brackets («...»)
If you want to create a key that searches for a certain type of embedded command, do not use the normal command-bracket keys (**Ctrl+, or Ctrl+.)** to enter the command brackets in the keyboard table. Instead, press and hold down **Ctrl+Shift**, and type the numerals) **174** or **175** (from the alphanumeric keyboard, not the numeric keypad. The characters that result will look like uppercase 'E' and 'F'.
An example of such an assignment is a key that searches (in Show Codes View) for every point in the file where you changed the point size (**sz#pt**), so that you can check whether the value is correct:

##=bc,s,e, ,\,E,s,z,\,xc (where the 'E' has been entered with Ctrl+Shift+174)

This method of inputting the command brackets can also be used in programs to search for command brackets.

A key assignment that executes an embedded command should not use these characters. Instead, define the key using **bx** and **q2** so that it implements the command on the command line (see 'Including Command', p 21).

Function Command
To test what a keyboard function does before you include it in a keyboard table, or to use an unassigned keyboard function at any other time, use the function (**func**) command:

┌—keyboard function

F9 func xx F10

That will execute the function. For instance, func rc will delete the character under the cursor. It will work if the keyboard function is of the form 'xx', as in **bc**, **xc** etc.

Changing the Windows Key Assignments for Control, Shift, Alt and Caps Lock
This is not, properly speaking, a Nota Bene subject. I am including it because a number of users on the Nota Bene Users' List ask how to change these shift-state keys.

Since NB users are constantly pressing key combinations such Ctrl+Shift+Alt or Shift+Alt, plus a letter, number, Function key or keypad key, it can be useful to exchange Caps Lock and Alt, so that the left-side Ctrl, Sh and Alt keys are in an easy-to-reach line, and can be confidently pressed without looking down. Furthermore, since Right Alt, (unlike like Right Shift and Right Control) does not have the same action as its left-side counterpart, you will probably, if you are a touch typist, crowd customizations that include the Alt key onto keys that can be reached with the right hand. It can therefore be useful to redefine Right Alt so that it has the same action as Left Alt.
This cannot be done within NB, alone, but it can be done. There are a number of freeware keyboard remapping utilities on the web. I use Keytweak:

http://webpages.charter.net/krumsick/

—but it is only one of many. Like all online remappers that I have found, it has the limitation of being based on the US keyboard. It will happily remap Ctrl, Shift and Alt (not to mention the Win key), but it does not recognize key 86.
You do not need to alter the numbers at the head of your NB keyboard table (see p 15 above); the rejigging has been done for you, in the Windows Registry.

Creating New Tables
In addition to redefining keys, and defining keys that do not already have definitions assigned to them, you can create entirely new state tables (within a maximum total of 20), thereby greatly increasing the number of keys available to you for redefinition. If your keyboard is the 104-key keyboard that has the CTRL, SHIFT, ALT keys duplicated, (and if the one key of a pair has a different scan code from the other), you can separate the duplicates into distinct keys by making changes in the initial table. For example, you can change the line that reads:

CTRL=29,99

into two lines

LCTRL=29
RCTRL=99

—thereby making the Left and Right CTRL keys distinct from each other. You now could keep LCTRL with the existing CTRL definitions, and create an entirely new set for RCTRL. (You must remember to change the name CTRL that occurs in subsequent state tables in which it occurs to either LCNTRL or RCNTRL.) You can do exactly the same thing with the SHIFT and ALT keys. There is one price that you have to pay—that you need to remember, when typing, which of your CTRL keys does which; and, if you are a touch typist, that is likely to be a considerable handicap. You might find it preferable to add one or more entirely new tables by making some other key(s) into Shift-type keys. Any key on the keyboard can be converted in this way, but the key then loses its present function, which, for almost all keys, would be an unacceptable inconvenience. However, if you have a 104-key keyboard, you have a number of duplicate keys, such as *, /, INS, DEL, etc. You have to insert in the

initial table a line that consists of the name for the key, say SLASH if you wanted to use one of the / keys (53 and 94), followed by an = sign and the key's number, as in:

SLASH=94. (The name for a key in the initial table can be anything you please, except that it must contain no numbers: F11 would not be acceptable—FXI would be.) You can now create a 'SLASH=' table, in which any key will, when you strike it while holding down the SLASH key, operate as you have defined it in that table.

There is an even simpler change that might recommend itself to many users. The CAPS key at present is not a Shift-key but a Toggle-key left over from typewriter days: when it is in one position all alphabetical characters are typed on the screen in lower case, in the other position in UPPER CASE. But, if like most users, you have very limited use for extensive typing in upper case text, that key is being very largely wasted. If you were to turn it into a shift-type key, the keys in the table=CAPS and the table=CAPS+SHIFT state tables would all become available for redefinition. You need to change the line in the initial table from:

CAPS=58,T: (removing the T, which marks it as a toggle)

to

CAPS=58

From now on the CAPS key behaves like a normal Shift-key. At the same time it is possible for you to continue to have the old benefits of the CAPS key by loading the following short program on to a phrase key (see p 10 for discussion of phrase libraries):

GT «sv09, »«lbRK»«sx50,«rc»»«if«is50»==«is09»»«ex»
«ei»«sx50,@upr(«is50»)»«pv50»«gLRK»

You can convert that textual representation of the program to the actual program by taking two steps (do it in Show Codes View):

i. Open a new file, and copy the new program to it.
ii. Create a program (see chapters on XPL programming) that reads:

TF XP BX ci /«/E/**Q2 BX** ci /»/F/**Q2**

(where 'E' and 'F' are input with Ctrl+Shift+174 and 175, as described above)
When you have finished, the new program will, in Show Codes View, have genuine command brackets surrounding the expressions.

Whenever you strike the phrase key to which you have loaded that program, the characters that you then type will appear in uppercase until you cancel it by striking **Shift/Alt/PrintScrn**

Examples of User Key Definitions

Assigning Leader

You can assign a leader (**ld**) command to a key to produce a hairline leader (using the "–" character):

##=bx,l,d, ,–,q2	inserts leader at cursor position, separating any existing items on line
##=lb,bx,l,d, ,,q2	inserts leader in front of any existing items on line
##=lb,bx,l,d, ,–,q2,FF,&X,C,R	inserts margin-to-margin leader, forcing any existing text down to next line

All of the above key definitions work as indicated if the cursor started in the text area. To ensure the same result if the key is used when the cursor is on the command line, add a **gt** at the beginning:

##=gt,lb,bx,l,d, ,–,q2,FF,&X,C,R	inserts margin-to-margin leader even if cursor was on command line

Delete and Backdelete by phrase

You can move the cursor to the previous phrase and next phrase (Alt+[, Alt+]), and highlight the next phrase (Alt+\);, but in NB.KBD you can't delete or backdelete by phrase. Here are key definitions (adapted from the NB3 keyboard file) that let you do it:

```
;  Backdelete phrase
##=yd,xd,df,pw,bx,s,e,b, , ,^,S,^,S,^,O,^,R,^,O,^,T,q2,cl,nw,df,rd
;  Delete Phrase
##=yd,xd,df,bx,s,e, , ,^,S,^,S,^,O,^,R,^,O,^,T,q2,df,rd
```

Zoom by 1%

NB zooms window size in 5% increments. To zoom in and out by 1% increments, you could add these two keys, and perhaps a third to return to 100% view (these are from J-P Takala):

```
##=bx,z,o,o,m, ,-,1,q2
##=bx,z,o,o,m, ,+,1,q2
##=bx,z,o,o,m, ,1,0,0,q2
```

Abandon a file without having to confirm

To abandon a file without getting a message box asking if you want to save it:

```
##=bx,a,b,q2
```

Remove hard page breaks

To remove all hard page breaks from the cursor position to end of file (and be told, on the command line, that it's been done):

```
##=bx,c,i, ,/,E,P,G,F,/,/,q2,bc,P,G, ,c,o,d,e,s, ,r,e,m,o,v,e,d, ,f,r,o,m, ,h,e,r,e, ,t,o, ,e,n,d,
,o,f, ,f,i,l,e
```

('E' and 'F' are input as described on p 22 above.)

Change preceding punctuation mark

These definitions are useful if you do a lot of rewriting.

To remove the punctuation mark at end of the word preceding the current word (e.g., change 'keys, and' to 'keys and'—useful if you do a lot of rewriting):

```
##=ql,qr,bc,s,e,b, , ,wa,ws,xc,qr,rc,qr,ch
```

To put a semicolon, colon, question mark or comma at end of the word preceding the current word (e.g., changes 'keys, and' to 'keys: and'). Can be adapted to any punctuation mark.
NB: Function 'co' must be used instead of an actual comma, since keyboard tables use commas as separators.

semicolon:	##=ql,qr,bc,s,e,b, , ,wa,ws,xc,qr,rc,;,qr,ch
colon:	##=ql,qr,bc,s,e,b, , ,wa,ws,xc,qr,rc,:,qr,ch
question mark:	##=ql,qr,bc,s,e,b, , ,wa,ws,xc,qr,rc,?,qr,ch
comma:	##=ql,qr,bc,s,e,b, , ,wa,ws,xc,qr,rc,co,qr,ch

Copy highlighted material to adjacent window; undefine in this one

##=as,yd,cp,as,gt,xd

This leaves the cursor in the first window. To go to the adjacent window after the copy, add 'as' (preceded of course by a comma) to the end of the definition.

Place marker like NB4's

NB4's place marker was pretty good. NB for Windows has bookmarks, which don't survive from session to session, and annotations, which do. There are two problems with annotations.

—They can't be empty. If you press Ctrl+Shift+Alt F1, you have to enter at least one character before the dialog will close.

—You can't search for and delete them in Page Layout View.

But if you put the following on 3 keys, you have a quick, easily searchable, permanent (till you delete it) marker. (The marker is on uppercase 2 in the Insert, Special Characters, Text Characters menu).

```
##=no,•                    [the NB4 marker character]
##=bx,s,e, ,\,•,\,q2       [search forward in file for marker]
##=bx,s,e,b, ,\,•,\,q2     [search backward in file for marker]
```

Lists of Keyboard Functions

These topical and alphabetical lists contain most of the two-character function mnemonics needed for keyboard customization. For a full list of all the codes that it is possible to use in Nota Bene, see Chapter 8.

TOPICAL LIST OF FUNCTIONS

Name		Description
ASCII NUMBERS		
r1	Record ASCII 1-	input ASCII digit 1- input ASCII digit 9
r9	Record ASCII 9	
r0	Record ASCII 0	input ASCII 0
CELLULAR TABLES		
ec	End Cell	move cursor to end of current cell
ed	Entry Define	select current row of cells
mc	Mark Column	mark column - select cell at cursor location in a table
tl	Table Left	move cursor to previous cell
tr	Table Right	move cursor to next cell
COMMAND LINE		
bc	Blank Command	move to beginning of command line
cc	Change Cursor	move between command line & text
xc	eXecute Command	execute command on command line
ch	Clear Header	clear command line w/o moving cursor
gh	Go to Header	move cursor to command line
gt	Go to Text	move cursor to text area
s-	Show last command	displays last command on command line
bx	Blind eXecute	execute command without putting it on command line
q2	execute command	finish command started with BX
COPYING & MOVING DEFINED BLOCK		
cp	CoPy define	copy block to cursor position
mv	MoVe define	move selected block to cursor position
un	paste from clipboard	paste copy from clipboard. (In Page layout View, in NB Lingua, an invalid «XAEnglish» code is inserted as well as the text.)
COUNTERS		
c1-c9	Counter 1 - Counter 9	insert counter («C1») in text insert counter («C9») in text
c0	Counter 0	insert counter («C0») in text
CURSOR MOVEMENT IN TEXT AREA		
cl	Cursor Left	move left one character
cr	Cursor Right	move right one character
pc	Previous Char.	same as cl
nc	Next Character	same as cr

ql	cursor left	move cursor left one character (to next line if at end)
qr	cursor right	move cursor right one character (to next line if at end)
cu	Cursor Up	move up one line
cd	Cursor Down	move down one line
ll	Linear Left	move to left (inc. dead space)
lr	Linear Right	move to right (inc. dead space)
lu	Linear Up	move up one line (stay in column)
ld	Linear Down	move down one line (stay in col.)
pw	Previous Word	move to beginning of previous word
nw	Next Word	move to beginning of next word
pt	Previous Tab	move to prev. tab column on line
nt	Next Tab	move to next tab column on line
el	Express Left	to beg. of line, then straight up
xm	eXpress Middle	to middle character on line
er	Express Right	to end of line, beg. of next, end
lb	Line Beginning	move to beg. of current line
le	Line End	move to end of current line
pl	Previous Line	move to beg. of previous line
nl	Next Line	move to beg. of next line
ps	Previous Sentence	move to beg. of previous sentence
ns	Next Sentence	move to beg. of next sentence
pp	Previous Paragraph	move to beg. of previous paragraph
np	Next Paragraph	move to beg. of next paragraph
hm	HoMe (of screen)	move to first character on screen
bs	Bottom of Screen	move to last character on screen
mu	Move Up	scroll up one line
md	Move Down	scroll down one line
pu	Page (screen) Up	scroll up one screen
pd	Page (screen) Down	scroll down one screen
vu	scroll Up	scroll up one screen
vd	scroll Down	scroll down one screen
pf	Previous Form	move to top of previous page
nf	Next Form	move to top of next page
tf	Top of File	move to beginning of file
bf	Bottom of File	move to end of file

DEAD ACCENTS

s1	acute accent	insert temporary dead acute accent
s2	grave accent	insert temporary dead grave accent
s3	umlaut	insert temporary dead umlaut
s4	circumflex	insert temporary dead circumflex
s5	° accent	insert temporary dead °
s6	tilde	insert temporary dead tilde

DEFINING

dc	Define Column	begin column define
df	Define Free-form	begin/set free-form defining
dm	Define Modify	extend (or shrink) a block of selected text to cursor position (only with persistent selection)
dn	Delete, No undelete	delete selected text without saving it on delete stack
dw	Define Word	define current word
dl	Define Line	define current line
ds	Define Sentence	define current sentence
dp	Define Paragraph	define current paragraph
dz	Define end	end selecting a block if selection is in progress.
nb	uNbreakable Block	designate selected block of text as unbreakable
yd	"Yank" Define	clear define
xd	"X" Define	clear define, don't close window
db	Define Begin	move cursor to start of defined block
de	Define End	move cursor to end of defined block
dd	Delete block/char	end selecting block and delete block. If no selection, delete character
dz	Define end	end selecting a block if selection is in progress.

DELETING

rd	Rub out Defined block	delete defined block
nu	delete, No Undelete	delete selected text, don't save for possible later undelete.
nu	delete, No Undelete	delete selected text, don't save for possible later undelete.
bd	Backspace Delete	delete previous character
rc	Rub out Character	delete current character
rw	Rub out Word	delete current word
re	Rub out to line End	delete to end of current line
rl	Rub out Line	delete entire current line
rs	Rub out Sentence	delete current sentence
rp	Rub out Paragraph	delete current paragraph
ud	Un-Delete	restore text from undelete buffer
up	Un-Pad spaces	delete space(s) to left of cursor (up to next character)
rb	Rub out word Before	delete the word before the word the cursor is on

DOCUMENT DISPLAY MODES

nm	No Markers	conceal markers
ef	Edit Footnote/ code	open note or delta cursor is on
xp	eXPanded mode	change to Show Codes View
wg	normal mode	change to old Page-Line mode
sp	Show Pg-Ln	turn on Page-Line counter
tp	Toggle Pg-Ln	toggle Page Layout—old Page-Line mode
cm	Change Mode	toggle old Page-Line mode—Show Codes View
mk	MarKers	toggle display of format markers and line ending markers.
wz	Page Layout View	change to Page Layout View

MATH

dt	Dump Total	dump total at cursor in text area
mt.*	Multiply	multiply (*) accumulated sum by selected number
mt,/	divide	divide (/) accumulated sum by selected number
sm	SuM	add number (or defined numbers)
su	SUbtract	subtract number (or def. numbers)

MENU/HELP/SYSTEM

h@		open NB Help

PHRASE LIBRARIES

ad	Append Def. to key	append define to end of phrase key
sv	SaVe def. to key	save defined block on phrase key
@A	@Z	insert phrase from alphabet key
@0	@9	insert phrase from numeric key
&A	&Z	insert “ampersand phrase”
&0	&9	insert “ampersand phrase”
sg x or #	get Save/Get	insert text or run program from phrase key x or #

PRINT MODES

m0	Mode 0	select context mode
m1	Mode 1	select normal mode
m2	Mode 2	select bold mode
m3	Mode 3	select underline mode
m5	Mode 5	select bold-underline mode
m7	Mode 7	select superscrlpt mode
m8	Mode 8	select subscript mode
m9	Mode 9	select italic mode
mx	Mode conteXt	type in mode at cursor - same as M0, but does not get inserted in programs
mz	Mode bold italic	type text in, or make selected text, bold italic

SEARCHING

fd	File Difference	find next difference in two files
fm	File Match	find next similarity in two files
wa	WildAlphanumeric	any letter or number
wl	Wild Letter	any letter
wn	Wild Number	any number
ws	Wild Separator	any separator
ww	Wild "Within"	any intervening string (up to 80 characters long)
wx	Wild X	any character (including space)
nn x	generic wild card	generic Wild Card - the next character is the wild card. [see Chapter 8]

SYSTEM & MISCELLANEOUS

bk	BreaK	stop command or user program
ex	EXit Nota Bene	exit program
co	COmma	insert comma in key definition
ni	No Interrupt	suppress non-Nota Bene effect
no	No Operation	precedes word assigned to key
es	EScape	release selected text or close command window
fd	File Difference	find next difference in two files
fm	File Match	find next similarity in two files
xn	Transpose teXt	transpose text [see Chapter 8]
1/2/3/4/5/6		
sa	Save	save file
sl	Save aLL	save all open files in all windows.
it	Insert Tab	insert a tab on command line or in text
<<	copyright/«	enter ® in program or opening command brackets on command line
>>	high line or »	enter ¯ in program or closing command brackets on command line markers
nm	No Markers	hide format markers and line ending markers
bl	Balanced Left	jump to left edge of current balanced pair of command brackets
br	Balanced Right	jump to right edge of current balanced pair of command brackets
fc	Force Centre	insert centre text command
fl	Flush Left	insert flush left command
fr	Flush Right	insert flush right command
ff	Force reFresh	refresh screen
dx	Display X (off)	freeze display (pair with do)
do	Display On	turn display on (pair with dx, follow with ff)

The **dx** and **do** functions must be used in pairs in key assignments or in programs. If you turn off the display by using **dx** without later turning it back on by using **do** in the same key assignment, it will appear as if the computer has locked up. If you mistakenly create such an assignment or program, you can restore the screen by pressing F9, typing func do (thouqh you won't be able to see it being typed), and then pressing F10.

TOGGLING KEYBOARD MODES

ci	Clear Insert	switch to Overstrike mode (from Insert)
mi	toggle Insert	switch from Overstrike to Insert mode until a cursor key is pressed
si	Set Insert	switch to Insert mode (from Overstrike)
ti	Toggle Insert	toggle between insert and overstrike modes
tw	Toggle Word	toggle between Insert mode and Word Overstrike mode
tg	ToGgle views	toggle between Page Layout and the view previously displayed
ts	Toggle program recording mode	toggle program recording mode
ss	turn on recording mode	turn on Program Recording Mode

WINDOWS

as	Alternate Screen	toggle between two windows
nx	NeXt window	cycle through windows
cb	Cycle Backwards	move through windows in the reverse order to that in which they were opened
#1	#9	move cursor to window #
ef	Edit Footnote	open or close window (e.g., footnote window)
mw	Microsoft Windows functions	see Alphabetical List below]

SPELLING CHECKER/THESAURUS

ac	Auto-Correct	toggle Auto-Correct mode
az	Auto-Replace	toggle Auto-Replace mode
fs	Fix Spelling	go to last questioned word
ir	Insert Replacement	insert replacement word in personal dictionary
so	Spell One	check spelling of a word
sy	SYnonyms	display list of synonyms

REDLINING/BLUE-PENCILLING

ro	Redlining On	toggle Redlining/Blue-Penciling

ALPHABETICAL LIST OF FUNCTIONS

Name		Description
<<	**copyright or «**	Enters ® in program or « on command line
>>	**high line or »**	Enters ¯ in program or » on command line
#1 — #9		move cursor to window #
@0—@9		insert phrase from numeric key
@A —@Z		insert phrase from alphabet key
&A—&Z		insert "ampersand phrase"
&0—&9		insert "ampersand phrase"
ac	**Auto-Check**	turn Auto-check on and off.
ad	**Append Def. to key**	append define to end of phrase key
ar	**Auto-Replace**	execute Expand Abbreviation (NB: *toggle* Expand Abbreviation is AZ)
as	**Alternate Screen**	toggle between two windows
az		Toggle Auto-Replace on and off.
bc	**Blank Command**	move to beg. of command line
bd	**Backspace Delete**	delete previous character
bf	**Bottom of File**	move to end of file
bk	**BreaK**	stop command or user program
bl	**Balanced Left**	Jump to left edge of current balanced pair of command brackets
br	**Balanced Right**	Jump to right edge of current balanced pair of command brackets
bs	**Bottom of Screen**	move to last character on screen
bx	**Blind eXecute**	execute command without putting it on command line
c1-c0	**Counters 1-9 & 0**	insert counter («C#») in text
cb	**Cycle Backwards**	Move through windows in the reverse order to that in which they were opened.
cc	**Change Cursor**	move between command line & text
cd	**Cursor Down**	move down one line
ch	**Clear Header**	clear command line w/o moving cursor
ci	**Clear Insert**	Switch to Overstrike mode (from Insert)
cl	**Cursor Left**	move left one character
cm	**Change Mode**	toggle old draft mode - Show Codes View
co	**COmma**	insert comma in key definition

cp	**CoPy define**	copy block to cursor position
cr	**Cursor Right**	move right one character
cu	**Cursor Up**	move up one line
db	**Define Begin**	move cursor to start of defined block
dc	**Define Column**	begin column define
dd	**Delete block/char**	end selecting block and delete block. If no selection, delete character
de	**Define End**	move cursor to end of defined block
df	**Define Free-form**	begin/set free-form defining
dl	**Define Line**	define current line
dm	**Define Modify**	Extend (or shrink) a block of selected text to cursor position. (Only with persistent selection on)
dn	**Delete, No undelete**	Delete selected text without saving it on the delete stack
do	**Display On**	turn display on (pair with **dx**)
dp	**Define Paragraph**	define current paragraph
ds	**Define Sentence**	define current sentence
dt	**Dump Total**	dump total at cursor in text area
dw	**Define Word**	define current word
dx	**Display X (off)**	freeze display (pair with **do**)
dz	**Define end**	end selecting a block if selection is in progress.
ec	**End Cell**	move cursor to end of current cell
ed	**Entry Define**	define current row of cells
ef	**Edit Footnote/code**	open or close window (e.g., footnote window)
el	**Express Left**	to beg. of line, then straight up
er	**Express Right**	to end of line, beg. of next, end
es	**EScape**	release selected text or close cmd window
ex	**EXit Nota Bene**	exit program
fc	**Force Centre**	centre text
fd	**File Difference**	find next difference in two files
ff	**Force reFresh**	refresh screen
fl	**Flush Left**	insert flush left command
fm	**File Match**	find next similarity in two files
fr	**Flush Right**	insert flush right command
gh	**Go to Header**	move cursor to command line
gt	**Go to Text**	move cursor to text area
h@	**Help**	open NB Help
hm	**HoMe (of screen)**	move to first character on screen

ir	**Insert Replacement**	Open auto-check/auto-replace pair dialog
it	**Insert Tab**	insert a tab on command line or in text
lb	**Line Beginning**	move to beg. of current line
ld	**Linear Down**	move down one line (stay in col.)
le	**Line End**	move to end of current line
ll	**Linear Left**	move to left (inc. dead space)
lr	**Linear Right**	move to right (inc. dead space)
lu	**Linear Up**	move up one line (stay in column)
m0	**Mode 0**	type text in context mode
m1	**Mode 1**	type text in normal mode
m2	**Mode 2**	type text in **bold** mode
m3	**Mode 3**	type text in underline mode
m5	**Mode 5**	type text in **bold-underline** mode
m7	**Mode 7**	type text in superscrlpt mode
m8	**Mode 8**	type text in subscript mode
m9	**Mode 9**	type text in *italic* mode
mc	**Mark Column**	select cell at cursor location in a table
md	**Move Down**	scroll down one line
mi	**toggle Insert**	switch from Overstrike to Insert mode until a cursor key is pressed.
mk	**MarKers**	toggle display of format markers and line ending markers.
mu	**Move Up**	scroll text and cursor up one line
mv	**MoVe define**	move selected block to cursor position
mw	**Microsoft Windows functions**	Microsoft Windows functions (do 'func mw', then enter 2-letter code):
	ac	Cascade all text windows
	ah	Split all text windows horizontally
	ar	Tile all text windows
	av	Split all text windows vertically
	cb	Display contents of Windows Clipboard
	cl	Close text window
	cp	Copy selected text to Windows Clipboard
	cu	Cut to Windows Clipboard
	hh	Display help on using Help files (Windows Help)
	hi	Display Help Index (Nota Bene Help)
	mn	Minimize NB screen
	pa	Past text from Windows Clipboard
	mv	Display 4-headed arrow to move NB screen (minimizes NB at top lhs of screen; dragging enlarges it)
	mw	Move window
	mx	Maximize NB screen
	pa	Paste text from Windows Clipboard

pl		Paste link (doesn't seem to do anything)
pr		Display information about Windows printer driver
ps		Paste special
qu		Quit
rm		Restore text window to maximum size
rs		Restore NB screen to previous non-max|min size
rw		Restore file
sf		Repaint the screen (doesn't seem to do anything)
sl		Scroll left
sr		Scroll right
sw		Size document window
sz		Display 4-headed arrow to move text window
wf		Make current text window full screen
wi		Minimize text window
mx	**Mode conteXt**	type in mode at cursor - same as M0, but does not get inserted in programs.
mz	**Mode bold italic**	type text in bold italic, or make selected text bold italic
nb	**uNbreakable Block**	designate selected block of text as unbreakable
nc	**Next Character**	cursor right (same as cr)
nf	**Next Form**	move to top of next page
ni	**No Interrupt**	suppress non-Nota Bene effect
nl	**Next Line**	move to beg. of next line
nm	**No Markers**	no markers - hide format markers and line ending markers
nn x	**generic wild card**	generic wild card (see Chapter 8)
no	**No Operation**	precedes word assigned to key
np	**Next Paragraph**	move to beg. of next paragraph
ns	**Next Sentence**	move to beg. of next sentence
nt	**Next Tab**	move to next tab column on line
nu	**delete, No Undelete**	delete selected text, without saving it for possible later undelete.
nw	**Next Word**	move to beginning of next word
nx	**NeXt window**	cycle through windows
pc	**Previous Char.**	same as cl
pd	**Page (screen) Dn**	scroll down one screen
pf	**Previous Form**	move to top of previous page
pl	**Previous Line**	move to beg. of previous line
pp	**Previous Paragraph**	move to beg. of previous paragraph
ps	**Previous Sentence**	move to beg. of previous sentence
pt	**Previous Tab**	move to prev. tab column on line
pu	**Page (screen) Up**	scroll up one screen
pw	**Previous Word**	move to beginning of previous word

q2	**execute cmd**	finish command started with BX
ql	**cursor Left**	move cursor left one space (to previous line if at beginning)
qr	**cursor Right**	move cursor right one character (to next line if at end)
r0-r9	**Record ASCII 0-9**	input ASCII digits
rb	**Rub word Before**	delete word before word cursor is on.
rc	**Rub out Character**	delete current character
rd	**Rub out Defined block**	delete defined block
re	**Rub out to line End**	delete to end of current line
rl	**Rub out Line**	delete entire current line
ro	**Redlining On**	toggle Redlining/Blue-Penciling
rp	**Rub out Paragraph**	delete current paragraph
rs	**Rub out Sentence**	delete current sentence
rw	**Rub out Word**	delete current word
s-	**Show last cmd**	displays last command on command line
s1	**acute accent**	insert temporary dead acute accent
s2	**grave accent**	insert temporary dead grave accent
s3	**umlaut**	insert temporary dead umlaut
s4	**circumflex**	insert temporary dead
s5	**° accent**	insert temporary dead °
s6	**tilde**	insert temporary dead tilde
sa	**Save**	save file
sg x or #	**get Save/Get**	insert text or run program from phrase key x or #
si	**Set Insert**	switch to Insert mode (from Overstrike)
sl	**Save aLL**	save all open files in all windows.
sm	**SuM**	add number (or defined numbers)
so	**Spell One**	check spelling of a word
sp	**Show Pg-Ln**	Switch to old Page-Line view
ss	**turn on recording mode**	turn on Program Recording Mode
su	**SUbtract**	subtract number (or def. numbers)
sv	**SaVe def. to key**	save defined block on phrase key
sy	**SYnonyms**	display list of synonyms
tf	**Top of File**	move to beginning of file
tg	**ToGgle views**	toggle between Page Layout and view previously displayed
ti	**Toggle Insert**	toggle insert - Replace modes
tl	**Table Left**	move cursor to previous cell

tp	**Toggle Page layout-**	toggle Page Layout-old draft mode
tr	**Table Right**	move cursor to next cell
ts	**Toggle program mode**	toggle program recording mode
tw	**Toggle Word**	switch between Insert mode and Word Overstrike mode.
ud	**Un-Delete**	restore last text deleted
un	**paste from clipboard**	Paste copy from clipboard. (In NB Lingua, in Page layout View,an «XAEnglish» code is inserted as well as the text.)
up	**Un-Pad spaces**	delete space(s) to left of cursor
vd	**scroll Down**	scroll down one screen
vu	**scroll Up**	scroll up one screen
wa	**Wild Alphanumeric**	any letter or number
wg	**normal mode**	change to old draft mode
wl	**Wild Letter**	any letter
wn	**Wild Number**	any number
ws	**Wild Separator**	any separator
ww	**Wild "Within"**	any intervening string up to 80 chars.
wx	**Wild X**	any character (including space)
wz	**page layout view**	change to Page Layout View
xc	**eXecute Command**	execute command on command line
xd	**"X" Define**	clear (don't delete) define
xm	**Xpress Middle**	to middle character on line
xn	**Transpose teXt**	transpose text (see Chapter 8)
1/2/3/ 4/5/6		
xp	**eXPanded mode**	go to Show Codes View
yd	**"Yank" Define**	clear define, don't delete or close window

Programming: Introduction

In addition to **Nota Bene**'s many editing and formatting commands, a versatile programming language (called XPL) is available for enhancing or modifying the program according to your needs. It is likely that many users are unaware of the existence of this feature of **Nota Bene**; and that others, after a brief glance at this chapter, will think that it all looks very forbidding, and conclude that it is not for them. That is a pity, because learning the programming language is quite like learning a new foreign language, with the advantage that its vocabulary is far smaller; furthermore its rules of syntax are fewer, and do not bring with them all the exceptions that have to be memorised when learning a new foreign language. It is also unfortunate that books introducing beginners to a computer language often start off showing what you can do in the language by presenting you with specimen programs that will write for you on the screen something as useful as "Hello! My name is ....." ─ on a level with the "Où est la plume de ma tante?" of elementary French. That is not necessary, and will be avoided here. Instead it will be shown that even rudimentary programs can save you time or trouble in performing tasks in **Nota Bene**; and that from there you can gradually move on to more sophisticated operations.

It should be mentioned here that the programming language used for writing programs in **Nota Bene** is identical with much of the programming language used by **Nota Bene** itself, so that a beginner in **Nota Bene** programming has more familiarity with the language than he/she supposes. If you have not been talking XPL all your life, you have been nearer than you may think.

The programs you create with **Nota Bene** are regular files composed of three different kinds of instructions, used individually or in combination:

i. normal text
ii. program functions (similar to the functions in the keyboard table─see "Keyboard Customization" chapter)
iii. special programming codes, or "calls"

Text A program that contains the first of these is the simplest kind. Text is inserted from a program file, rather than by being entered from the keyboard. This can be useful for inserting boilerplate text, and does not keep any memory tied up when not being used, as does storing text on phrase keys or in an abbreviation dictionary.

Unlike phrase libraries and keyboard assignments, both of which insert text instantly, a program containing text is "played back" sequentially ─ characters appear one after another, as if being typed by a fast typist.

Program Functions Programs that contain "program functions" are a little more complicated. Like keyboard assignments, such programs can perform editing **actions**--they are not limited to inserting text or copying existing format codes. Program functions are embedded in ordinary files on disk. Unlike programs assigned to keys in the keyboard table, which must be loaded into memory, programs recorded as program files do not need to be loaded into memory until you want to use them. Also, storing routine editing functions in programs

enables you to "run" a program as required without having to create and load different keyboards for particular applications. For further details see **Program Functions** below.
Program "Calls" The most complex, versatile, and powerful programs, however, contain a series of programming operations that will be referred to as "program calls". These program calls allow you to compare characters and numbers, check for error conditions, or perform conditional actions, among other things. Full use of this programming language enables you to get **Nota Bene** to do many things that it cannot already do. For further details see **Program Calls** below, and the "Program Calls" chapter.

Program File Commands

The following commands are used for creating, editing, and implementing programs:

Creating New Program To create a new program file:

F9 ne x:filename.**run F10**

recommended extension

Choosing a standard extension like .RUN for your program files makes it easier to get a directory that lists only program files, and helps to avoid treating them like regular files. But you can call programs anything you like. If you want to be able to run them from the command line, use no more than 8 characters for the file name, and no more than 3 for the extension.

Calling Program to Screen To call an existing program file to the screen so you can check its content or edit it:

F9 ca x:filename.**run F10**

Running Program To implement (or "run") a program:

F9 run x:filename.run **F10**

This command causes the program to be executed. Depending on the content of the program file, it will insert text either on the command line or in the text area, and/or it will execute program functions, and/or it will cause program calls to be evaluated and implemented.

If no filename is specified, the last program that was run will be rerun.

If the program file has already been loaded into memory (see next section), you do not need to specify the drive (and/or path) when running it.

If the program file has been loaded onto a phrase key, you can implement the program just by using that key (see next section).

Normally, program files are read from disk each time they are run. If you want faster response, you can load the program file(s) into memory, as explained in the next section. *[Probably superfluous on modern computers, and with Win XP.]*

Loading Programs into Memory

Loading on Phrase Key A program can be loaded on an **Alt** phrase key:

substituting for 'x' the letter or digit of the phrase key.

The program can then be implemented at any time by holding **Alt** down and pressing the alphanumeric key.

Remember not to assign a different phrase to the same key unless you want to overwrite the program in memory.

Even when a program is assigned to a phrase key, text is entered into the file sequentially, as explained earlier.

You can use the **salib** command to create a phrase library that includes programs assigned to phrase keys. Thereafter, **ldlib** will reload the programs even if the original program files are not on the disk.

Loading on "Ampersand Phrase" If you need to load more programs, use ampersand phrases:

The program can then be implemented
(i) by using the function (**func**) command with the corresponding phrase: **func &x**;
or
(ii) by mapping the ampersand phrase to a key in your keyboard file with:
NN=&x ;where NN is the number of the key in the table.

Programs loaded on ampersand phrases, unlike those loaded on **Alt** phrase keys, cannot be saved to disk. They are saved only to memory, and are lost when you quit **Nota Bene**.

Loading in General Memory A program can be loaded into general memory:
F9 ldpm x:filename.run **F10**
The program can then be implemented at any time by using the **run** command with the program's filename. **Nota Bene** first looks in general memory for the indicated program; if the program file is not there, it checks the default disk.

Loading via NBSTART.INT NBSTART.INT is a program file that is run every time you start Nota Bene. In vanilla NB it is empty. You can add commands to it, e.g.:

BX run x:filename.run **Q2**
BX load x:filename.run,a **Q2** *[See p 165 for BX and Q2.]*

Other Ways of Running Programs For users with sufficient experience of **Nota Bene** there are other ways of loading and of running programs, which will be only mentioned here. They can be saved on extended phrases and then run with a single «pv#» command; or they can be added to the bottom of the XYWWWEB.U2 file, if you have it loaded - see http://www.serve.com/xywwweb/, and p 163.

Removing Program(s) If you want to free up memory by clearing a program that is no longer needed, use the **remove** command:
remove x for program on **Alt** phrase (A-Z, 1-9)
remove filename for specific program only

Normal Text in Program Files

Text is typed into a program file just as it is in an ordinary file. You can use the normal editing keys or commands to move the cursor, delete, or otherwise edit the text. When any of the available editing operations are used (such as moving the cursor to the beginning of the line), the operation is implemented as usual.

Although such text can also be typed while the "program recording mode" is on (as described in a later section), it is not necessary to have that mode on for typing text. In fact, having the recording mode on makes it more difficult to edit the text, because pressing an editing key embeds a program function into the document rather than actually implementing the editing operation. See **Program-Recording Mode** below.

If a file is to consist solely of text, there is, in fact, little point in making it a program file. It can be a regular text, e.g., boilerplate, file, which can be inserted into the body of another file by use of the **merge** command. But, if you have a file that already is a program file, and wish to add text to it at a certain place, you can do so, by using **ca** to call the file and then entering the new text in the desired place.

Program Functions

"Program functions" and the "keyboard functions" used in keyboard tables are the same things used in different ways. Keyboard functions are two-character codes, and program functions are three-character codes (although they look to be two-character followed by a space); and all instruct **Nota Bene** to perform a particular editing or other operation.

Keyboard Functions

As explained in the "Keyboard Customization" chapter, a keyboard function is a two-letter code that is assigned (by itself or in combination with other keyboard functions or characters) to a particular key in the keyboard table to determine what that key does. When you press the key, the keyboard function is implemented.

Keyboard functions are typed as ordinary text in the keyboard table, which must then be stored and loaded into memory before the function can actually be used. For example, the F9 key is defined in the keyboard table as

```
68=xc
```

so that, when you press F10, the 'xc'functions is implemented: the command on the **C**ommand line is executed.
Alternatively, keyboard functions can be implemented from the command line with the **func** command. For example, **F9** func dl **F10** will define the current line.

Keyboard functions are generally lowercased in the keyboard table; but the case makes no difference. In this manual, they are always shown in lowercase (and in bold [as "**xc**"] in explanatory text).

Program Functions

A program function is a function that is entered into a program file to be performed as part of the program. Program functions implement exactly the same operations as the corresponding keyboard functions, but do so automatically as part of a program rather than when a key is pressed. If a program contains the program function **BC** , the cursor is automatically moved to the command line, and the command line is automatically blanked.

If the program-recording mode is on (see next section), program functions can be entered into the file by pressing the key to which they are assigned. For example, striking the key **F9**, the function of which is to move the cursor to the command line and clear anything that is there, will not actually perform that function, but will instead enter the codes **&X BC** into the file. If program-recording mode is not in effect when **F9** is struck, it will not enter the codes into the file; instead it will actually execute its function of clearing the command line.

A program function consists of the same two letters as the corresponding keyboard function, followed by what looks like an ordinary space but is really a special null code. These three characters are in fact a single unit: if you put the cursor on the first character, say the **B** of **BC** , and then strike the cursor-right key once, the cursor will jump three places to the right. The program function is also defined and copied or moved as a single unit.

Program functions always appear uppercased. In Show Codes View they appear as white letters in a black rectangle. They cannot be typed as actual characters.

In this manual, when a program function is referred to by itself in explanatory text, the null character is not normally shown because an extra space looks unnatural in the text.

Searching for program functions: To search for program functions, strike the pfunc key (**Ctrl+;** or **Ctrl+Keypad 5**) twice, then type the two letters of code you're looking for and strike F10.

Recording Program Functions: Because program functions are not ordinary text, they cannot be directly typed into a program file. Instead, they must be embedded, either
(i) in program-recording mode, by pressing the key to which they are assigned (described below).
(ii) By using the **pfunc** command, which converts a mnemonic into its corresponding function (see below).

Program-Recording Mode This mode operates differently in **Nota Bene** from the the macro-recording modes of other word processing programs. In those programs, when the mode is on, entering keystrokes does two things:

(i) it makes the keystrokes function in their ordinary way, e.g., to delete or to backdelete a character, to enter a character in uppercase rather than lowercase, to execute a command, etc.; at the same time

(ii) it also records the character's editing function, so that, when a string of such functions has been recorded and saved, what is often called a macro has been created, which can be used on subsequent occasions to perform the operation that was on this occasion performed in (i) by the successive keystroke entries.

If **Nota Bene** were of this kind, striking in succession (with no spaces between them) the keys **F9** d i r <space> ***** . d o c **F10** would

(i) bring up a directory of all filenames in the current subdirectory that have the extension 'doc';

(ii) record the command

BC dir *****.doc**XC** (which could be used to do that job in future)

Nota Bene's recording mode is simply and solely a recording mode. It does not perform function (i), just function (ii). This makes the editing and correcting of recorded strings much easier. Because **Nota Bene** *records* the keystrokes you make, but does not simultaneously *execute* them, you can correct your typing errors there and then; you do not have to go back to square one and start all over again. But you *must* first toggle off the Recording mode; otherwise, instead of, say, deleting unwanted characters, you will be embedding unwanted 'delete' commands into the program you are trying to record.

Program-recording mode records your keystrokes in the file rather than using them to edit the program file. This makes it very easy to write programs that incorporate editing or other operations equivalent to those already assigned to keys: while writing the program, you can continue thinking in the way you do when you use **Nota Bene** itself. Whenever you want to record a particular editing or other operation as part of the program, simply turn on the program recording mode. Then, when you want to return to moving around in or editing the program file itself, turn off the program-recording mode.

Every time you turn off Recording Mode, a **NI** code is entered into the program at the cursor position. You can safely delete it - just press backdelete *once*.

Ctrl Alt; [semicolon] toggles program-recording mode on/off

So do **Ctrl Alt+keypad 5, Ctrl+Shift+Alt+;**, and **Ctrl+Shift+Alt++keypad 5**.

If you decide upon one, you can delete the others from your keyboard table, leaving 3 keys free for user customizing.

Editing Remember that to perform operations,—cursor movement, deletion, copying, etc.,—upon a program file rather than recording them as components of the file, you need to turn off the program-recording mode. Otherwise, for instance: if you make a mistake when recording and want to delete the previous character, striking **Bkdel** will insert the string on that key rather than deleting the character.

Inserted Material Whenever a key is pressed while the program-recording mode is on, everything assigned to that key in the currently loaded keyboard table is inserted into the file: a single function, a series of functions, individual characters, etc.

If you press a key to which a program has been assigned, the entire program will be embedded.

In any case, it is inadvisable to use Recording mode, because in Nota Bene 8 many keys which used to be defined as single functions are defined with strings that cannot be used in programs. E.g., **Bkdel** is defined as '**[U &X** BD**U]** '. You can enter this into your program in program-recording mode, but it will not backdelete. You will need to use 'Pfunc bd', which will insert a **BD** code into the program. That will work.

PFUNC Command The put-function (**pfunc**) command allows you to type the two-letter mnemonic for a function and have the actual function embedded in a file at the cursor position:

┌─two-letter mnemonic (e.g., bc)
F9 pfunc xx **F10**

Example: Pressing **F9 pfunc bc F10** embeds the function **BC** into a file.

Pfunc is on **Ctrl+;** and **Ctrl+Keypad 5**.

A more detailed discussion of the two methods of embedding function codes (Recording Mode and the PFUNC Command) can be found in the chapter on Writing XPL Programs.

Sample Program: Text & Program Functions

You can use **Nota Bene**'s programming language to automate your work even without learning the more sophisticated program calls. Here are two simple examples:

(a) move screen up 10 lines

You may sometimes, when writing a file, find that you are always writing at the bottom of the screen, and would like to have the screen moved up by ten or a dozen lines, so that you can continue working, but now with the cursor starting in the middle of the screen. You can achieve that by striking **Ctrl-** , say, 10 times. But, if you put this program on a key, the job would be done much quicker:

MU MU MU MU MU MU MU MU MU MU

You can create that program by
(i) creating a file (with **ne**),
(ii) toggling Recording mode on,
(iii) striking **Ctrl-** 10 times,
(iv) toggling Recording mode off, and
(v) saving the file to disk.

(b) copy from 1 window to adjacent one.

Suppose you have two documents open in adjacent windows and want to selectively copy paragraphs from one to the other. Although you could do this with only a few keystrokes in any case, you would like to make the process simpler yet. The following program would do the trick:

DP AS BF CP AS XD

This sequence defines the current paragraph (**DP**), switches windows (**AS**), moves to the bottom of the file (**BF**), copies the paragraph (**CP**), switches back to the first window (**AS**), and clears the define (**XD**).

If you want to add a dividing line between the old material in the receiving file and the new, add

[CR] [i.e., strike the Enter key] **BC** ld **-XC [CR]** immediately after the **BF** program function:

DP AS BF
BC ld **-XC**
CP AS XD

(You put a carriage return before and after the leader in order to have it on a line by itself.)

You can have a message inserted on the command line to tell you what was done. To do so, add the following program functions and text after the **XD** in the example above:

BC Moved paragraph to other window**GT**

The prompt (**pr** message) command can be used within programs to insert short messages on the status line rather than on the command line. Examples are given in the sections later in this chapter.

You could save either of those programs on a phrase key by using **ldpm** x:filename,x. (Use **salib** if you want to save the assignment permanently.)

Although the above key sequence could be assigned to a key in the keyboard table, you will probably find it more useful to create task-specific programs rather than constantly modifying the keyboard table. This is especially the case if your programs become long.

Program Calls

Macros Programs made possible by the use of Program Functions are simple, straightforwrd, and limited in their scope. They are what are normally called ‘macros’ in other software. A macro is a simple ‘batch’, i.e., a flow of uni-directional commands which get executed in sequence, one after the other. They do not allow you to insert conditions or ‘if’s, and they do not allow the insertion of options. It is like having a special key for a set of tedious operations; e.g., in the previous example: ‘define this paragraph, go to the file in the alternative screen, copy the paragraph, insert one blank line, return to the source file, cancel define’.

Program Calls The second, and far more powerful, versatile, and flexible type of XPL program is that which makes use of Program Calls as well as Program Functions. They make it possible to implement a wide variety of special operations: updating calculations, performing automated searches, or customizing the program in many other different ways.

These operations can be performed because the program calls let you do such things as:

conduct string and numerical comparisons and execute other logical operations
perform conditional actions (based on results of these logical operations or the occurrence of error conditions)
designate and jump to specified labels
check the status of variables such as available memory or current format settings
read and evaluate characters typed from the keyboard

Most program calls are Embedded Commands (codes) that—unlike ordinary embedded commands—take effect only in files "played back" with the **run** command. Embedded commands cannot be seen in Page Layout View. Therefore, a program should always be read (and written) in Show Codes View, where the contents of the codes become visible. They are what make XPL stand apart from simple macro programs, however detailed those may be. Through program calls, XPL enables you to introduce conditions and options into programming, much as you can in other high level languages. It is done
(a) by storing information in memory buffers and later using it; and
(b) by (i) looping to repeat part of a program,
(ii) jumping to a marker or label,
(iii) branching in response to an 'if' clause,
(iv) entering characters from the keyboard during the running of a program, and
(v) exiting from part of a program, or from the whole of it.

The storing of information is done on phrase keys, either the regular phrase keys (A-Z, 0-9) or extended phrases (00-99, 000-099, 100-999 and 1000-1999). In general it is best to avoid the regular phrase keys, which in any case you may have committed to other uses; anyway, there are far more extended phrases available than you are ever likely to need.
The difference between the 00-99 and 000-099 ranges on the one hand and the 100-999 range on the other concerns their survival in memory. Anything stored to phrases in the **00-99** and **000-099** ranges is deleted from memory as soon as you leave the particular program in which they are used; that has the advantage that you can use the same phrase-numbers over again in programs run later in a single working session. Phrases stored in the **100-999** range remain in memory throughout a working session, or until they are replaced; that is useful for storing what are called sub-routines.

Although you have that enormous range of phrase-numbers to choose from, you need to exercise some care in doing it, in particular avoiding those that **Nota Bene** itself uses in its own programming. Of the lower ranges 00 is reserved for a special function (storing all or part of the contents of the command line), but the remainder (01-99 and 000-099) can be used freely. Of the upper range 100-999 are currently available, and are not used by **Nota Bene**; **Nota Bene** uses phrases in the 1300, 1700, and 1900 blocks, so you should avoid those.
(The range reserved for XYWWWEB.U2, if you use it, is 600-799.)

va @# You can always check for a given number by running on the command line the command **va @#** (substituting for # the phrase-number that you want to find out about); you must have a file on screen for this command to work. If there is something stored in that memory location, it will be displayed on the screen (if you are in Page Layout View); you can delete the string and the code preceding it with a stroke of the <**BkDel**> key. If all you get is a plain code, then there is nothing stored there, and the phrase-number can be used.

The discussion of "program calls" in the remainder of this chapter contains information that will come more readily to those who have had some prior programming experience; but that is certainly not an absolute requirement. Nota Bene does not provide support for users wishing to implement programs using these extended features. The following documentation, however, is intended to provide sufficient information for those wishing further to enhance the functionality of **Nota Bene**. Examples are provided so that even those without programming experience should be able to construct powerful and useful programs. In general it pays to try working out the sequence and the flow of a program either in your head or on paper, before actually starting to write it. More information about the use of program calls is given in Chapter 6 and Chapter 8.

Display Mode to Use In Page Layout View, program calls are typed on the command line and, after execution are invisible. Creating and editing program files in Show Codes View is much easier. That way the content of all of the program call codes can be seen (rather than just the one the cursor is on), making it much easier to find the proper label or follow the flow of the program. All examples will be shown in this mode.
Show Codes View is also better because you can type the program calls in lowercase and control the case of any variables to make the program easier to read. If executed in Normal Display mode, program calls and some variables would be automatically converted to uppercase.

A program without breaks can be hard to read, because it appears to be a continuous stream of functions and program calls, with no punctuation to break them up, or to display the logical relationship between clauses. You cannot break the lines of a program simply with paragraph markers, because they will either execute the preceding string or be entered into the text.

Instead, use this sequence:

;*; *[semi-colon star semi-colon]*

Any line that begins with this sequence is a comment and will not be executed.
Any line that ends with this sequence will be executed up to the beginning of the ;*; string.

So you can have a program broken up like this:

```
;*; Program [macro, really] to change to adjacent window
;*;
DP AS BF CP AS XD ;*;
;*;
```

```
;*; You can add a leader character after the BF
;*; written by X, on date Y.
```

Only the **DP AS BF CP AS XD** line will be executed. The **BF** on the penultimate line won't be, because of the ;*; at the beginning of the line.

When writing a program it is a good idea to insert comments, explanations, and descriptions (where they can be put in without affecting the operation of the program), because a program can often seem unfamiliar even to its author after a long period of not scrutinising it. The ;*; sequence can used to insert such explanatory or descriptive text, either at the beginning of a program file, or in the course of it; they can also be inserted after the end of the program.

In Show Codes View, command brackets are typed with **Ctrl+,** or **Ctrl+.,**. Remember that there must be a closing bracket for every opening bracket. Many program calls contain nested elements, so make sure you correctly create the pairs.

Definitions of Terms Related to Program Calls

String A **string** is alphanumeric text—a character, a word, or a phrase, including numbers, punctuation marks, separators, and box graphic characters.

Number A **number** is simply that—a numeric value that can be manipulated by the ordinary four-function math operations.

Variable A **variable** is a "place holder" that can be replaced with text or numbers that vary depending on the situation.

Expression An **expression** is an operation performed on strings or numbers whereby the components are "analyzed" or "evaluated," and the result saved for further use.

Operators There are three different (though related) kinds of **operations:** simple **mathematical** operations, **comparative** operations (for string or numerical comparisons), and **logical** operations (to determine truth or falsity of expressions).

Subroutine A **subroutine** is a section of a program that can be reused repeatedly or in different contexts within the program.

Pattern of Explanations The program calls and related elements are described in the following chapter. For each, the program call itself is shown on the left, with the format (in Show Codes View) to the right. Be sure to use commas and parentheses wherever they are indicated. Notes and short examples are given after the descriptions.

Programming: Program Calls

Saving to a Phrase

There are three different commands for saving information to phrases, **sv**, **sx**, and **su**. Each performs a different function from the others, as will be described in the next three sections.

Save Variable

sv «sv#,text or number to be saved»

saves a string of characters or a number (treated as a string of digits) to a regular phrase (a-z, 0-9), or to "extended" phrases [01-99, 000-799] provided within programs for later use. The string is saved exactly as it is, without any interpretation or evaluation. That means that the string is saved simply as text; if, for example, there are numbers in the string, they are saved as numerical characters. Once a string has been saved to a phrase, the **pv** command (see **Put Variable** below) can be used to insert it—again just as it is—at the cursor's position, as if it had been a defined block being copied to a new location.

> Note that 0;*; Program reports ASCII value of character under cursor (only if character is one of the ascii character set, other than ASCII 254 (see below)). If character under cursor is not an ASCII character, program reports that æSX command requires a number.'
> 0-9, 01-09, and 000-799 are all different.
>
> Embedded commands (such as format commands and program calls) and mathematical expressions are **not**, when saved with the **sv** call, "evaluated", but treated as regular text.

Examples:

«sv01,Y»	—saves "Y" as phrase 01
«sva,WEBER.DOC»	—saves "WEBER.DOC" as phrase "a"
«svA,WEBER.DOC»	("a" and "A" are same phrase)
«sv56,49.034»	—saves "49.034" as phrase 56
«sv57,49.034+1»	—saves "49.034+1" as phrase 57
«sv3,«IP5»»	—saves **ip5** delta as phrase 3
«sv83,**RC RC RC** »	—saves three rubout-characters (**delete**) functions as phrase 83

Note: the example in the fifth line saves the string as a string, and does not evaluate it, i.e., does not add the two numbers and save the sum as 50.034. But see what happens with **sx** (below): «sx56,«pv57»» does evaluate it, and «pv56» would then display 50.034.

In addition to the above use of **sv**, which is of the form «sv01,#», there are two other uses, each of a slightly different form: «sv#,» and «sv#». The first, in which there is nothing following the comma, saves nothing to the phrase specified. By doing that it clears from memory anything already saved to that phrase, and ensures that it will be available for use in the program in which the **sv** is embedded. For example, suppose the program is going to make use of phrase 799, where there might be something left from a previous program run during the current working session. «sv799,» removes anything that may be there. In addition, a program will not recognize a phrase as empty, unless it has been specifically emptied. If, for example, you want a program to treat the extended phrase 01 as being initially empty, it will not recognize it as that until you embed in it the call «sv01,», even although at the outset of the program there will be nothing stored at 01.

«sv#», where there is nothing following the #, not even the usual comma, has yet a different function. It is used in just one situation, viz., where the program in which it is embedded has just defined a block of text. «sv#» saves the defined block to the phrase specified, where it will be kept in memory. For example, a variation on the earlier example of defining a paragraph and moving it might run: **DP** «sv25»**YD**. That defines the current paragraph, saves it to phrase 25, and then clears the definition; the defined block remains stored in memory, and by use of the **pv** call (see **Inserting a Phrase**) can be reinserted into the text area of a file at any time during the remainder of the program.

Note: this function of «sv#» is not available in versions of **Nota Bene** before 3.1. In them it's possible to save defined blocks only to regular phrase keys (a-z, 0-9), and it's done by using the function code **SV**. The previous example would in earlier versions be: **SV** 1**YD**

Save eXpression ———————————————————————————

sx «sx#,expression» where # is a-z, 0-9, 01-99, or 000-799

"evaluates" or performs an operation on numbers, strings, and variables, and **stores result in specified phrase**; in addition, reads and "identifies" characters typed from the keyboard, cursor and column position, the number of windows open, the amount of free memory, or any other of many values (see **Values** below).

> If **sx** is executed on the command line when in Page Layout View, an input window opens for entry of the expression to be saved (as with **Ctrl F10** for opening a note window). But, as previously recommended, Show Codes View should always be used when writing a program.
>
> **Actual text is not allowed within an expression**. Strings must be referred to by using the **is** command—or by enclosing them in straight double quotation marks (see example below and **InSert phrase** later).
>
> Within expressions the plus operator concatenates strings (see example), whereas relational operators (e.g., "<=" and "==") compare strings according to their sort sequence; other operators are described in the "Mathematical & Logical Operators" section.

Examples: (based on previous **sv** examples)

«sx34,«pv56»+23»	─adds 23 to value of phrase 56 (49.034) and saves result (72.034) as phrase 34
«sx56,«pv56»+1»	─adds 1 to value of phrase 56 (49.034) and saves new value (50.034) as same phrase
«sx56,«pv57»»	─evaluates the value of phrase 57, and saves it (50.034) as phrase 56
«sx56,«pv57»+«pv34»»	─adds the values of phrases 57 and 34, saving the result to phrase 56
«sx10,«cp»»	─saves cursor position as phrase 10, i.e., registers the number of bytes/characters from the top of the file to the cursor's current location
«sx99,«va$wn»»	─saves window number (see **Values** below) as phrase 99
«sx5,«is01»+«isa»»	─saves as phrase 5 the strings saved on phrases 01 and a; result: YWEBER.DOC

Text in double straight quotation marks:
Actual text cannot be used within an **sx**, except within double straight quotation marks. So the following would be invalid:

«sx5,«is01»+ if file is +«isa»»

The old method of getting round this was to save "if file is" as a phrase, e.g., 2:

«sv2, if file is »

Then use:

«sx5,«is01»+«is2»+«isa»»

This would save as phrase 5 the sequence of strings on phrases 01, 2, and a:
Y if file is WEBER.DOC

But now you can simply enclose the text in double quotes:

«sx08,"A rolling stone"»	saves 'A rolling stone' in phrase 08.
«sx09«is08»+" gathers no moss"»	saves as phrase 09 the phrase saved on phrase 01 and '" gathers no moss' - result: 'A rolling stone gathers no moss'

So in the WEBER.DOC example above, you don't need to save 'if file is' to phrase 02. Instead, simply do:

«sx5,«is01»+ "if file is" +«isa»»
Result: Y if file is WEBER.DOC

SUbroutine

su «su#,subroutine» where # is a-z, 0-9, 01-99, or 000-799

saves text, or a section of programming code that can be inserted or "called" at any point in a program using the **pv** command and executed. **su** stores without evaluating, and is almost identical to **sv**: it is named differently only to indicate what the phrase involved is designed for, viz., saving text, function codes, XPL statements, procedures, or complete programs─or

any mixture of them. The contents stored on a phrase by **su** are treated by **Nota Bene** as a program.

If **su** is used, it is advisable, and often necessary, to add a paragraph marker before the final closing command bracket.

If **su** is used for saving programming code, its phrase can be executed either with **pv** or with **gt**. If **sv** is used, only **pv** will execute it. Also, if **su** is used for saving text, **gt** will insert it either in the text area or on the command line, as you wish; if **sv** is used, **gt** will insert it only in the text area (see **Get Text** below)

An **su** can contain any program material, including program functions and program calls. Labels within a subroutine should not duplicate labels in the main program. If they do, a go-to-label call «gl...» (see **Go to Label** below) may find the wrong label, and prevent the correct execution of the program

Examples:

«suj,**BC** run jump3.run**XC** » —saves on phrase key J the command to run a program called 'jump3.run'. Any time you strike **Alt-J**, that program will be run.

For an alternative method of using a phrase key to run a program without loading that program on the key see **Loading indirectly on a Phrase Key** in Chapter 9

«su101,**DP AS BF CP AS XD** » —saves on phrase 101 the program for copying paragraphs from one file to another (see **Sample Program: Text & Program Functions** in Chapter 5

An economical and efficient way of running a program is

(i) to create the program, and save it complete in a subroutine, as in «su199,<program>»;

(ii) to create a second program consisting solely of the matching «pv», in this case «pv199»; then to run the latter from a keyboard key or a phrase key, or to add it to the U2 compendium.

See **Program using subroutine** section in Chapter 5 for further discussion of such subroutines.

Inserting a Phrase

There are three different commands for inserting phrases, i.e., inserting what has been previously saved to a phrase. They are **pv**, **gt**, and **is**. If either **sv** or **su** was used to save, either **pv** or **gt** can be used to insert; rules determining where and how they will insert are given in the appropriate sections below. **is** inserts a string only inside the expressions **sx** and **if**.

Put Variable ———————————————————————————————

pv «pv#» where # is a-z, 0-9, 01-99, or 000-799

The **pv** program call has a double function, depending on whether it has been saved as a string, or as an expression. The phrase can be a string saved with **sv** above (or defined and saved using «sv#» during a program); or a string that has been "operated" upon and saved with **sx** (see **sx** above), or a numerical result of an expression. If the phrase is a string of text (alphabetical and/or numerical) that has been saved with **sv** (or defined and saved using «sv#» during a program), it inserts that text, and does so sequentially (like a fast typist), rather than instantly. If the string has been saved inside an expression, with **sx**, **pv** is taken to be a number, not a string. It can be, e.g., added to a number (as in «sx10,«pv11»+1») or to another numerical phrase (as in «sx10,«pv11»+«pv12»»); see examples under **sx** above. It can be evaluated, either against a natural number (e.g., expressing the condition 'if «pv10» is greater than 1'), or against another phrase that contains numerical values (e.g., the conditional 'if «pv10» is greater than «pv11»').

Additionally, if the string has been saved with **su** rather than **sv**, and is a program, or a portion of one, then **pv** will execute it, instead of inserting the string into text or command line.

If a **pv** is used in a program to insert a command bracket in a file that is on screen in Page Layout View, the "**Extra « bracket**" message appears until the closing "»" is added. To avoid this, have the program switch the display mode of the file to Show Codes View, or insert the phrase by using the "get text" (**gt**) program call (see next page).

Examples: (based on **sv** examples in **Save Variables**)

BC «pv01»	—puts "Y" on command line
LE «pv56»	—puts "49.034" at end of line
BC ca «pva»	—puts phrase "a" on command line
BC «pv3»	—puts «IP5» on command line

There is one case where «pv#» is used, although there has been no previous «sv#,»; it is the only case where **pv** can be used without an earlier **sv**, **su**, or **sx**.

«pv00»	—puts into the program either part, or all, of the current contents of the command line.

If the command there is followed by a separator, such as a comma (or a space), and that followed by a string of characters (known as an '**argument**'), then «pv00» inserts the argument into the program. E.g., if the command on the command line is 'run program.run,today', «pv00» inserts 'today' at the cursor location. Or suppose you wanted a program that would locate the cursor at a specified number of bytes from the top of the file. This program would do it:

BC jmp «pv00»**XC**	If the program were called 'jump.run', the command **run jump.run,25000** would locate the cursor 25000 bytes from the top of the file.

If the command does not have a separator followed by an argument, e.g., 'run program.run', then «pv00» inserts the command itself into the program.

If the string in **pv** is programming code, and has been saved with **su**, **pv** will execute it. Example:

«su25,**DF CL CL CL CL DF** »«pv25» —will execute that portion of the program

Get Text ——————————————————————————————

gt «gt#» where # is a-z, 0-9, 01-99, or 000-799

inserts saved string at cursor position in the text, if the string has been saved with **sv**. It inserts it instantaneously, as if it were a defined block being copied or moved (and therefore is quicker than **pv**); and it leaves the cursor at the beginning of the string, as contrasted with **pv**, which leaves the cursor at the end of the string. (You can make **gt** leave the cursor at the end of the string by embedding the following code immediately after the **gt** delta:

«sx02,«cp»»«sx03,@size(«is01»)»«sx04,«pv02»+«pv03»»**BC** jmp «pv04»**XC**

substituting for the 01 in «is01» the number of the phrase in the «gt..» call.)

gt cannot be used to insert text into an expression; **is** (see next section) must be used for that. **gt** will not insert a string on the command line, unless the string has been saved with **su**. In that case it will insert the string either in the text or on the command line, whichever you want. Also, if what has been saved with **su** is a program, or a portion of it, **gt** can be used to execute it.

Examples:

«sv25,A passage of text»«gt25» —enters string of text in text
«su25,A passage of text»**BC** «gt25» —enters string of text on command line
«sv25,**DF CL CL CL CL DF** »«gt25» —enters string of code in text
«su25,**DF CL CL CL CL DF** »«gt25» —executes a portion of program

InSert phrase ——————————————————————————

is «is#» where # is a-z, 0-9, 01-99, or 000-799

inserts string (text, or numbers considered as text) within an expression. **is** is used only within **sx** and **if** statements, and only if the string consists of text and/or numbers treated as text. If the string 'Dragonfly's' has been saved to phrase 01, and the string '**Nota Bene** 4.2' to phrase 02, then «sx03,«is01»+«is02»» would save 'Dragonfly's **Nota Bene** 4.2' to phrase 03, and «gt03» would insert it into a file's text.

is is also used in programming

—for comparing strings (e.g., 'if «is01» is textually the same as «is02»')
—if the expression within which it is being used contains certain operators:
— + of concatenation (i.e., joining two strings, as in the previous example; **not** the + of mathematical addition)

—**î** of inclusion
—**@siz**, **@upr**, **@cnv**. See the **Operators** sections, below, p 68

Examples:

«sv01,Dragonfly's »«sv02,**Nota Bene**»«sx03,«is01»+«is02»»«pv03»
—enters Dragonfly's **Nota Bene** in text
«sv01,10»«sv02,20»«sx03,«pv01»+«pv02»»«pv03» enters 30 in text
«sv01,10»«sv02,20»«sx03,«is01»+«is02»»«pv03» enters 1020 in text

Other Calls

IF

if «if(expression») (parentheses are optional)

a conditional expression that begins a program segment that is to be executed if the stated expression is true, or skipped if it is false. Unlike many ordinary conditionals, which state what is/will be the case, or what is to be done, or will happen, *if* the conditional is fulfilled, but leave it open what is/will be the case, etc., if the conditional is not fulfilled, the 'ifs' of programming are narrower and more rigid. They stipulate both what the program is to do if..., and what it is to do if not...; i.e., they are always of the form 'if such-and-such, do this, otherwise/else do that'.

If the conditional is true, all code between the **if** and the closing «ei» is executed.
If the conditional is false, the program begins to execute the code that immediately follows the **ei**.
The result of the "evaluation" is **not** saved in a phrase key, unlike **sx**.

Examples: (based on original **sv** examples)

«if«pv56»==46»	—**False**: phrase 56 has a value of 49.034.
«if«pv56»==49.034»	—**True**: Value of phrase 56 is equal to 49.034.
«if«pv56»>49.034»	—**False**: 49.034 is not greater than itself.
«if«pv56»<=49.12»	—**True**: Value of phrase 56 is less than 49.12.
«if«is56»=>40»	—**Command entry error**: A string (referenced by 'is') can't be compared with a number (40)considered as a number, not as a string.

Assuming that «sv41,Mauss» and «sv42,Durkheim» have been assigned, the following expressions have the specified values:

«if«is56»==«is41»»
False: The string saved on phrase 56 (49.034) is not identical in sort sequence to that on phrase 41 (Mauss).
«if«is42»<«is41»»
True: The string saved on phrase 42 (Durkheim) comes before that saved on phrase 41 (Mauss) in sort sequence.

You cannot nest **if** conditionals, as you can in other programming languages. But you can achieve the same effect by jumping to a label (see final example in this section), evaluating a second conditional, and then returning to the original position (which has been tagged with a label of its own).

[Read sections on End If, etc.., on following pages, then return to this point:]
End If («ei»), **LaBel** («lb»), **Go to Label** («gl»), **ERror** («er»), **EXit** («ex»)]

Because **if** requires that the program specify what is to be done if the «if» conditional is false, it is essential that every «if» be complemented by an «ei», indicating that that is the end of the conditional clause, and leading directly to the specification of the course to be followed if the conditional is false.
Examples:

«if«is56»==«is41»»«glNEXT»«ei»«ex»f the strings on phrases 56 and 41 are identical, go to label NEXT; otherwise exit program

BC se \clause**XC** «if«er»»«ex»«ei» —search for the next occurrence of the string 'clause'; if there is an error, i.e., if no further occurrence is found, exit program

«lbSEARCH»**BC** se \clause**XC** «if«er»» «ex»«ei»«glSEARCH» —if there is no error, i.e., if another occurrence of 'clause' is found, go to the label SEARCH, and repeat this part of the program: search for the next occurrence of 'clause'.

The third of those examples completes the second: it specifies both what is to be done if no further occurrence of 'clause' is found and what is otherwise to be done. In this case it prescribes a loop, i.e. that the search for occurrences of 'clause' is to be continued until no further occurrences can be found. The following example would set up an endless loop:

«lbSEARCH»**BC** se \clause**XC** «glSEARCH»

It requires a search for the next occurrence of 'clause' to go on repeating itself indefinitely, with no provision for what is to be done when no further occurrences can be found. This illustrates the need

(i) for the combination of **if** and **er**, providing for the event of there being no more occurrences of 'clause';

(ii) for **ex**, providing for a way of ending the program and thereby preventing an endless loop; and

(iii) for **gl** and **lb**, providing for the search to continue as long as occurrences of 'clause' are to be found.

«lbSEARCH»**BC** se \clause**XC** «if«er»»«glIF»«ei»«glSEARCH»«lbIF»«if...» —this illustrates the branching of «if»s. The first one stipulates that, if no occurrence of 'clause' can be found, the program go to label IF; and at that label a second «if..» starts.

End If

ei «ei»

marks the end of a segment of program code that begins with **if**. It is essential that every «if» segment be concluded with an «ei». Otherwise the code that follows will be interpreted as part of the conditional clause, instead of specifying what is to be done if that conditional is false; the program will not execute correctly.

The code after the **ei** will be executed regardless of the truth or falsity of the **if** clause, unless that clause

(i) contains an **ex**, which terminates the program , thereby preventing the code following the **ei** from being reached, or

(ii) contains a **gl**, which directs the program to a label and may bypass the code immediately following the **ei**.

LaBel

lb «lbNAME»

labels a point in the program to which you can later jump (either backward or forward) to resume execution from that point.

Label names must exactly match in spelling and case the names used in goto-label (**gl**) commands. If the match is not exact, the «gl..» will be unable to find the correct «lb..»

Do not duplicate a label name within a program, or in two programs if one is a sub-routine of the other. If a label name is duplicated, the goto command may find the wrong one; and the program will not execute correctly.

Labels can be inserted at any point within a program. Although there should be a label matching every goto command, the reverse is not true. Therefore labels (unmatched with gotos) can be freely used in a program. A label containing nothing but a paragraph marker is useful for breaking up a long program (see example below), thereby making it easier to read and edit.

Labels can be of any length (though shorter labels are easier to use).

Labels can be used for comments to yourself within a program (such as why something was done a certain way). This can be helpful when writing or revising a long or complicated program, because the comments will serve as reminders of what this particular section of programming code is doing.

Labels inserted from the command line (with **F9 lb label F10**) will retain their case exactly as typed (see note below)

Examples:

«lbtest» —creates label called "test"

«lb—return here when finished with #5»
—creates label indicating when to return

GO to Label

gl «glNAME»

goes to the label bearing the same name and continues execution with the commands at that point.

The **gl** name must exactly match the **lb** name in spelling and case.

If you enter a **gl** delta from the command line (with **F9 gl label F10**), the label name will be inserted with all capital letters. This is not the same result as inserting **lb** deltas (see note above); therefore, it is important to check that the **gl** and **lb** names are strictly identical.

Because the program goes directly (either forward or backward) from a «gl..» command to its matching «lb..» command, a paragraph marker can always be inserted after a «gl..» command; it must come **after** it, not inside the «gl..» itself. It will not be interpreted as part of a program, and the insertion of a blank line at that point serves to break the program up at that stage, making it easier to read and to edit.

This feature of a «gl..» command, that it jumps straight to its corresponding «lb..» command, disregarding anything in between, can be used to insert at the start of a program file any comments on it, or explanation of it, that the author wishes to include. If, for example, the comments are prefixed by the goto command «glSTART», and if the program coding is prefixed by the label «lbSTART», the comments can be read by calling the program file to the screen; and the program can be run correctly, because, having read the «glSTART» command it will jump immediately to the«lbSTART» label, disregarding everything in between.

Alternatively comments and explanations can be inserted into the file after the program codes. The program will quit when it reaches an «ex»; and therefore the textual matter in the comments will not interfere with its execution.

eXtract String ——————————————————————————————

xs «xs#,#,#,#,#,» where each # is different, drawn from a-z, 0-9, 01-99, or 000-799

parses the string saved on the first phrase in a way that makes it possible to extract and use each of the component parts of the string independently of the others. The analysis of the string is determined by the values of the first two phrases.

The **first #** is the number/letter of the phrase where the string to be parsed is stored. The string could be the command on the command line, or an argument following that command (in which case the phrase's # will be 00), or it could be a phrase to which a string or expression in the program has previously been saved (in which case it will have that #)

The **second #** is the number/letter of the phrase where you have stored the string that you wish to be the parsing operator

Example:

If you wanted to be able to parse the filename MYFILE.TXT, perhaps in order to have a program change either the name or the extension, then

«sv01,.»«sv02,MYFILE.TXT» would provide the first two phrases of the «xs» call:

«xs02,01,#,#,#»

The remaining three #s can be any that you choose for saving the three parts of the string to be parsed

#3 will contain that part of the initial string that *precedes* the parsing operator

#4 will be identical with #2, with one exception (see below)

#5 will contain that part of the initial string that *follows* the parsing operator

If the phrase numbers for #3, #4, and #5 are, say, 03, 04, and 05, the total **xs** call in this case will be «xs02,01,03,04,05», with these values:

02 MYFILE.TXT
01 .
03 MYFILE
04 .
05 TXT

It is now possible for a program to perform operations on this filename without affecting its extension, or vice versa.

[Explanatory note: MYFILE.TXT consists of 3 parts: (1) MYFILE; (2) . (a period); (3) TXT. XS separates (parses) them , putting each part into one of the last 3 phrases: 03, 04 and 05. 03 contains MYFILE; 04 contains . ; 05 contains TXT.]

The string to be parsed and the parsing operator (in phrases 02 and 01 in the example) must neither of them be numbers. If they are, **xs** will not work; it works only with string data. If a number is first converted to a numerical string [of numbers-as-text], then it will work. If you wanted to use **xs** to preserve only the integer in the *number* 1234.56, it would not do it; but, if you first converted that to the *numerical string* 1234.56, it would.

xs makes it possible to perform operations that previously were either extremely difficult or even impossible. **Nota Bene**'s two command brackets (« and »), for example, create problems, if you try to introduce them as characters in a program. But with **xs** you can save each of them to a phrase, and then use that phrase in a program as you wish:

```
Program to insert command brackets in program
«sv01,.»«sv06,«.»»«xs06,01,02,03,04»
06      «.»
01      .
02      «
03      .
04      »
```

Wherever the program calls for the insertion of « or », «pv02» or «pv04» will do it.

Wildcards (see below) may be used as (or in) the parsing separator; and this is the one case where the string saved at #2 and that saved at #4 will not be identical: in #4 the wildcard of #2 is replaced by the actual text that matches it.

Another advantage of **xs** is that it can be used recursively. This enables you, for example, to get a program to branch to a specified one of a possibly long list of options; or to remove a number of unwanted characters from a string. Changing a file's filename and drive/path to just its filename (e.g., changing 'c:\nb\prgrm\mailclr.run' to 'mailclr.run') can be automated by getting **xs** successively to delete each '\' and the string preceding it until there are no '\'s left.

Wildcards

This list is a visual representation of wildcards. Chapter 8 contains the same list, but with actual wildcards. The wildcards in Chapter 8 can be copied and pasted into programs; these cannot.

- or ^0-^9	Defines maximum no. of times the character can appear in the string
or or ^A	Any single letter or number
or ^B or ^-	Any but next single character (represents NOT)
or ◂ or or ^C	Carriage return character [Ascii 17, ']
■ or ^E or ^+ ??	Any single sentence separator
or ^F	Line Feed Character
or or ^L	Any single letter A-Z
or or ^N	Any number 0 through 9
or ^O	Allows search for more than one string
or ^P	Regular or Alternate paragraph return
or ^R	Regular paragraph return Carriage return+linefeed (Enter with 'func WC')
or or ^S	Any single separator
or or ^T	Tabs
or or ^W	Any string from 1 to 80 characters. Must be used with at least 1 other character. 'se /x^W/' works; 'se /^W/' doesn't.
or or ^X	Any single character

To put a wildcard that looks like a reverse-video single character into a program or the keyboard, do 'func nn' plus the character.
To put one of these on the command line, do F9 func nn F10, then press the appropriate letter or number (e.g., 'n' for any single number). The wildcard will appear on the command line at the end of the 'func nn' command. You can erase 'func nn' and substitute (for instance) a search command.
To put one into a program, do F9 func nn Alt F8 F10
To put double-character wildcards into a program, do 'pfunc' plus the 2 characters.
To put these on the command line, enter them into text with pfunc, then cut and paste to the command line.
To input caret + letter wildcards (e.g. ^L) into text or on the command line, type the caret character plus the letter.

In NB for Windows, you can use wildcard characters on either side of a change string. E.g.,
BX ci /rolling^Wmoss/rolling stone^Wmoss/**Q2**
will change 'A rolling gathers no moss' to 'A rolling stone gathers no moss'

ERror

er «er»

used in conditional expressions to indicate an error condition, such as a specified file not being found, or the failure of a search.

The result is true if there was an error in the previous command; otherwise, it is false.

The value-of-error (**va$er**) command can be used to display the numerical code corresponding to a specific error condition (see **Values** section for details).

Examples:
BC se \Handel**XC** «if«er»»**BC** ab**XC** «ei»
—abandons file if "Handel" is not found

BC ca memo.522**XC** «if«er»»«glnextfile»««ei»
—goes to "nextfile" label if MEMO.522 doesn't exist

EXit

ex «ex»

exits from the current program (and continues with the main program if the current program was a subroutine)

ex1 «ex1»

exits from the program entirely, regardless of whether you were in the main program or in a subroutine

In most cases **ex** is sufficient to terminate a program; it will always do it, if it is part of the main program. Because **ex1** will always halt a program completely, it should be used with care. Sometimes, when a program will not run through to its end, it is because an **ex1** has been encountered where there should have been an **ex**. At other times, when it looks as if an **ex** will be sufficient, an **ex1** will prove to be necessary, to get the program to clear completely.

Error Suppression ———————————————————————————

es **BC** es #**XC** ES 1 to suppress bell and error messages

In a program in which beeps and error messages would otherwise occur (e.g., one involving a Search command that would beep and display the 'Not Found' message) it saves time and avoids interference, if you include in the program the command **BC** es 1**XC** to suppress them. In NB for Windows it is no longer necessary to reactivate **ES** with command **BC** es 0**XC**.

Read Character ———————————————————————————

rc	«rc»	reads character typed on keyboard
rk	«rk»	ditto, reading it as upper case

The character read can be thrown away, saved in a phrase or evaluated (see **String Operators** section)

Examples:

«rc»	—reads character; inserts it or performs function
«sx01,«rc»»	—reads character, saves it as phrase 01 for later use
«sx01,«rc»»«pv01»	—reads character, saves it as phrase 01, inserts it into text

The effect of «rc» is that the program pauses for you to strike a key, and resumes as soon as you have. How it resumes depends on the instructions that follow the «rc» call. A common use is in programs where the user has to make a choice between various options, such as 'Y/N'; if the user enters Y, the program goes one way; if N, it goes another way.

Example:

«sv02,Y»«sx01,«rc»»«if«is01»==«is02»»«glOne»«ei»«glTwo»

If the key pressed is Y, the program goes to label One, otherwise it goes to label Two.

The example illustrates two further points:

(i) If the key to be matched is uppercased, e.g., Y, then striking lowercase y when the program pauses at the «rc» call will not do. Y is not identical with y (Y is ASCII 89, y is ASCII 121), and so the condition in «if«is01»==«is02»» will not be satisfied. Pressing y will have the effect that the program will follow the second option, not the first that you wanted. There are two ways to prevent that. One is to use the string operator **@upr** (see **String Operators** below), which uppercases the character, if it has been entered in lowercase. In the present example, if the program had been:

«sv02,Y»«sx01,@upr(«rc»)»«if«is01»==«is02»»

then pressing either Y or y would cause the program to take the Y option. The character you entered (say, y) was saved on phrase 01, and then the character (y) saved on 01 was uppercased and saved again on 01 (i.e., as Y).

(ii) If there are just two options, as in this case Y and N, there is no need to specify the second. It is sufficient that the program is told what to do if Y is pressed, and what to do if some other key is pressed. When running the program, if you don't want option Y, it does not matter what other key you press: the program will take the second option. If you had included in the program the call «sv03,N», then you would have also to have «if«is01»==«is03»«glTwo» «ei», and you would have to press N, if you wanted the second option; no other key would do. It is, therefore, more economical and efficient, when you are writing a section of program that requires just two options, not to specify a key that must be pressed if the second option is to be chosen.

The other way round the problem caused by the difference between lower and upper case is to use **rk** instead of **rc**. The only difference between the two calls is that «rk» automatically uppercases the string that is entered. It is suitable for an instance like the present, where the string consists of only one character, but in instances where the string to be entered consists of more characters than one, it can be a nuisance: you do not always want everything that you enter at the keyboard to be uppercased. Also, with some characters, the use of **rk** simply reverses the shifted and unshifted characters: whereas «sx01,«rc»»«sx01,@upr(«is01»)» will record all the keyboard's numerical keys correctly, whether they are entered shifted or unshifted, «rk» will record« both 1 and ! as 1, 2 and @ as 2, etc.

Here is an example of using «rk», taken from the XYWWWEB.U2 program compendium. It uppercases the character typed at the keyboard, and proceeds to fulfil the if-clause if the character typed was y or Y.

```
«sv02,Y»«sx01,«rk»»»«if«is01»î"Yy">0»
```

Often, you will want a program to pause while you enter from the keyboard, not a single character as in the 'Y/N' example, but a string of characters, such as a filename, or a string to be searched for. «rc» by itself is not sufficient for that. But you can write a section of program in which you instruct the program

(i) to read the character you enter;
(ii) if the character is identical with one that you have previously specified, to continue with the program's execution;
(iii) if it is not identical, then to save the character to a phrase, and to append that to the phrase on which the original character has been saved (in (i) above);
(iv) to continue the process until the key specified in (ii) is struck. Let us suppose that the specified key is the asterisk, then the following section of coding will do what you want

Routine to read keyboard input

```
«sv20,*»«sv26,»«lbChain»«sx25,«rc»»«if«is25»==«is20»»«glResume»«ei»
«sx26,«is26»+«is25»»«glChain»
```

If the key struck is the asterisk key the program goes to the label Resume (somewhere else in the program); otherwise it saves to phrase 26 what was originally there (nothing) plus the character saved on phrase 25, and starts the loop again. If the next character entered is still not the *, then the program saves on phrase 26 the character that was already there, plus the new one; and so on, increasing the string saved on phrase 26 character by character, until the * key is finally struck. Eventually, when the * key is struck, the program goes to the label Resume. If «lbResume» is followed by, say, «gt26», the total string saved on phrase 26 will be inserted in the text at that point.

Cursor Position ———————————————————————————————

cp «sx#,«cp»» where # is a-z, 0-9, 01-99, or 000-799

used within an expression to indicate the current cursor position, expressed as the number of characters from the beginning of the file

See notes under **jmp** below for what counts as a character.

Examples:

«sx21,«cp»»**BC** Cursor at «pv21» bytes from Top of File

—displays on the command line a message reporting the current cursor position

«sx21,«cp»»**BF** «sx22,«cp»»«sx22,«pv22»+1»**BC** jmp «pv21»**XC** **BC** «pv22» characters in file

—displays on the command line a message reporting the number of characters in the file, and returns to current position in file.

«sx21,«cp»»**BF** «sx22,«cp»»«sx22,«pv22»+1»«sx23,(«pv22»/7)»**BC** jmp «pv21»**XC** **BC** About «pv23» words in file

—similar, but reports approximate number of words in file

Column Location ———————————————————————————————

cl «sx#,«cl»» where # is a-z, 0-9, 01-99, or 000-799

used **within an expression** to indicate the current column location

The columns are numbered from 0 through 254

Examples:

«sx01,«cl»»«sx02,«cp»»**BC** Cursor is in column «pv01»**GT**

—saves current column location to phrase 01, and current cursor location to phrase 02

BC jmp «pv02»**XC**

—makes cursor jump to column location saved on phrase 01 (identical with cursor location saved on phrase 02)

BC ip 0,«pv01»**XC**

—sets hanging indentation at column saved on phrase 01

cl can be used for a variety of other purposes, such as drawing a line between predetermined points. Note: a program cannot directly jump (see following section) to a given column location, only to a character location identified with it, as in «sx01,«cl»»«sx02,«cp»» above. The jump would have to be made to 02, not to 01. But the difference between two column locations in a single line can be calculated, and made use of by the program.

JuMP ——————————————————————————————

jmp BC jmp #**XC**
causes cursor to jump to character # from beginning of file

Embedded commands are counted as they appear in Expanded Display mode.
The paragraph marker is counted as two characters (a carriage return and line feed).

Characters entered as three-character sequences are counted as three characters (see Reference Manual, p. 387).

Example:
BC jmp «pv02»**XC** —jumps to cursor location saved on phrase 02

The following program would cause the cursor to jump to any location in a file that you indicated by the argument on the command line:
BC jmp «pv00»**XC**. If you call that program JUMP.RUN, and run it with the command **BC** run jump.run,#**XC** replacing an argument # with the number of the location you wanted to go to, the program would position the cursor at that place in the file.

Argument Insert——————————————————————————

as «as»

"passes" the string typed after the program's filename on the command line (after a comma, or other separator, following the **run x:program** command) to the program so that, for example, the program can operate on the designated file at the indicated point within the program.

«as» performs just the same function as «pv00», with one difference. If the program containing «as» is run with the **run** command, the «as» records only the argument on the command line (if there is one); if there is no argument-string there, it does not record the command itself. But «pv00» will do either, and is to be preferred.

Examples:
BC ca «as»XC
—if that command is embedded at a certain point in a program, then at that point it will call the file that is specified by name, after program name, when the program is executed. E..g, if a program called SEARCH was run with **run search,demo**, the file DEMO would be called at the indicated point in the program

Mathematical Operators

+ addition «sx20,4.25+17»
- subtraction «sx21,100230-45374»
/ division «sx22,1000/.0825»
* multiplication «sx24,2.34*9.56»
These operators can also be used in combination with each other, using parentheses as required:
«sx25,45+(.059/(56.34*102))-.034»

Note: the + of addition must be distinguished from the + of concatenation: the first produces the sum of the two strings (as numbers), the second joins the second string to the first string (both as text)
Examples:

«sv01,10»«sv02,15»«sx03,«pv01»+«pv02»»**GT** «pv03»
—in phrase 03 adds the value of phrase 02 to that of phrase 01, and inserts the result into text area as 25

«sv01,10»«sv02,15»«sx03,«is01»+«is02»»**GT** «pv03»
—in phrase 03 joins the string in phrase 02 to the string in phrase 01, and inserts the result into text area as the string 1015

Comparative Operators

== equal to (double "==" is required)
< less than
<= less than or equal to; variant form: =<
> greater than
=> greater than or equal to; variant form: >=
<> less than or greater than (not equal to)

For numbers, these expressions compare numerical values; for strings, they compare sort sequence. Two strings are identical if they have exactly the same sort sequence (down to the last character).

Examples:

«if«pv01»=>2»«gl2»«ei»
«if5==2»«glnotso»«ei»
«if«pv29»<>470.45»«gltax3»«ei»
«if«is30»==«is40»»«prLines are identical»«ei»

Logical Operators

& performs a logical **and** of two values
«if((«pv50»>25)&(«is28»==«is29»))»
True only if both expressions are true

! performs an **inclusive or** of two values

«if((«pv50»>25)!(«is28»==«is29»))»
True if either or both expressions are true

@Xor performs an **exclusive or** of two values
«if((«pv50»>25)@Xor(«is28»==«is29»))»
True if one, but only one, is true

@not performs a **not** of the following value
«if@not(«pv25»==10)»
True if phrase 25 is not equal to 10

Sometimes in programming it is more convenient to use a negative conditional, e.g., "if there is no error, then...", as in
«if@not(«er»)»«glA»«ei»«ex»
—if there is no error go to label A, otherwise exit

String Operators

Element of ——————————————————————————————

î (ASCII 238)«sx#,«is#»î«is#»»1 where # is a-z, 0-9, 01-99, or 000-799
determines if one string is contained in another
The i-circumflex can be entered with Ctrlt+Shift+ 238.
If the first string is not contained within the second, the result is indicated as "-1".
If the first string is contained within the second, the result is given as the character position in the second string at which the matching portion begins.
Note: the first position in the containing string is **0**, the second **1**, etc.

Examples:
«sv01,Mark Twain»«sv02,Mark Twain»«sx21,«is01»î«is02»»
The phrases begin to match at the beginning of the second phrase (i.e., at the 0 position). Result (recorded as phrase 21): 0.

«sv01,Twain»«sv02,Mark Twain»«sx21,«is01»î«is02»»
The phrases begin to match at the sixth position of the second phrase.
Result: 5.

«sv01,M. Twain»«sv02,Mark Twain»«sx21,«is01»î«is02»»
String "M. Twain" is not contained in "Mark Twain". Result: -1.

NB: In Nota Bene for DOS, the î operator displayed as an ε.
The **î** operator, providing a means of detecting whether a character/string is included within another string, can be used for prescribing different courses to be followed, depending on the location in the second string at which the first character/string begins to match it. It is particularly helpful in programs where the user has to make a choice among a number of options: "If A, then..., if B, then ...,if...etc."
Example:

Routine to branch to label whose letter the user inputs at keyboard
«sv01,ABCD»«prEnter A, B, C, or D»«sx02,«rk»»«if«is02»î«is01»<0»«glEnd»
«ei»«if«is02»î«is01»==0»«glA»«ei»«if«is02»î«is01»==1»«glB»
«ei»«if«is02»î«is01»==2»«glC»«ei»«if«is02»î«is01»==3»«glD»
«ei»«lbEnd»**BC** Wrong character entered«ex»
The program aborts if none of the specified letters is entered, i.e., if the letter entered is not contained in the string saved on phrase 01. If the letter entered is contained in the string, the program goes to labels A, B, C, or D, according to the position in the string of the letter which is entered

Containment

ð (ASCII 240) determines if one string contains another (true or false) *[new in NBWin*]
It returns "TRUE" if string1 contains string2.
It is principally used in conditional tests, where the position of string 2 within string 1 is unimportant. It is case sensitive. E.g., this program segment:
«IF"limpet"ð"limp">«PR OK»«EX»«EI»«PR Not OK»«EX»
returns 'OK'—but this segment:
«IF"limpet"ð"Limp">«PR OK»«EX»«EI»«PR Not OK»«EX»
would not.

Size

@siz «sx#,@siz(«is#»)» where # is a-z, 0-9, 01-99, or 000-799; parentheses required

checks the number of characters in a string

Example:
«sv21,Jurgen Habermas»«sx22,@siz(«is21»)»
records that "Jurgen Habermas" has 15 characters

This call, which is used with «is», is useful for detecting whether a character has a value of 1 byte (as most, although not quite all, keyboard characters have), or is 3 bytes in length, as, for example, all function codes (such as **BC** , **XC** , **DF** ,etc.) are

Example:
Routine to branch depending upon whether FN or Alphanumeric key pressed
«sx01,«rc»»«sx02,@siz(«is01»)»«if«pv02»==3»«glA»«ei»«glB»
makes the program branch one way if the key struck was a function key, another way if it was an alphanumeric key

Uppercase

@upr «sx#,@upr(«is#»)» where # is a-z, 0-9, 01-99, or 000-799; parentheses required

uppercases the designated string

Examples;

«sv03,Nasa»«sx23,@upr(«is03»)»
reads "Nasa" and records "NASA" as phrase 23

Routine branches to label 'cont' if 'y' or 'Y' is struck.
«sv10,Y»«sx30,«rc»»«sx31,@upr(«is30»)»«if(«is31»==«is10»)»«glcont»«ei»
records "Y" as phrase 10, reads character typed and records it as phrase 30, uppercases phrase 30 and records it as phrase 31, compares phrase 31 with phrase 10; if "y" or "Y" was pressed, goes to label "cont".

Sometimes a program requires the user to enter a letter, as in the 'Y/N' choice, and will branch one way or the other, according to the letter struck. But, as in the example above, it will recognize only uppercase Y, not lowercase y. **@upr** uppercases a lowercase letter, if one was struck, so that the program will recognize it. The call **rk** combines into one the two calls **rc** and **@upr**, but sometimes gives unwanted results (see **Read Character**, above).

CoNVert ——————————————————————————————

@cnv«sx#,@cnv(«is##»)» where # and ## are a-z, 0-9, 01-99, or 000-799*;*
«sx##,«rc»» has been set; parentheses required

takes a function call read from keyboard (using «rc» command) when a function key is pressed and converts it into the corresponding two-character keyboard function/mnemonic.

Command entry error ─ The key has a character, not a function, assigned to it.

Example:

«sx40,«rc»»«sx41,@cnv(«is40»)»**BC** «pv41»
pressing "y" results in "**Command entry error**"
pressing **F10** results in XC (execute)
pressing <left cursor key> results in CL (cursor left)

Note: the XC and CL are the 2-character **keyboard** functions (or function mnemonics), to be distinguished from **XC** and **CL** , the 3-character **program** functions (or function codes). In fact, the **@cnv** function does the opposite of what is done in Program-Recording (see Chapter 1, **Recording Program Functions**). There when, for example, the **F9** key is pressed, the keystroke is converted into the embedded function **BC**. Here, when an **F9** keystroke is processed by **@cnv**, it is converted into the two characters BC that would be used to assign the function to a key in a keyboard table.

Other operators ——————————————————————————

[See also Operators section of Chapter 8.]

@ Operators

@int save result of calculation as an integer (throw away fractional value, if any)
@abs returns absolute value of a number or calculation, i.e., the numeric result without regard to sign
@dec Convert hexadecimal number to decimal number

@hex Convert decimal number to hexadecimal number
@dat convert date to hexadecimal number
@dts Convert hexadecimal date YYYYMMDD to decimal in format determined by default FZ. These two are used to compare two input dates, for instance, to determine which is earlier
@lwr Lower Case function
@num Changes datatype of phrase from string to number (numbers have an invisible 2-byte flag, consisting of Ascii 0 followed by Ascii 1, appended to them in memory and therefore are 2 bytes longer than their string counterparts)

Values

[There are hundreds of valid values. See Chapter 9, on variables.]
The value command (**va**) reads and inserts into the file at the cursor location the current value of a status variable. Settings for variables other than defaults are preceded by a $. The display (seen only in Page Layout View) can be removed by one stroke of the **Backdel** key.

Examples:
BC va $pa**XC** displays the current drive and path
BC va ts**XC** displays the current tab settings

The following is a partial listing of the variables, showing how the value of each can be embedded in a program

drive and PAth ———————————————————————————

va$pa «sx#,«va$pa»» where # is a-z, 0-9, 01-99, or 000-799
reports drive and current path

Filename ———————————————————————————

va$fi «sx#,«va$fi»» where # is a-z, 0-9, 01-99, or 000-799
reports name of file in active window

Filename and path ———————————————————————————

va$fp «sx#,«va$fp»» where # is a-z, 0-9, 01-99, or 000-799
reports filename and drive/path of file in active window

PaGe number ———————————————————————————

va$pg «sx#,«va$pg»» where # is a-z, 0-9, 01-99, or 000-799
reports current page number (the page-line counter must be on)

Line Number

va$ln «sx#,«va$ln»» where # is a-z, 0-9, 01-99, or 000-799
reports current line number (the page-line counter must be on)

MEmory

va$me «sx#,«va$me»» where # is a-z, 0-9, 01-99, or 000-799
reports amount of memory currently free

Window Number

va$wn «sx#,«va$wn»» where # is a-z, 0-9, 01-99, or 000-799
reports number of active window

Window Status

va$ws «sx#,«va$ws»» where # is a-z, 0-9, 01-99, or 000-799
reports status of current window:
0 = no file open *[doesn't seem to work in NBWin]*
1 = file open
2 = directory open *[doesn't seem to work in NBWin]*

File Status

va$fs «sx#,«va$fs»» where # is a-z, 0-9, 01-99, or 000-799
reports which window(s) contain file(s), using sum of values from the following list:

1	if window 1 contains file
2	if window 2 contains file
4	if window 3 contains file
8	if window 4 contains file
16	if window 5 contains file
32	if window 6 contains file
64	if window 7 contains file
128	if window 8 contains file
256	if window 9 contains file

Examples:

If windows 1, 4, and 5 contain files, the value is 25
(1 + 8 + 16).

Display Type

va$dt «sx#,«va$dt»» where # is a-z, 0-9, 01-99, or 000-799
reports document display mode in use for current file:

0	Show Codes View
1	Draft View without page breaks
2	Draft View with page breaks
4	Page Layout View *[for other types see Allcodes, entry on DT]*

ERror Code

va$er «sx#,«va$er»» where # is a-z, 0-9, 01-99, or 000-799
reports code number of error condition

format commands

vaxx «sx#,«vaxx»» where # is a-z, 0-9, 01-99, or 000-799
and xx is format command
reports current setting of format command *[not tested for NBWin]*

default settings

vaxx «sx#,«vaxx»» where # is a-z, 0-9, 01-99, or 000-799
and xx is default setting
reports current setting of default settings (as in NB.DFL).

Miscellaneous Commands

Two commands have, to improve clarity, been discussed earlier (see **Error Suppression** and **JuMP** above)

Pause

p **BC** p**XC**
causes program to pause for about one second before continuing (actual duration depends on hardware configuration). For longer pause increase the number of **XC**s

Example:
BC p **XC XC XC XC XC XC** ─pauses 6 seconds

Wait

wait **BC** wait**XC**

causes program to finish a background task, such as printing, before continuing
Without **wait**, execution continues immediately
Example:
BC print test.doc**XC** **BC** wait**XCBC** call test.doc**XC**

Suppressing Display

DX/DO **DX** freezes video display for current window
DO reactivates video display for current window

These are less necessary than they were in NB for DOS. Try a program without them; use if needed.

Nested Programs Programs can invoke other programs. When the second program is finished, control is returned to the first. These subroutines, of course, can be saved on disk. The program described previously that reads a character from the keyboard could be turned into such a subroutine.

Interrupting Programs A program cannot ordinarily be interrupted unless it is expecting some input from the keyboard

---

Extended Phrases Of the extended phrases available for use in programs (with **sv**, **sx**, **pv**, and **is**) those in the ranges 00-99 and 000-099 are cleared when a program is exited. Extended phrases above 100 are retained in memory until the end of the session, and are available for use in other programs. Regular phrases (a-z, 1-9) can also be used by executing **sx**. For example, to save the variable recorded on extended phrase 45 as regular phrase 5, use:

«sx5,«is45»»

Numbers & Strings Numerical values should be converted to strings if they are to be used **outside a program**. Thus, «sx1,25+45.05» saves the value " 70.05" in phrase 1 for later use with the «pv1» code.

Parentheses Parentheses must be used around the item(s) operated upon by commands beginning with "@":

«if@not(«er»)» — will work
«if@not«er»»
«if(@not«er»)» — will not work
«if@(not«er»)»

Paragraph Marker Carriage Return To enter a paragraph marker as part of a search string to be implemented within a program,
'^R', e.g.: **BX**se /^R/**Q2**.
To have a program insert a paragraph marker into the actual text of a file when the program is being run, enter a normal paragraph marker with the Enter key. See also p 77.

Programming Error Messages

Too many program calls — You created an endless loop (for example, you tried to run a program that includes the command to run itself).
Mismatched operands — You cannot use string operators with numbers, compare **pv#** with **is#**, or perform mathematical operations on strings.
Command entry error — You have used **pv** when you should have used **is** (or vice versa), have attempted to perform a string operation on a numerical value or a mathematical operation on a string, or have improperly entered the command or mistyped the operator (for example, have only one "=" instead of the pair of "==" symbols).
Label not found — The designated label cannot be found.
No «ei» — No **ei** code exists to end an **if** statement.
Need ID & expression — You attempted to enter a program call requiring a phrase-key identification, but did not specify either it or the expression to be evaluated.
Repeat w/alphanumeric — A label name must be specified. (This is the same error message used for assigning phrases [see "Phrase Libraries" chapter], so the wording is tailored for that situation.)

See also useful expansion of this topic in the Appendix, p 179.

Notes: Entering and Searching for Commands, Functions and Special Characters

1. **Embedded Commands**

Embedded commands are embedded in a file or program as codes contained within command brackets (also known as format brackets, double angled brackets or guillemets - this last being XyWrite usage).

Searching for Embedded Commands

You can search for embedded codes in Page Layout, Draft or Show Codes view. You can find successive instances of one type of code (such as labels, or italics) using only the opening command bracket in the search string.

BC se /ELB/**XC** [where 'E' is ASCII 174]
BC se /EMDIT/**XC**

will find the next label or italics code. (After finding most embedded codesyou can move the cursor to the left to read the code contents on the prompt line [to see print modes like this, you have to uncheck the box 'Never show type-style commands' in Tools, Preferences, Document Views].)

Entering embedded commands in programs:

You can do it from the command line—for instance:

F9 md +bo**F10** [note space after 'md']
F9 sv 01,text**F10**

or by writing it directly in Show Codes View, between command brackets:

«md+bo»
«sv01,text»

You can enter them in upper or lower case, e.g.,

«sx01,«pv56»»

or

«SX01,«PV66»»

They will be converted to uppercase the first time you run the program.

2**. Functions**

Searching for Function Codes

Strike the Pfunc key twice (this puts 'SE/FN [square dot] on the command line, with the cursor beside the dot), then type the two characters of the code you're looking for and strike F10.

Entering function codes in programs:

Functions can be entered in programs by executing

F9 pfun xx F10 ('pfun' and 'pfunc' are variant forms of the same command)

or by striking the pfunc key (**Ctrl+;**). If you are in Codes view, the functions will appear as black rectangles with white letters, with a space after the rectangle. The space is part of the code; you cannot delete it.

Entering function codes in keyboard tables:

Functions can be entered in keyboard tables as two-letter sequences *not* separated by commas, e.g.:

##=cp,rd [copy define; erase (rubout) define]

Optionally, you can enter them without commas:

##=cprd

Executing functions from the command line

They can be executed from the command line with:

F9 func xx**F10** (where xx is the function code, e.g. 'func as' will go to the adjacent open window, if any)

You can dedicate a keyboard-table key to this sequence:

##=bc,f,u,n,c, ,

The you have only to type the two-letter function code and strike F10 to execute the function.

Or you can save this program in your XPL subdirectory as FUNC.RUN—copy everything from the first ';*;' to the second, inclusive. (Change to Codes view to see the whole program.)

```
;*; Program puts 'func + space' on command line, reads from the keyboard the 2 characters
you type for the function you want to test, and executes.
BC func XC ;*;

Then dedicate a keyboard-table key to this sequence:
    ##=bx,r,u,n, ,f,u,n,c,.,r,u,n,q2
This puts the func command on the command line and executes it as soon as you type the
two-letter sequence.
```

3. **Immediate commands**
Immediate commands perform immediate actions on text/files/directories. You can execute them from the command line - for instance,

F9 se /text/ F10
F9 run apost.run F10

They can be run as, or as part of, programs - where they are executed with the codes **BC...XC** or **BX...Q2**, e.g.:

BX run apost.run**Q2**

In keyboard files they are executed by bc...xc or bx...q2. Each character of the command must be followed by a comma. Eg:

bx,r,u,n, ,c,:,\,n,b,w,i,n,\,x,p,l,\,a,p,o,s,t,.,r,u,n,q2

Note the space between the command (run) and the argument, and note that the functions (bx and q2) are not divided up by commas.

In programs they are executed by functions **BC...XC** or **BX...Q2**.

4. **Operators**
Operators are used mainly in programs. They work within expressions. E.g.:

«sx02,@upr(«is01»)»

uppercases the text saved in phrase 01. They cannot be entered from the command line, only directly in the program in Codes view.

5. **Defaults**
Permanent defaults are stored in \NBWIN\USERS\DEFAULT\NB.DFL. Many of them can be changed permanently or temporarily, either through the Tools, Preferences menu, or directly in NB.DFL. You should edit NB.DFL in Show Codes view. Back it up before editing.

In NB.DFL the form is:

DF xx=# (where xx is a two- or three- letter code, and numbers or letters follow the equals sign).

You can change defaults for an NB session or part of one by issuing the command:

F9 d xx=# F10

For instance,

F9 d dt=0 F10

causes all files called after the command is issued to be shown in Show Codes view.

6. Paragraph markers, (loosely known as CRs)
[Strictly speaking, NB's paragraph marker is a combination of a carriage return character and a line feed character—a CrLf. But since single CRs or LFs are vanishingly rare in Nota Bene, I use CR to stand for CrLf.]

To put a CR into a search string in a program
To enter a paragraph marker as part of a search string to be implemented within a program, use '^R', e.g.:

BXse /text^R/**Q2**.

In a program, to save a CR in a phrase and compare to one saved to another phrase:
Save an ordinary CR with SV. Then save the second one (e.g., one that you've searched for and defined) to another phrase, also with SV. Finally, use IF...IS...IS to compare them, as in this example, where you search for a separator, define it, save it to phrase 04; then save a CR to phrase 05 and compare the two phrases.

BX se /**S**/**Q2 DF CL DF SV** 04**XD** «SV05,
»«IF«IS4»==«IS5»»..then do one thing, otherwise do another...

In a program, to insert a CR in a file
To have a program insert a paragraph marker into the actual text of a file when the program is being run,
—either enter a normal paragraph marker with the Enter key.
[But if you use Enter after a programming string beginning with BC,
it will execute the string. E.g., BC ci /text/more text/↵ executes the change.]
—or do:
«SV02,↵
»**GT** «PV02»

In a keyboard file, to insert a CR in a file:
Use Carriage Return (alone) symbol to enter a regular paragraph marker in text.
Or use the definition on Unshifted 28: 'FF,&X,C,R'.

In a keyboard file, to insert other types of paragraph marker/line ending in files:
Use:
Line Feed
Carriage Return (alone)
Paragraph End (both types)
Alternate Paragraph only (aka Soft CR)
Regular Paragraph only
[These are extracted from Find/replace menu by clicking on Red/Blue button to right of Find box; clicking on each type of CR; and copy/pasting it into this file. You can do this with any of the characters.

7. **Command Brackets**
To insert in a program,
For opening brackets («) use ASCII 174. For closing brackets (») use ASCII 175.
To input them, hold down Ctrl and Shift keys together, and, while keeping them pressed down, type the numerals 174 or 175.

To search for the key definitions of command brackets in a keyboard file:
Search for ASCII 174/175, input as above.

To search for command brackets in a file (from cmd line or Find/Replace dialog):
Just enter them using the command bracket keys, **Ctrl+,/Ctrl+.**

In a program, to search for command brackets:
If you enter opening command brackets in a program in a search string, without matching closing command brackets, you will get an error dialog every time you open the program in Page Layout View. You can either ignore this or use ASCII 174 instead.
You can also insert command brackets in programs by putting the string «sv01,E»«sv02,F» [E/F=ASCII 174/175] at the top of your program. Then «pv01» will insert an opening command bracket, and «pv02» a closing one. Note that if you want the «»s in the text, you need to include a GT function, or else the string may appear on the command line (depending on what is going on at that point in the program).

8. **Tabs**
In keyboard table or programs:
simply insert a tab character with Unshifted Tab.

9. **Tilde**
Tilde for command line searches or to input long file name in truncated form:
To reproduce, hold down Ctrl and Shift keys together, and, while keeping them pressed down type the numerals 126.

Tilde over letter in text
F9 func MF [go to text, type letter. Press F10, then numerals 7461. Tilde appears over 'n'.
Cursor must be in file when function is executed, otherwise character appears on command line.

Programming: Sample Programs

These sample programs were written for NB 4.5 DOS. In NB for Windows most of them will not work as written, principally because they use function **OV**, which is inoperative in NBWin. However, most of the code is still valid, and the comments give excellent examples of programming strategy. I recommend them for study. I have crossed out invalid parts.
The programs in sections 7, 8 and 10 are still valid in their entirety.

The following sample programs first show each program approximately as it would look on screen, except that the line breaks are arbitrary in these diagrams: the actual programs would be continuous unless you use a ;*; string to break them on screen. In each case the program is followed by an explanatory analysis of it. Where the usage for a program is given as

USAGE: **run <program>**

it is always (except where the program is seldom used) more efficient and economical to load the program on a key-combination, or to use one of the other methods of running a program described in Chapter 7. *[Action line=command line. Expanded mode= Show Codes View. Normal Mode=Page Layout View.]*

---

1. Program closing all windows

The program closes and clears all windows.
USAGE: **run <program>**

«sx11,«vaEP»»«if«pv11»>0»**BC** d EP=0**XC** «ei»«sx40,«va$wn»»«lbAbandon»**BC** ~~**OV** x~~ab**XC** **BC NX** «sx41,«va$wn»»«if«pv40»==«pv41»»«glRS1»
«ei»«glAbandon»«lbRS1»**BC** ~~**OV** x~~**OV** ε1 ~~**BC RSXC BC**~~ «if@not(«er»)»«glRS1»
«ei»**BC** «sx40,«va$wn»»~~**OV** ε0~~ **BC NX** «sx41,«va$wn»»«if«pv40»==«pv41»»«glRS2»
«ei»«glRS1»«lbRS2»«sx44,«va$fs»»«if«pv44»==0»**#1 DX AS** ~~**BC RSXC**~~ «ei»
«lbEND_abcl»**BC** d EP=«pv11»**XC BC** ~~**OV ODO**~~ «pr All windows abandoned and closed» «ex1»

«sx11,«vaEP»»	—saves value of Error Prompt) as phrase 11
«if«pv11»>0»**BC** d EP=0**XC** «ei»	—if value of EP is greater than 0, sets default value of EP=0 (Erase without prompting); endif
«sx40,«va$wn»»	—saves the number of the active window as phrase 40
«lbAbandon»	—label Abandon
BC ~~**OV** x~~ab**XC**	—clears action line, ~~freezes display for all windows~~, abandons present file
BC NX	—clears action line, and goes to next open window
«sx41,«va$wn»»	—saves the number of that window as phrase 41
«if«pv40»==«pv41»»«glRS1»«ei»	—if the number of the window is the same as that saved on phrase 40, goes to label RS1; endif
«glAbandon»	—otherwise goes to label Abandon (and repeats previous operation)
«lbRS1»	—label RS1
BC ~~**OV** x~~	~~—clears action line, freezes display for all windows suppresses error messages and beeps clears current screen~~

BC «if@not(«er»)»«glRS1»«ei» —clears action line; if no error, goes to label RS1; endif
BC «sx40,«va$wn»» —otherwise saves the number of the active window as phrase 40
~~**OV** c0~~ ~~—reactivates error messages and beeps~~
BC NX —clears action line, and goes to next open window
«sx41,«va$wn»» —saves the number of that window as phrase 41
«if«pv40»==«pv41»»«glRS2»«ei» —if the number of the window is the same as that saved on phrase 40, goes to label RS2; endif
«glRS1» —otherwise goes to label RS1
«lbRS2» —label RS2
«sx44,«va$fs»» —saves as phrase 44 the information on which windows contain files
«if«pv44»==0»**#1 DX AS** ~~**BC RSXC**~~ «ei» —if value of phrase 44 is 0 (i.e., if no windows contain files), switches to Window 1,freezes screen display, switches to adjacent window,~~ clears screen;~~ endif
«lbEND_abcl» —label End_abcl
BC d EP=«pv11»**XC** —restores original **BC** D EP=0**XC** value
~~**BC OV** O~~ ~~—unfreezes screen display for all windowS~~
DO —unfreezes current window
«pr All windows abandoned and closed»—reports completion of operation
«ex1» —ends program

2. **Program closing all windows but current one**

The program checks each window in turn, clearing the file (if there is one) in it, and closing the window, leaving only the current window with its file in it.
USAGE: **run <program>**

«sx98,«va$wn»»~~**OV** x~~«lbClear»**NX** «sx99,«va$wn»»«if«is98»==«is99»»~~**OV** o~~**GT** «prAll Windows but this Cleared / Closed »«ex»«ei»~~**OV** a~~**BC** rs **XC** «glClear»«ex»

«sx98,«va$wn»» —saves the number of the current window as phrase 98
~~**OV** X~~ ~~—turns off display for all windows~~
«lbClear» —label Clear
NX —goes to next open window
«sx99,«va$wn»» —saves the number of the current window as phrase 99
«if«is98»==«is99»»**BC** d ep=«pv02»**XC** ~~**OV** o~~**GT** «prAll Windows but this Cleared / Closed»«ex»«ei»
—if the current window is the same as that saved to phrase 98, restores original EP setting; ~~turns on display for all windows,~~ goes to text, displays message, and ends program; endif
~~**OV** a~~ —otherwise,i.e., if the condition is not satisfied, abandons file on screen
~~**BC** rs **XC**~~ ~~—closes window~~
«glClear» —goes to label Clear and continues closing windows
«ex» —ends program

3. **Program comparing screen file with disk file**

This program compares the current screen file with the file (if any) on disk, to check if changes have been made since it was last saved. It does it by calling to screen in an adjacent window the disk-copy of the file (if there is one), and looking for the first difference between the two files. If it finds none, it reports No Change in the screen file; if it finds any change, it invites you to save the screen file to disk.
USAGE: **run <program>**

BC ~~OV~~ ~~X~~«sx76,«va$fp»»«sx77,«cp»»**~~OV nw~~**«glCA-F»
«lbCA-F»**BC** ca «pv76»**XC XP** «if«er»»**~~BC rsXC~~ OV** o**BC** «pv76» Not on Disk!**GT**
«ex»«ei»«glCO»
«lbCO»**AS XP TF FD AS BF** «sx80,«cp»»**~~OV~~ ~~a BC rsXC~~** «sx78,«cp»»**BF**
«sx79,«cp»»«if((«is78»)<(«is79»)!(«is78»)<>(«is80»))»**BC** jmp «pv77»**XC BC WG-OV** o**GT**
«pr File Changed; Save?»«ex»«ei»**BC** jmp «pv77»**XC BC WG ~~OV~~ ~~o~~GT** «pr No Change in File»«ex»

BC **~~OV~~ ~~X~~**	~~—turns off display. This command differs from **DX** in that it applies to all windows, not just to the one open at the time the function was executed. Its complement,**OV** O, turning display on, again like **DO**, comes just before the end of the program.~~
«sx76,«va$fp»»	—saves full specification of file (drive/path and name) as phrase 76
«sx77,«cp»»	—saves current cursor position as phrase 77
~~OV nw~~	~~—opens next empty window~~
«glCA-F»	—jumps to label CA-F; this is a device to break up the appearance of the program on screen, to make reading of it easier. *[Still valid, but NBWIN's ;*; string is easier and more elegant.]*
«lbCA-F»	*—label CA-F*
BC ca «pv76»**XC**	—calls the file-on-disk by the specification stored as phrase 76
XP	—changes to Expanded Mode
«if«er»»**~~BC rsXC OV~~o****BC** «pv76» Not on Disk!**GT** «ex»«ei»	—if there is an error (that file is not on disk), ~~it clears that window, turns on display again,~~ reports that the file is not on disk, and exits; end-if
«glCO»	—if no error, goes to label CO
«lbCO»	—label CO
AS	—switches to the other window (the screen-file)
XP	—switches to Expanded Mode
TF	—goes to top of file
FD	—finds the first difference between the two files
AS BF	—switches back to file-on-disk and goes to bottom of file
«sx80,«cp»»	—records value of cursor position there
~~OV~~ ~~a~~	~~—removes the file-on-disk from screen~~
~~BC rsXC~~	~~—closes window~~

«sx78,«cp»»	—records as phrase 78 value of cursor position at that point in screen file (the first point of difference previously found between the two files)
BF	—goes to end of file
«sx79,«cp»»	—records as phrase 79 value of cursor position at end of file
«if((«is78»)<(«is79»)	—if the first point of difference between the two files occurs before the end of the screen-file
!(«is78»)<>(«is80»))»	—or if the first point of difference is not the same as the cursor position at the end of the file-on-disk
BC jmp «pv77»**XC**	—then returns to original cursor position
BC WG OV o**GT**	—clears action line, returns to Normal Mode, ~~turns display back on,~~ moves cursor to text area
«pr File Changed; Save?»«ex»«ei»	—reports that there has been a change in file since last saved, and exits program; end-if
BC jmp «pv77»**XC**	—otherwise returns to original position
BC WG ~~OV~~ o **GT**	—clears action line, returns to Normal Mode, ~~turns display back on~~, moves cursor to text area
«pr No Change in File»«ex»	—reports no change in file since last saved, and exits program

4. **Program comparing screen file with disk file**

This program does the same job as the previous one, but is much shorter, ~~because it makes use of one of the many new overlays now available in~~ **~~Nota Bene~~** ~~4. In this case the overlay is~~ **~~OV~~** ~~ft, which performs just one function: it writes 1 into phrase 99 if the current file has changed since it was last saved; otherwise it writes 0. In this program, if the value is 1, the user is invited to save the file to disk; if the value is 0, the program terminates without saving the file. The brevity of the program illustrates the economy in program writing that the new overlays make possible~~

USAGE: **run <program>**

«sv01,Y»~~**OV ft**~~«if«pv99»==1»**BC** File changed. Save? (Y/N)«glSave»»«ei»**BC** File not changed since last saved«ex»«lbSave»«sx02,«rk»»**BC** «if«is02»==«is01»»~~**OV sa**~~«prFile saved»«ex»«ei»«prFile not saved»«ex»

«sv01,Y»	—saves Y (for Yes) to Phrase 01
~~**OV ft**~~	~~—executes the overlay function **OV ft**~~
«if«pv99»==1»**BC** File changed. Save? (Y/N)«glSave»»«ei»	—if the value of Phrase 99 is 1, declares that the file has changed since last saved, and asks whether it is to be saved now. Answer to be Y or N. Goes to label Save; endif
BC File not changed since last saved«ex»	—if the value of Phrase 99 is not 1, declares that file has not changed, and terminates program
«lbSave»	—label Save

«sx02,«rk»» —reads character typed at keyboard in response to the question Save? asked above, and saves it (uppercased) to Phrase 02

BC —clears command line

«if«is02»==«is01»»~~OV s~~a«prFile saved»«ex»«ei»
—if character typed at keyboard is same as that saved in 01 (i.e., Y), ~~current file is saved to disk~~, a message that it has been saved is displayed, and program terminates; endif

«prFile not saved»«ex» —otherwise (if key typed was not Y) displays message that file has not been saved, and program terminates.

5.**Programs Using Incremental Counter**

The program illustrates the use of an incremental counter for finding how many occurrences of a specified word there are in a file.
USAGE: **run <program>,<word>**

TF «sv25,0»**DX** ~~OV c~~1«lbW»**BC** se **WS** «pv00»**WS XC** «if«er»»**TF OV** ε2 **BC** «pv25» occurrence(s) of the word '«pv00»'«ex»«ei»«sx25,«pv25»+1»«glW»

TF —goes to top of file

«sv25,0» —saves 0 as phrase 25

DX —freezes screen display

~~**OV** c1~~ —~~suppresses error messages and beeps~~

«lbW» —inserts label W

BC se **WS** «pv00»**WS XC** —searches for the word entered as argument on the action line

«if«er»»**TF** ~~**OV** c2~~**BC** «pv25» occurrence(s) of the word '«pv00»'«ex»«ei»
—if error (word not found), goes to top of file, ~~unfreeezes screen display~~, turns off error suppression, reports the number of occurrences of the word in the file, and ends program; endif

«sx25,«pv25»+1» —otherwise, increments by 1 the number (initially 0) saved as phrase 25

«glW» —goes to label W (and resumes search for next occurrence of the word)

6.**A more elaborate version of previous program.** It reports the frequency either of a word or of a string of characters, prompting you to choose which it is that you want.
USAGE: **run <program>,<word>**

TF «sv25,0»«sv01,W»«prStrike W for Word, S for String»«sx02,«rk»»**DX** ~~**OV** c1~~
«if@not(«is02»==«is01»)»«glS»
«ei»«lbW»**BC** se **WS** «pv00»**WS XC** «if«er»»**TF** ~~**OV** c2~~**BC** «pv25» occurrence(s) of the word '«pv00»'«ex»«ei»«sx25,«pv25»+1»«glW»

«lbS»**BC** se «pv00» **XC** «if«er»»**TF** ~~**OV** c2~~**BC** «pv25» occurrence(s) of the string '«pv00»'«ex»«ei»«sx25,«pv25»+1»«glS»

TF	—goes to top of file
«sv25,0»	—saves 0 as phrase 25
«sv01,W»	—saves W as phrase 01
«prStrike W for Word, S for String»	—prompts to enter W or S
«sx02,«rk»»	—saves (in uppercase) key struck
DX	—freezes screen display
~~**OV** c1~~	~~— suppresses error messages and beeps~~
«if@not(«is02»==«is01»)»«glS»«ei»	—if key struck is not W (i.e., if it is S), goes to label S; endif
«lbW»	—labelW
BC se **WS** «pv00»**WS XC**	—otherwise, searches for the word entered as argument on action line
«if«er»»**TF** ~~**OV** c2~~**BC** «pv25» occurrence(s) of the word '«pv00»'«ex»«ei»	—if error (word not found), goes to top of file, ~~unfreezes screen display~~, turns off error suppression, reports number of occurrences of the word, and ends program; endif
«sx25,«pv25»+1»	—otherwise, increments by 1 the number (initially 0) saved as phrase 25
«glW»	—goes to label W (and resumes search for next occurrence of the word)
«lbS»	—label S (see «glS» above)
BC se «pv00» **XC**	—searches for string saved as argument on action line
«if«er»»**TF** ~~**OV** c2~~**BC** «pv25» occurrence(s) of the string '«pv00»'«ex»«ei»	—if error (string not found), goes to top of file, ~~unfreezes screen display~~, turns off error suppression, reports number of occurrences of the string, and ends program: endif
«sx25,«pv25»+1»	—otherwise, increments by 1 the number (initially 0) saved as phrase 25
«glS»	—goes to label S (and resumes search for next occurrence of the string)

7. **Program using parsing to execute a command a specified number of times**

The program uses **xs**. It executes the specified command the specified number of times, e.g., find the 24th occurrence of a specified word, or print 10 copies of this file.
USAGE: **run <program>,#*<command>**

E.g. to find the 15th footnote in the screen file; the screen file must first be changed into Expanded mode.
run <program>,15*sea FN

«sv01,*»«xs00,01,02,03,04»«sx05,0»«lbexe»**BC** «pv04»**XC**
«if«er»»«ex»«ei»«sx05,«pv05»+1»«if«pv05»==«pv02»»«ex»«ei»«glexe»

«sv01,*»	—saves asterisk as phrase 01. The asterisk will be the parsing operator in the parsing operation
«xs00,01,02,03,04»	—parses the argument on the command line, i.e., the string '15*sea FN ' The parsing saves '15*sea FN ' as phrase 00,'15' as phrase 02, '*' as phrase 03, and 'sea FN ' as phrase 04
«sx05,0»	—saves 0 as phrase 05
«lbexe»	—label exe
BC «pv04»**XC**	—executes the command **sea FN**
«if«er»»«ex»«ei»	—if error (no more notes left), ends program; endif
«sx05,«pv05»+1»	—otherwise, increments by 1 the number (initially 0) saved as phrase 05
«if«pv05»==«pv02»»«ex»«ei»	—if phrase 05 equals phrase 02, i.e., if it is 15, ends program; endif
«glexe»	—otherwise (i.e., if neither of preceding conditionals is true) goes to label exe, and searches for the next footnote

8. **Program using subroutine**

The following is an example of an 'su' in action. The «su#,...», say «su105,...», can be stored in memory with
run <name of 'su's file>. Then it can be embedded in any program with 'pv105', where it will be executed. This can be illustrated with a simple «rc» loop.

This is a loop that enters into the text each stroke made from the keyboard until * is struck, which leads to exit from program:

```
«sv20,*»«lbChain»«sx25,«rc»»«if«is25»==«is20»»«ex»ei»
«pv25»«glChain»
```

It can be saved in an 'su', like this:

```
«su105,«sv20,*»«lbChain»«sx25,«rc»»«if«is25» ==«is20»»«ex»«ei»«pv25»«glChain»
»
```

Note: there must be a paragraph marker immediately before the «su»'s closing command bracket.

This subroutine can be embedded at the start of a program in which you are going to want interaction from the keyboard. At any point in the program where you want a pause for keyboard entry embed the command «pv105». The program will then pause for you to make as many keystroke entries as you want; when you have finished strike *, and the program will resume. Note that to break out of the subroutine and return to the main program «ex» must be used; if «ex1» were used instead, the entire program would be ended.

Making any such subroutine into a program of its own (called, say INTER.RUN) and running it (say with the line:

BC run inter.run

in NBSTART.INT) stores it in memory. That means that the subroutine itself does not have to be embedded in any program in which you want to use it. All you have to do is to embed the «pv105» in the place(s) where it is needed to cause the program to pause for keyboard entries.

9. A complete (and possible lengthy) program can be stored as a subroutine on an extended phrase, and subsequently executed from a regular phrase key, without taking up more than a fragment of the limited memory available for the storage of regular phrases. If a program of any length is stored in a subroutine with «su105,<program>», then with Alt-F3 «pv105» can be put on a phrase key and saved as an XPL program. Alternatively it can be loaded, by creating a program named, say, PV105, and consisting just of the one command «pv105»; that second program can then be loaded on an Alt-key, say S, with the command **ldpm PV105,s**. Then any time that Alt-S is struck the full program will be executed. This makes the running of a program much faster, because no access to disk is involved; on the other hand, some general memory is kept locked up by the storing of the subroutine/program on an extended phrase. The user must decide in any particular case which is the most efficient and economical method to follow.

10. **Load whole phrase library on one key**

USAGE: **run <program>**+letter/numeral of phrase key wanted.
In this case the only efficient way to run the program is by assigning it to a key-combination. The program does exactly the same job as using the ALT key+a letter/numeral to insert a phrase or command from a phrase library; and it does the job no better. The advantage of using it instead of the Alt+key method is that it releases the 35 keys in the ALT table of NB.KBD that are by default defined as
@x ;where x is either a letter or a numeral. As those keys are no longer needed as phrase keys, they all become available for redefinition by the user. The program makes use of the fact that all phrases are entered by the function codes **@A-@Z**, and **@1-@9**. It does not affect the use of Alt+F3 for displaying and editing phrase keys.

Closely similar programs can be written using, instead of 'func @':
(1) 'func &' for running programs loaded on ampersand phrases;
(2) 'func #' for moving to a specific window.

NB: In NBWin this job can be done better by defining a key as: ##=SG. See p. 157.

BC func @«prEnter A-Z or 1-9; <Esc> to cancel»«sx01,«rc»»«pv01»**GT XC** «ex»

BC func @	—clears action line, enters beginning of function command, and waits for you to enter the correct alphanumerical character
«prEnter A-Z or 1-9; <Esc> to cancel»	—prompts for character
«sx01,«rc»»«pv01»**GT XC**	—saves character entered, inserts it on action line, goes to text area and executes function command
«ex»	—exit program

Programming: Writing Programs

1. **Planning** There can be no hard and fast 'how to' rules, or infallible recipes, about methods for writing programs. There are too many factors involved: how experienced you are; how good you are at keeping in your head at one time a number of phrase values, of 'if's, of 'go to labels', etc.; how long and intricate the program itself is going to have to be; and so on. The nearest thing to a rule perhaps is that, except in the case of a very short program, the way to start writing a program is not to start by writing it: it is better to start by thinking about it, jotting down notes on paper, and planning the general flow of the program, so far as you can, before getting down to the details.

If the program is going to be short, proceeding in a single linear sequence, with no branches or loops, then you can go straight at it. For example, suppose you required a program that would tell you, whenever you wanted, what the present location of the cursor is in your current file, you could write that out without previous preparation:

«sx01,«cp»»**BC** Cursor at «pv01» bytes from Top of File«ex»

That is easy, because it is so short, and because the «ex», marking the end of the program, comes as the final entry. But that does not often happen: the program may have more than one «ex» in it; and they may be dotted over the program, anywhere but as the final entry.

2. **Building a Program** When you do start creating a program, remember that it is not necessary, and often not advisable, to compose it in exactly the order which it is going to bear when finished: it is usually better to build it up from its central aim, inserting the necessary additions stage by stage. As an illustration, take the case of a program that you might want to create for embedding paragraph markers at the end of every line in a file. When you are writing a file, such as a letter, to send by email, you may want every line to have a carriage return at the end of it. You can, when writing the file in **Nota Bene**, make a point of hitting the <Enter> key at the end of each line; but that is not easy to remember, and gives you extra work if you make any revisions in the file. It is much easier to write the file in the ordinary **Nota Bene** way, and then put the paragraph markers in afterwards; it is easier still if you have created a program to do that for you.

The first thing you will need in the program are formatting codes to set the Point size and page width settings. SZ11PT and PW70DI would be generally suitable, so the program can start by going to the Top of the File, and embedding those codes:

TF «SZ11PT»«PW70DI»

The program will run better if you have it save those codes to a phrase, and then insert the phrase at the right place.

«sv07,«SZ11PT»«PW70DI»»**TF** «gt07»

You also need to ensure that the file is in Page Layout View, by inserting a **WZ** function:

«sv07,«SZ11PT»«PW70DI»»**WZ TF** «gt07»

Then you want the program to go to the end of the line, and replace the space that is there with a paragraph marker, as in:

LE CR BD ↵

That goes to the end of the line (**LE**), moves the cursor one place right (**CR**), backdeletes the space, and inserts a paragraph marker (↵). To insert a paragraph marker *[also confusingly known as a carriage return or CR]* into a program to be put into a file (i.e., not to be used as part of a search string), you simply strike the <Enter> key.

The program, so far, looks like this:

«sv07,«SZ11PT»«PW70DI»»**WZ TF** «gt07»**LE CR BD** ↵

At this stage the program takes care of only one line. You have to elaborate it so that it will do two more things:

(i) repeat the process for succeeding lines in the file;

(ii) recognize when it reaches the end of the file, so that it does not try endlessly to continue repeating (i)'s process of replacing end-of-line spaces with markers. (i) is taken care of by adding a «gl...» at the end of the existing program, and adding a matching «lb...» to the program in the right place to repeat for succeeding lines the operation of replacing space with marker: the place for the «lb...» is immediately before **LE**. The program would now become this:

«sv07,«SZ11PT»«PW70DI»»**WZ TF** «gt07»«lbLE»**LE CR BD** ↵

«glLE»

Now the program will work right through your file replacing end-of-line spaces with paragraph markers. But it needs to be given some way (ii) of recognizing when there are no more lines left to alter.

(ii) can be achieved by using the fact that when the cursor reaches the end of a file, the instruction to move it one place to the right has no effect. So at the end of the file this code, saving the cursor position (cp) to phrase 05, moving one place to the right and saving the cp again::

«sx05,«cp»»**CR** «sx06,«cp»»

will leave the values of «is05» and «is06» identical. The program will take the values of the two «cp»s, and be told to do one thing if they are identical, a different thing if they are not. After «sx05,«cp»»**CR** «sx06,«cp»» instructions must be added about ending the program if the two values are identical. The instructions would be: to go back to the top of the file and remove the two formatting codes that had been inserted there at the outset, and to exit the program:

«if«is05»==«is06»»**TF RC RC RC** «ex»«ei»

This new section of code must be inserted in the existing program immediately after **LE CR** , making the program now read:

«sv07,«SZ11PT»«PW70DI»»**WZ TF** «gt07»«lbLE»**LE CR** «sx05,«cp»»**CR**

«sx06,«cp»»«if«is05»==«is06»»**TF RC RC** «ex»«ei»**CL BD** ↵

«glLE»

It is necessary to insert a **CL** immediately before the **BD**, to offset the **CR** between the two calls saving the «cp»s to their respective phrases.

The program is now complete, but it can be improved by

(i) freezing the video while it is being executed; and

(ii) inserting a message at the end to report that its execution has been completed.

(i) is achieved by inserting **DX** to turn the video off and **DO** to turn it on again; and (ii) by inserting a prompt message «prConversion complete» immediately before the «ex». It would also be helpful to the user if you inserted a «prWorking...» code near the start of the program, which will be displayed throughout the running of the program *[hardly necessary in NBWin on a*

reasonably fast computer—one running Win XP]; but the **DX** needs to be moved so that the prompt message will not be made invisible; the eventual result should be:

Program to add line ends to emails

«sv07,«SZ11PT»«PW70DI»»**WZ** «prWorking...»**DX TF** «gt07»«lbLE»**LE CR**
«sx05,«cp»»**CR** «sx06,«cp»»«if«is05»==«is06»»**TF RC RC DO** «prConversion complete»«ex»«ei»**CL BD** ↵
«glLE»

If you use smart quotes (curly quotes), you may want to change them to straight quotes for sending in emails. This bit of code does it:

BX ci /’/'/**Q2 BX** ci /‘/'/**Q2**

You can insert it before the first **TF** code.But you must move the **DX** code to after the change string, which will not work with **DX** before it. Suppress the ‘Cannot find item’ error message by putting **BC** es 1**XC** at the beginning of the program.

BC es 1**XC** «sv07,«SZ11PT»«PW70DI»»**WZ** «prWorking...»**TF BX** ci /’/'/**Q2 BX** ci /‘/'/**Q2 DX** «gt07»«lbLE»**LE CR** «sx05,«cp»»**CR** «sx06,«cp»»«if«is05»==«is06»»**TF RC RC RC DO** «prConversion complete»«ex»«ei»**CL BD** ↵
«glLE»

3.**Comments: Breaking Programs into Lines** Programs in other programming languages are made up of distinct lines of code, sometimes with blocks of code indented from preceding and succeeding lines, for the sake of clarity. An XPL program, on the other hand, is one continuous line from beginning to end (like a single paragraph in **Nota Bene** text); and that can make it difficult to pick out the trees from the wood. Even a short program like the one above would be easier to read, if set out like this:

«sv07,«SZ11PT»«PW70DI»»**WZ** «prWorking...»**DX TF** «gt07»

«lbLE»**LE CR** «sx05,«cp»»**CR** «sx06,«cp»»

«if«is05»==«is06»»**TF RC RC RC DO** «prConversion complete»«ex»«ei»

CL BD «glLE»

To do this, use the commenting string ‘;*;’:

«sv07,«SZ11PT»«PW70DI»»«prWorking...»**DX TF** «gt07»;*;
;*;
«lbLE»**LE CR** «sx05,«cp»»**CR** «sx06,«cp»»«gl2»;*;
;*;
«lb2»«if«is05»==«is06»»**TF RC RC RC DO** «prConversion complete»«ex»«ei»«gl3»;*;
;*;
«lb3»**«CL BD** ↵
«glLE»

Any line or para that begins with ;*; is a comment. It can be many lines long; nothing will be executed until the line after the first paragraph mark that follows the ;*;.

You can insert nearly-blank lines by putting a commenting string on a line on its own.

You can break a line of code anywhere with a commenting string (except in the middle of an

expression, «GT;*;
07» will not do).
When writing a new program it is a good idea to use a lot of these commenting separators. It helps to keep clear the various elements of the program; and it helps to keep you on track while doing the writing. It is also a good idea to include comments describing what each element is doing. Once the writing has been done, and the testing successfully completed, you can remove as many of them as seems suitable. But there is much to be said for keeping them, and, in particular, in keeping your notes on what is happening in that section of the program. That can be a help when you are revising or expanding the program, and it can be very useful to any other user with whom you share the program.

4. **Embedding Codes in Programs** In Chapter 5 the two ways of embedding function codes (Recording Mode, and the PFUNC Command) were described; and the question arises whether one is to be preferred to the other. The answer is that each has its pros and cons, and that you should exercise judgement in deciding which to use at a given time. The advantage of Recording Mode is that you can enter program functions into a file without knowing what the codes for the functions are. Striking **Ctrl+Shift+F10** will put the **CC** code into the file; striking unshifted **F10** will input **XC**. But with each new version of NB for Windows there are fewer keys that hold simple two-character functions that can be entered in a program like this, and more and more keys which, if pressed in Recording Mode, put code like this into the program: **&X BC** —this is the definition on unshifted **F9**; or **[U &X BDU]** —this is on the **Backspace** key. These codes will not work; you will have to hunt for the code you want in Chapter 8 (or the shorter function list in Chapter 2). Recording Mode has the further disadvantage that if you hit the wrong key, you may reach for the backdelete key—which will input **[U &X BDU]** rather than backdeleting your mistake. It's easy to lose track and have to start from scratch.
So it is almost always better to use **pfunc**. The disadvantage of **pfunc** is that, in order to embed a function code, you need to know what the code is (which you do not with the Recording Mode method); or, if you do not know it, you have to look it up in the lists of codes in Chapter 8, or the table of **Keyboard Functions** inChapter 2.
With this method, the function is embedded as soon as you strike the **Ctrl+;** combination and type the two-letter mnemonic.
As you become more familiar with the vocabulary of function codes, you will find the **pfunc** method the more economical. And you can, in fact, have the best of both worlds by always writing and editing a program with **pfunc**, and switching to Recording mode just for those functions the mnemonic letters for which you do not know, or do not remember. Even if it inputs non-functional codes like **&X BC** or **[U &X BDU]**, they often contain a reminder of what the the two- character code is—**BC** and **BD** in these instances.

Program to make PFUNC embed codes in file
In fact there is a simpler and more economical way of getting **pfunc** to embed function codes in a file, which does not involve distracting your attention by moving up to the command line and entering the two mnemonic characters there. Write for yourself this very short program:

GT YD DF CL CL DF «sv01»**RD BC** pfunc «pv01»**XC GT** «ex»

Save the program as, say, PFUNC.RUN, and load it on an Alt-key, for example on **Alt-F**. Then, whenever you want to embed a function code into a file, type into the file the two letters of the mnemonic, and strike **Alt-F**. That will replace the two letters you have just typed with the corresponding function code. For example, typing the two letters bc and striking **Alt-F** will replace the bc with **BC** .

Replacement Dictionary Another, even more economical method, which is to be recommended on other grounds (see next section), is to create a personal abbreviation dictionary (called, say, PROGRAM.SPL or XPL.SPL), and, with it loaded, use Automatic replacement. If the dictionary contains the line

bc **BC**

then any time you type bc and hit the **Ctrl** key, the two letters will automatically be replaced by the **BC** function code. *[NB: Use the Ctrl key, not the space bar, which would insert an extra space into your program.]*
You can keep XPL.SPL open on one side of your NB screen while programming to remind yourself of what abbreviations to use.

Phrase library You can also save codes and program segments to a phrase library, perhaps named PROGRAM.LIB or XPL.LIB, though this has two disadvantages: you cannot keep the library open as an *aide-memoire*; and you are limited to 36 phrases (A-Z and 1-0).

5. **Embedding Program Calls in Programs** This cannot be done by the use of Recording Mode; calls must be entered as if they were text. All program calls begin and end with a command bracket, and consequently in Page Layout View appear as undifferentiated codes; Codes View should therefore always be used when writing programs. Opening and closing command brackets can then be inserted with **Ctrl+<** and **Ctrl+>** [keys 51 and 52] respectively; if you try to strike the former key-combination when in Page Layout View, the opening bracket will be entered, but you will get an error message, and will not be able to continue until you delete the bracket .

Replacement Dictionary / Phrase Library - With calls as with function codes, it pays to set up an abbreviation spell file and/or phrase library (the same one can be used for both), and to include in it most of the common, and certainly the more complicated, calls. Wherever a call requires a complementary call, e.g. «if» requiring «ei», it is worth putting both of the pair on one abbreviation key, so that both will be entered together, with less risk of your forgetting to supply the complementary one. The second member of the pair must then be moved to its correct place in the program. Abbreviations such as

ife «if»...«ei»
ifr «if«er»»...«ei»
ifx «if»...«ex»«ei»

can save both time and mistakes.
As every «gl..» requires an «lb..», you might think it worth putting both into a single abbreviation. This also applies to commands and codes requiring complements. For example you can freeze the video display during the running of a program by inserting **DX**; but, if you do not insert a **DO** somewhere before the «ex» call (possibly more than one) occurs, then, when you leave the program, although the computer will continue to work, it will look as if it had locked up. The abbreviation

dx **DX DO**

Then there are routines that you may need to use quite often, but do not want to have to recreate each time you want them. For example, sometimes a program will give the user a choice, requiring the response of Y or N to a question. The expansion of 'rc' in the next line will do that *[Note: the 'rc' at the beginning of the line is the abbreviation to be expanded, not part of the program segment.]*

Yes-or-no routine
rc «sv01,Y»«prMessage....Answer Y/N»«sx02,@UPR(«rc»)»«if«is02»==«is01»»do such-and-such«ei»otherwise do so-and-so.

The expansion of 'ks' below provides for every keystroke entered by the user being entered into a file until F10 is struck, whereupon the program ends, or does whatever else you substitute for «ex»

User keystroke routine
ks «sv10,**XC** »«lbRec»«sx51,«rc»»«if«is51»==«is10»»«ex»«ei»«pv51»«glRec»

The compiling of such a dictionary could go on indefinitely, but it is best to limit it toabbreviations and expansions that you will use regularly; otherwise you spend as much time hunting for them in the .SPL file as it would take to type them from scratch. But one further particular one is worth mentioning.

Searching for command brackets
In a number of programs that you write you will have occasion to include search commands of the form **BC** se \...**XC** (or **BX** se \...**Q2**); and sometimes the string to be searched for will include one or other of « and », the two command brackets. To insert them in a search string in a program, hold down **Ctrl+Shift** and, while holding the keys down, strike 174 for opening command brackets or 175 for closing ones.

6. **Setting Defaults** In some programs you may have occasion to set some new defaults, for convenience in the running of the program. But you will not want those defaults to continue to hold after the programs has been run. For example, you want a program that you are writing to work without prompting the user for confirmation before erasing a file; but you want a reversion to normal defaults once the program has finished. The code **BC** d ep=0**XC** ('ep' is Error Prompt) will prevent such prompts during the program. To get back to the normal default the program must first establish what the value of that is, and then at the end restore it. The call
«sx01,«vaep»»
will record the user's current setting for Erase Prompts. Itshould be inserted at, or very close to, the beginning of the program; then insert **BC** d ep=0**XC** to change it to 0 for the running of the program; then, before the program's «ex» call (or before each of them, if there are several «ex»s), insert **BC** d ep=«pv01»**XC** ,to restore the original setting. The corresponding operation should be performed for any other default settings that are made during a program.

7. **Writing for public use** If you are writing a program that is to be, or that may be, used by somebody other than yourself, you should always, if the program uses or assumes a certain default setting, have the program perform operations of the above kind. You cannot assume that other users are using the same defaults as you; if you do, and if they are not, the program, although it works for you, will not work for them.

8. **User Options** Sometimes a program will give the user an option, as in 'Press * to call file; press / to finish'. The program must then specify what is to be done if the first option is chosen, and what if the second is chosen. In the case of the User keystroke routine above, you can enter:

«sv12,*»«sv14,/»

followed later by

«sx11,«rc»»«pv11»«if«is11»==«is12»»do one thing«ei»«if«is11»==«is14»»do something else«ei»

But where, as in this case, there are only two options, it is actually unnecessary to specify the second, because the program will automatically take that, if the first is not chosen. In the above, «sv14,/» and «if«is11»==«is14»» can be omitted:

«sv12,*»«sx11,«rc»»«pv11»«if«is11»==«is12»»do one thing«ei»otherwise do something else

Multiple options need to be dealt with differently. Suppose the program is to give the user choice of one of six options, each marked by one of the letters 'ABCDEF', then it must provide for the specific letter that is chosen, and it must also provide for the case where none of them is chosen. The best way to handle that is to introduce the string operator î, which is used to determine whether one character/string is contained in another string, and, if so, which position in that string it occupies.

In the string ABCDEF A occupies position 0, B occupies 1, and so on. If the string is saved to Phrase 03 with «sv03,ABCDEF», and if the key struck (character entered) by the user is saved to Phrase 01 with «sx01,@UPR(«rc»)», then «sx02,«is01»î«is03»» will record as Phrase 02 the position in ABCDEF of the character entered by the user. The program must then specify what is to be done if the character entered is not one of the letters ABCDEF, which it does with «if«pv02»<0», (if the position that it occupies is less than 0), i.e., if it does not occupy any position in the string ABCDEF. If it does occupy a position in the string, the program must then specify what must be done for each possible position.

There are various ways of doing that, depending on the length of the string and on the rest of the program. But the simplest way in this case is to specify an «if» for each of the six positions; let us suppose it directs the program to one or other of six labels, each bearing the respective letter as its name:

«if«pv02»==0»«glA»«ei»«if«pv02»==1»«glB»«ei»«if«pv02»==2»«glC»«ei», etc. This is a a case where the program would be much easier to read if each «if» was on a line of its own, which can be achieved by ending each line with ;*;.

```
;*;
«if«pv02»==0»«glA»«ei»;*;
«if«pv02»==1»«glB»«ei»;*;
«if«pv02»==2»«glC»«ei»;*;
  and so on
```

9. **Suppressing Video Display** The same consideration applies to suppressing video display. In most completed programs you will want to embed **DX** and **DO** in the right places. They speed up the execution of the program, and they save the screen display from doing a frantic St. Vitus's dance. But it is best not to insert them until the last minute: it is helpful, when trying out the program, to see what is actually happening during its various stages.

10. **Suppressing Error Messages** Some programs will, if undoctored, send error messages when they are being executed, and make the computer beep; that can be distracting and irritating to the user. This regularly happens, for example, if the program involves a 'search' command: when no further instances of the string being looked for can be found, the command produces a beep and a 'Not found' message. This can be avoided by embedding at the start of the program, or near it, an Error Suppression command, **BC** es 1**XC**. *[In NB 4 it was necessary to reactivate bell and error messages with the command,* ***BC*** *es 0****XC****. This is not necessary in NB for Windows.]*. But it is advisable not to insert the command into the program until after it has been written and satisfactorily tested. If you put it in earlier, and if you commit an error in the course of writing the program, you will receive no warning of it. It can sometimes be difficult to spot just where in a program you have entered some wrong code; and the error messages are sometimes so general that they do not pinpoint the error: 'Command entry error', for example, covers a multitude of possible sins. But they are better than nothing; and nothing is what you will get, if you work with Error Suppression activated.

11. **Working Messages** One price that has to be paid for suppressing video display is that during the execution of the program nothing whatever appears to be happening. This can be disconcerting for the user who, if unfamiliar with the program, may start to wonder whether the computer has locked up. For this reason, it is worth including in the program a message to be shown on the prompt line, to reassure the user that the program is running. «prWorking...» is sufficient. Unfortunately, that does not always work: any **BC** that occurs later in the program clears not only the command line, but the prompt line too, and will wipe out the 'Working...' message. There are various ploys that you can use, such as inserting another prompt after the code that wipes out the previous message, like «prStill Working...». But you must remember to combine that with code to turn screen display on *before* the prompt/message, and another to turn it off again *after* the message. Something like **DO** «prStill Working...»**DX** may do the trick; you must be prepared for a certain amount of trial-and-error experimentation.

With any but the shortest program, it is a good idea to include a message reporting that it has done its job. It can be as short as the all-purpose 'Done' that **Nota Bene** commonly uses, but a more detailed message such as 'Conversion completed', 'File saved to C and B' is more informative. The message must be embedded in the program immediately before the «ex» (or «ex»s), and must come after the **DO** code, if there is one. It can be displayed on the command line, with **BC** Conversion completed, or on the prompt line with «prConversion completed», whichever is more convenient.

12. **Comments** For any program but the simplest and shortest it is worth including comments that will help the user to understand and follow the program. Even the author of a program can have difficulty, when he looks at it a month or so after writing it, in making it all out. Comments may occur:

(i) before the beginning of the program;
(ii) in the course of the program;
(iii) after the end of the program;

or any combination of the three. See 'Breaking Programs into Lines', p 90above

13. **Pruning** It often happens that, having written a program, one revises it (possibly more than once), producing one modified version after another. And it can further happen that some no longer needed function codes and/or program calls from an earlier version survive into a later one without actually inhibiting or interfering with the execution of the later version. These vestigial elements do no harm, but they do no good either, and can make the final version harder to read—especially when you come back to it after a long interval, when none of it looks as familiar as it once did. It is always worth checking through the text of a program, if it is the last of several versions, to make sure there is nothing in it that is no longer needed.

14. **Labels** Finally, it cannot be overemphasised that scrupulous care must be taken with labels. A «gl...» call and its corresponding «lb...» call must exactly match each other; if they do not, the first will not find the second, and the program will not run correctly. «glSTART» will not find «lbStart» or «lbstart», only «lbSTART». And secondly, every label in a program must be unique. For example, if there are two occurrences in a program of «lbCall», then «glCall» will always find the first of them, never the second, with the result that the program again will not run correctly.

15. **Naming programs**
You can name programs anything. If you want to run them from the command line, the name should be in 8+3 form (no more than eight characters before the (optional) extension, no more than 3 after). NB users often name their programs with an extension of .RUN, but it's not necessary. When I am writing temporary programs, I save them with a single-letter filename, e.g., R or T, because it is quicker to test a single-letter file than an 8+3 one: 'run t' versus 'run tempfile.run'.

You don't need to type the path name if you are working in the same folder as the program, but you should really keep your programs in a folder named c:\nbwin\xpl. In this case, if you are in another folder, you need to type the full path to the xpl folder—unless you put the xpl folder in your path (Control Panel, System Properties, Advanced, Environment Variables, System Variables. Highlight Path in the window, click Edit, and add c:\nbwin\xpl).

If you have downloaded and installed the XYWWWEB.U2 file, you can add programs to the bottom of the file and run them with your U2 help key.

16. When programs don't work Especially when you are first learning to write programs, it can be helpful to put each piece of code on a line of its own, with a description of what you mean it to do:

```
;*; Save type size and page width in phrase 07.
«sv07,«SZ11PT»«PW70DI»»;*;
;*; Put a Working prompt on the prompt line.
«prWorking...»;*;
;*; Turn off the display
DX ;*;
;*; Go to Top of File
TF ;*;
;*; Put phrase 07 [type size & page width] at top of file.
«gt07»;*;
```

If the program doesn't work, you can put a temporary end-program code—«EX»—at the end of a suitable line, save the program and run it to see whether it works up to that point. For instance, you could put an «EX» after «gt07», so that you can inspect your file in Codes View and check whether your formatting codes are being inserted at the top:

;*; Go to Top of File
«gt07»«EX»;*;

This can be helpful if troubleshooting a long program. Remember to remove the temporary «EX».

Before you first run a program, and after every change, switch briefly to Page Layout View. If you have a command bracket too many, or too few, you'll get a warning dialog (unless you turn it off with Help, Action Tips—but it is best left on). If you simply run the program, it will at best insert the bit of code in your file, e.g., «gt07, and at worst make the program, and maybe NB itself, choke.

It can also be helpful to change into Draft View with Shift F9, then do Shift F10 repeatedly to cycle through the Draft View display options.

17. **Default MB** Default MB is set in Tools, Preferences, Prompts, under Errors, where you have a choice of displaying error messages on the status line (df MB=0) or in message boxes (default MB=1). For running programs, default MB must be set to 0. Otherwise the contents will display in a Windows message box which will persist on the screen until you press Enter or click on "OK". This causes problems with programs that loop repeatedly through PRompt statements.

If you want to have default MB set to 1 most of the time, you should put a line at the top of your programs to change it to 0:

BX d MB=0**Q2**

then change it back just before the final «EX» with

BX d MB=0**Q2**

If the program goes into a loop Sometimes a program you are testing will go into an endless loop, probably repeating one or more error messages on the prompt line. Occasionally one can break out of the loop by pressing the Esc key repeatedly, but this is very seldom possible. You will have to close NB forcibly. To do so, right-click the Windows Taskbar, and click on Task Manager. Click the Processes tab and find NTVDM.EXE in the list—this will be easier if you click on 'Image Name' at the top of the list to sort the processes alphabetically.
Click NTVDM.EXE to highlight it, then click 'End Process'. You will be warned of possible dire results and asked if you really mean it. Click Yes—nothing bad will happen.
It is not enough to highlight NBEDITOR.EXE and then click 'End Process'. Likewise, it is not enough to highlight Nota Bene in the Applications tab (if it is visible there) and click 'End Task'. You need to close NTVDM.EXE. Otherwise, if you try to run Nota Bene again in the same Windows session, it may well not open.

See the end of Chapter 4 for a list of some of the most common error messages.

Programming: Running Programs

There are many different ways of running **Nota Bene**'s XPL programs; and no one way is always to be preferred to any of the others. There are a number of different factors that you should take into account when deciding which method to use for a specific program: the frequency with which you are likely to use it; the availability of regular phrase keys, and of unused keys in your keyboard file; and so on. In this chapter various methods will be described. You can experiment with them all, and decide which suits you best for a particular program and context. The list that follows is extensive, but it is not claimed to be exhaustive.

1. **Executing the command from the command line** Four basic methods were described in Chapter 5. The most straightforward is the one that runs the program from memory or disk, by entering a command on the command line and executing it from there with
F9 run x:filename.run **F10**
If the program has been previously loaded into general memory, **Nota Bene** will run it from there; otherwise it will look for it on disk. Access to memory is faster than access to disk, but some memory is being kept locked up for as long as the program is stored there. *[Probably not important in NBWin on a computer running WinXP.]* And, whether the program is run from memory or disk, this method normally takes several keystrokes (a minimum of 6 if you have a single-letter filename) to execute the command to run it. Even if you make no typing errors while entering the command, having to move your attention from your text to the command line and then entering all those keystrokes can be distracting from your work.

This can be avoided by staying with the **run** command but finding a more economical and efficient way of executing it. Fortunately there are several.

2. **Mapping to a keyboard key** If in one of the tables of your keyboard file you have a key that is not being used for other purposes, and preferably one that will serve as a good mnemonic for the program in question, you can map the program to that, with a line that looks like this:

NN=bx,r,u,n, ,p,r,o,g,r,a,m,.,r,u,n,q2

For 'NN' you substitute the number of the key in the table; for 'program' substitute the filename of the program. If the program is not in the subdirectory where you are when you want to run it, you must prefix ',p,r,o,g,r,a,m,...] with the drive\path needed to find it. (You do not have to use 'run' as the extension in the program's name; but you may want to use the same extension for all your programs, making it easier to identify them, and to find them with wild card directory orders, such as 'dir *.run'). It is essential that every character in that line (after the = sign) is separated from the character preceding it by a comma─with two exceptions,the keyboard functions 'bx' and 'q2'. When you have written that line, saved the keyboard file, and reloaded it, then any time that you want to run that program, a single keystroke will do it; your eye does not have to go up to the command line, and you do not have to do any distracting typing.

3. **Loading directly on a Phrase Key** If you have loaded the program to a regular phrase key (A to Z, 1 to 9), for example A, with

F9 ldpm <filename>.run,A **F10**

then, whenever you strike **Alt+Shift+A**, the program will be run. Using the keys available in a phrase library for loading and running programs can be one of the most efficient methods of running programs, because it is all done from inside memory; no access to disk is involved at all.

4. **Loading indirectly on a Phrase Key** This method may sound a little complicated, but it really is not. And it is very economical, because it enables you to run even the longest program from a phrase key, while requiring the minimum of memory to do it. Basically what you do is to load on to the key, not the program that you want to run from that key, but a second program, that consists solely of an instruction to run the first program. Let us suppose that the program that you want to run from the key is called WORDFREQ.RUN. Create a second program called, say, PGM, and write in it the following single line:

BX run wordfreq.run**Q2**

Save that file, and load it to a key with:

F9 ldpm PGM,A **F10**

If you now use **Alt+Shift+F3** to see what is saved to 'Alt+Shift+A' you will see that it is the line that PGM consisted of. Now, whenever you strike the key 'Alt+Shift+A', it will place on the command line the command 'run wordfreq.run', and execute it. The same technique can be used for indirectly loading other programs on other phrase keys. The programs themselves can be as long as you need, but the amount that is actually stored in memory per program is only a few bytes. Also you no longer need the file PGM, which can be deleted from disk, after you have saved to disk the phrase library into which you have loaded it.

[This is probably only worth doing in NbWin, in WinXP, with very long programs, if at all.]

5. **Loading on an Ampersand Phrase** These are phrases &A to &Z, &1 to &9, and they can be used only for loading programs to, and running them from; they cannot be used, as regular phrases and extended phrases can, for saving text or programming code to. The command to load is similar to that for regular phrases:

F9 ldpm <filename>,&x **F10**

substituting for 'x' any character from A to Z, or 1 to 9. There are two ways to run a program after it has been loaded to an ampersand key:

i. By the command on the command line:

BC func &x**XC**

ii. By mapping the ampersand phrase to a key in your keyboard file, so that the key's line reads:

NN=&x

After the keyboard file has been saved and reloaded, striking that key will run the program loaded on the ampersand phrase. Clearly, as in the case of the 'run <filename>.run' command above, method ii. is the more economical.

Unlike regular phrases, ampersand phrases are loaded to general memory, so that they do not compete for a portion of the limited 64k buffer that regular phrases go to.

But ampersand phrases do have certain limitations.

a. In the case of a regular phrase, you can find what has been loaded to it by displaying the list of current phrases with **Alt-F3**, or by displaying what has

been loaded to the particular key (say, A) with the command 'func sk', followed by 'A'. There is nothing corresponding that you can do with an ampersand phrase.

b. In the case of a regular phrase, you can save it to disk as part of a phrase library, and thus have it available for later working sessions. This cannot be done with ampersand phrases, which are stored in memory only, and are lost whenever you leave **Nota Bene**, or switch off the computer.

iii. Nevertheless there is something that you can do, and that is worth doing, if you have several programs that you would like regularly/frequently loaded to ampersand phrases. You can create a program that is, in effect, a batch file called, say, AMPERSND.RUN, consisting of a succession of 'ldpm program,&x' lines, looking like this:

Program to load ampersand phrases

BC ldpm program1,&1
BC ldpm program2,&2
BC ldpm program3,&3
BC ldpm program4,&4
BC ldpm program5,&5

Then, if you run AMPERSND.RUN, you will have those five programs loaded in a single operation; you can also, by viewing the file, see what programs you have loaded on which ampersand phrases.

iv. You can go one stage further, by including in your NBSTART.INT file the line:
BX run ampersnd.run**Q2**;*;
Your five programs will be loaded to their respective ampersand phrases every time you start up **Nota Bene**.

6. **Running Programs from XYWWWEB.U2** The XYWWEB.U2 program compendium allows you to run hundreds of programs by typing a mnemonic on the command line and striking your help key—and you can add your own programs to the bottom of U2.

7 **Running Programs from Macro Express menus** In Nota Bene 4 you could load programs on user help screens. Each help screen could contain 35 programs (corresponding to the alphanumerics, all but 0). You devoted one key in your keyboard table to calling the help screen; from there, striking one letter ran the program. It was beautifully economical. Instead of using up 35 keyboard-table slots, you used one; and the help screen told you not only the names of the programs, but also a brief description of what they did.

You cannot make user help screens in Nota Bene for Windows. But you can make the exact equivalent if you buy a shareware program called Macro Express (http://www.macros.com/). It allows you to build what they call menus of macros, 36 per menu. You can specify that the menus will work only with Nota Bene. Each line of a menu would contain a macro reading:

F9 run<filename>.run F10.

The key combination that opens the menu is defined within Macro Express, but it must be one that you do not want to use for something else in Nota Bene.

Exactly as with NB4 user help screens, you strike the key combination; a menu appears with a list of programs; and you strike the appropriate alphanumeric to run one you want. You can, of course, have more than one Macro Express menu devoted to running user programs in NBWin: you use one keyboard key for each menu.

There are other shareware macro programs; this is just the one I know and use—with NB and with other programs.
XYWWWEB.U2 is vastly more economical of keyboard space than using Macro Express, but you have to remember (or look up, or make a list of) the mnemonics.

8. **Running Programs from a Library file, using numbers as arguments**
This is a method that **Nota Bene for DOS** employed in its .OVL files, using letters as arguments. But numbers will do as well, as illustrated in the example below. That is a library of seven programs, constituting a file that we will suppose to be called FILES.RUN. Any one of the component programs can be run with the command **run files.run,#**, replacing # with the number for the program wanted. To save space here, all the programs have been left blank except No. 6, which reports the length of the current file, on the command

run files.run,6.

Benefits of library files
The advantages of packing a number of programs into a library are that it avoids wasting unused disk space on a multitude of possibly small program files, and that it cuts down on the number of entries in a directory. Programs can be picked out and run by any of the methods already described. There is theoretically no limit to the number of programs that can be stored in one library-program, but in terms of efficiency and speed of operation probably a limit of about twelve should be observed. *[This observation may not be pertinent in NB for Windows on a reasonably fast computer.]*

One precaution must be taken when creating a library file of this kind. No specific label must occur in more than one program. If the same label occurs in two programs, the corresponding «gl» call will always go to the first matching label that it finds in the library; conseqently the second of the two programs will never run correctly. *[There are routines to relabel and renumber programs in XYWWWEB.U2; I have not tried them.]*

In the following sample program, the first part determines what is the number entered as argument to the command to run the program, and then uses a «gl» call to point the program to the label at the start of the appropriate member program.

Sample library program, using numbers as arguments

```
«lbPROGRAMS»»;*;
«if«pv00»==1»«glSAVEA»«ei»;*;
«if«pv00»==2»«glSAVEB»«ei»;*;
«if«pv00»==3»«glDELBAK»«ei»;*;
«if«pv00»==4»«glCOPYBLK»«ei»;*;
«if«pv00»==5»«glSALIB»«ei»;*;
«if«pv00»==6»«glFILLNGTH»«ei»;*;
«if«pv00»==7»«glCOMPARE»«ei»;*;
;*;
;*;(1)«LBSAVEA»
;*;
;*;(2)«lbSAVEB»
;*;
;*;(3)«lbDELBAK»
```

```
;*;
;*;(4)«lbCOPYBLK»
;*;
;*;(5)«lbSALIB»
;*;
;*;(6)
«lbFILLNGTH»DX «sx20,«cp»»BF «sx21,«cp»»«sx21,«pv21»+1»BC jmp «pv20»XC DO BC
File is «pv21» Bytes longGT «ex»;*;
;*;
;*;(7)«lbCOMPARE»
```

---

9. Running Programs from a Library file, using text as arguments

[I have not tried this; the text of the program is unchanged from the NB4 version of the CPG. It is similar to that of the sample library on p. 101 above, so it should work.
The empty labels at line ends are NB4's way of doing what we now do with the comment string (;;). If you try the program, you must change all instances of 'ε' to 'î' - ASCII 238. You will also need to change all CRs to ;*;CR, and semicolon-hyphen strings at line beginnings to ;*;*
Note that the single letters/strings described in (1) below are separated by hard spaces.

The disadvantage of running programs using numbers as arguments is that it depends on identifying a sub-program within a library by one or more digits (or letters), and that neither of those is mnemonically helpful: to have to remember that 6 is the number of the program for reporting the length of a file can be a nuisance. An alternative method was devised by Itamar Even-Zohar, which replaces the numbers with text of your own choosing, so that the sub-program can be identified by, say, the word LENGTH, which is easier to associate with that particular program than the digit 6 is.

Below is a textual representation of this library-program, the name of the program here being assumed to be XC. Following the representation of the program there is a description of the basic procedure for entering sub-programs into it.

Sample library program, using text as arguments

```
«sv01, »«sv06,,»«sv14,»«sv99,XC:
»«lbParsText»«sx00,@UPR(«is00»)»«if((«is01»ε«is00»<0)&(«is06»ε«is00»<0))»«sx00,«is00»+
«is01»»«ei»«if(«is06»ε«is00»=>0)»«sx01,«is06»»«ei»«xs00,01,02,03,04»«sx05,«is02»»«sx00,«is
04»»«lbParsCommand»«sv11,? A B C LENGTH E F G H I »«sx99,«is99»+«is11»»«if«is05»ε«is
11»<0»BC «pr No such item in XC »«EX»«ei»«if«is05»==«is14»»BC «pr No com-
mand»«EX»«ei»«sv19,1»«sv01, »«lbPARS»«xs11,01,02,03,04»«if«is05»ε«is02»=>0»«glRUN-
ROUTINE»«ei»«sx11,«is04»»«sx19,«pv19»+1»«glPARS»

«lbRUN-ROUTINE»«lb
»«if«pv19»==1»«gl-?»«ei»«lb
»«if«pv19»==2»«gl-A»«ei»«lb
»«if«pv19»==3»«gl-B»«ei»«lb
»«if«pv19»==4»«gl-C»«ei»«lb
»«if«pv19»==5»«gl-LENGTH»«ei»«lb
```

```
»«if«pv19»==6»«gl-E»«ei»«lb
»«if«pv19»==7»«gl-F»«ei»«lb
»«if«pv19»==8»«gl-G»«ei»«lb
»«if«pv19»==9»«gl-H»«ei»«lb
»«if«pv19»==10»«gl-I»«ei»«lb
; for future routines:
»«if«pv19»==11»«gl$»«ei»«lb
»«if«pv19»==12»«gl$»«ei»«lb
»«if«pv19»==13»«gl$»«ei»«lb
»«if«pv19»==14»«gl$»«ei»«lb
»«if«pv19»==15»«gl$»«ei»«gl$»

«lb: Labels (programs) should be inserted here:»
«lb-?»BC «pv99»«sx98,@siz(«is99»)»«if«pv98»>73»«pr No room for additional display
»«ei»«ex»

«lb-A»BC this is A«ex»
;-<description>

«lb-B»BC this is B«ex»
;-<description>

«lb-C»BC this is C«ex»
;-<description>

«lb-LENGTH»DX «sx20,«cp»»BF «sx21,«cp»»«sx21,«pv21»+1»BC jmp «pv20»XC
DO BC File is «pv21» Bytes longGT «ex»
;reports length of current file

«lb-E»BC this is E«ex»
;-<description>

«lb-F»BC this is F«ex»
;-<description>

«lb$»«ex»
```

Explanation of library program

---

1. *The sv11 sequence* The call «sv11,...» saves to phrase 11 the following:

? A B C LENGTH E F G H I.

The interrogation mark identifies a sub-program that will display on the command line a list of all the sub-programs in the library.

The single letters A-I represent blanks, waiting to be replaced by mnemonic textual strings for individual programs.

The letter D has been replaced by the string LENGTH, which serves to identify a sub-program (the same one as in the previous section), that reports on the length of the current file.

2. *The gl sequence* The same sequence of identifiers is repeated in a succession of «gl» calls, each pointing to its corresponding label below.

3. *The labels* Then follow the labels, after each of which will be inserted the appropriate sub-program. At present «lb-LENGTH» is followed by its program, and the other labels by dummy messages, waiting to be replaced by programs.

4. *Adding a program* To insert a program, you need to enter a suitable mnemonic string three times:

(i) in place of an alphabetical letter in the «sv11,...» call;
(ii) in place of the same letter in the appropriate «gl...» call;
(iii) in place of the same letter in the appropriate «lb...» call. Then insert the program immediately after the label.

5. *Running a sub-program* To run a sub-program with the 'run' command, enter that command on the command line, followed by the name of the library file, followed by the mnemonic for the sub-program. A full version of the command to run LENGTH will be:

run xc,length

The argument, 'length' is there separated from the 'run xc' by a comma, but a space is just as good a separator, as in:

run xc length

With this particular type of library-program it is not necessary to type the whole name of the sub-program. All that is needed is a long enough string of characters from the name to identify it uniquely. In this case, as there are as yet no other names that might conflict with it, 'le' or even 'l' would be sufficient, as in:

run xc le or run xc l

If there are other sub-programs in the library, care must be taken to choose a genuinely unique string. For example, if there is one sub-program called COPY, and lower down the library another called COMPARE, then:

run xc co will always find COPY, never COMPARE

run xc com will find COMPARE

Although it is usually convenient, especially from a mnemonic point of view, to start from the opening characters of a name, this program does not require that. Although 'co' will find COPY, 'om', 'mp', 'pare', 'pa' etc., will all find COMPARE—in the absence of any earlier sub-programs in the library with those strings in their names.

If the sub-program is one that admits or requires arguments, they can be added on the command line after its name, or mnemonic abbreviation.

6. Displaying list of sub-programs The command:

run xc ?

will display on the command line the names of the sub-programs in the library, up to the limit of the command line's capacity.

Codes that work in Nota Bene for Windows

This list is perpetually provisional; new codes are added whenever I find them in updates of Nota Bene. You will find updates, under the title 'Allcodes' on Rick Penticoff's NB Users' website (see Introduction: Resources).

It provides an alphabetical list, as complete as I can make it, of codes that work in NB Win. It includes operators, wildcards, functions, immediate commands, embedded commands, and defaults. Operators and wildcards come before the main list.

Some codes have different meanings depending on whether they are:

—functions (2-character mnemonics that can be assigned to keys in the .KBD File)
—embedded commands (enclosed in command brackets)
—immediate commands (typed on the command line, or in programs, and executed with BX...Q2 or BC...XC)
—defaults (typically in NB.DFL as 'DF xx=yy').
—variables

The embedded command 'CP' means 'cursor position'; function CP means copy, and the variable «va$CP» shows the system Code Page. It is important to note these differences when deciding what code to use. It is no use embedding «CP» in a program if you want it to copy text.

Immediate commands are entered in the list in lowercase, to distinguish them from the other types of code. They need not be typed in lowercase on the command line.

Some codes are shown in the form: XX # or XX #/#; or XX x.
# = any number (sometimes a limited range)
x = any letter

The codes list does not include functions found in NB.KBD that have the form:

##=&X,Y,Z

These can be combined with other keyboard definitions, but cannot be used in programs. You can put them there, using programming mode, but they won't work. Some of the 'Y,Z' combinations are regular functions. For instance, '##=&X,Q,L' moves the cursor one space to the left; and so does the function 'QL'. But quite a few do not work except from the keyboard. I have included the functions, such as 'QL', that work on their own, but not those that only work from the keyboard.

I have tested the codes, except those marked 'not tested'. Most of those are default settings found only in NB.DFL. There are also a few commands such as 'delall', which I have not tested because I fear from the description that they might have drastic consequences.

Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
==	Equals to (double == necessary except in maths)
<	Less than
>	More than
<=	Less than or equal to
=>	More than or equal to
<>	Less than or more than
&	Performs logical **and** of 2 or more values
!	Performs **inclusive or** of 2 or more values
@cnv	Converts function call into keyboard function
@not	Performs a **not** of the following value
@siz	Checks number of characters in a string. This was used in NB DOS. In NB Win use instead «va|01» [| isnASCII 166]
@upr	Uppercases (use instead of NB DOS's rk)
@xor	Performs **exclusive or** of 2 values
@int	Save result of calculation as an integer (throw away fractional value, if any) (result shows on prompt line)
@abs	Returns absolute value of a number or calculation, i.e., the numeric result, without regard to sign
@dat	Convert date to hexadecimal number YYYYMMDD. E.g., «SX50,«VA$DAd.m.yyy»2.@dat**Q2** «PR@50» returns current date on prompt line
@dec	Convert hexadecimal number to decimal number [examples in XYWWWEB.U2—search for string '{{5@dec}}']
@hex	Convert decimal number to hexadecimal number [examples in XYWWWEB.U2—search for string '{{5@hex}}']
@dts	Convert hexadecimal date YYYYMMDD to decimal in format determined by default FZ. These two are used to compare two input dates, for instance, to determine which is earlier. [Search for '{{5@dts}}' in U2]
@lwr	Lower Case function [Search for '{{5@LWR}} ' in U2]
@num	Changes datatype of phrase from string to number (numbers have an invisible 2-byte flag, consisting of Ascii 0 followed by Ascii 1, appended to them in memory and therefore are 2 bytes longer than their string counterparts)
@tim	Convert military time HH:MM to hexadecimal number HHMM0000. [Search for '{{5@tim}} ' in U2]
@tms	Convert hexadecimal time HHMM0000 to time format determined by default MT [Search for '{{5@tms}} ' in XYWWWEB.U2]
î	(ascii 238 - in NB DOS showed as epsilon) Determines if first string is contained within second (reports number). Returns position of first occurrence of string1 within string2, starting at position zero (case sensitive). «SX01,"e"î"limpet"» returns "4" in phrase 01.

ð (ascii 240) Determines if one string contains another (true or false). Returns "TRUE" if string1 contains string2. Principally used in conditional tests, where th e position of string 2 within string 1 is unimportant. Case sensitive. E.g., this program segment:
«IF"limpet"ð"limp"><«PR OK»«EX»«EI»«PR Not OK»«EX»
returns 'OK'

Wildcards

View this part of the list in Draft or Show Codes View to see wildcards properly displayed, except those starting with a caret (^).

Entering wildcards that look like reverse-video single characters.
Into a keyboard table: type the two-character code (e.g., 'wl' for the single-letter wildcard. When you press the key combination, the wildcard will appear. Or for wildcards that do not have two-character versions, use nn+character (e.g., ##=nn,- for the any-but wildcard)
On the command line:do F9 func nn F10, then press the appropriate letter or number (e.g., 'n' for any single number). The wildcard will appear on the command line at the end of the 'func nn' command. You can erase 'func nn' and substitute (for instance) a search command.
Into a program, do F9 func nn; put cursor in file (e.g., with Alt F8), then press F10.

Entering double-character reverse-video-type wildcards
Into a program, e.g., **WA** , do 'pfunc' plus the 2 characters.
On the command line, enter them into text with pfunc, then cut and paste to the command line.

Entering caret + character wildcards
In text or on the command line, type the caret character plus the character.

0-9 or ^0-^9	Defines maximum no. of times the character can appear in the string. (e.g., to command "wild 80", use Wildcard-8 followed by Wildcard-0.)	[function]
A or **WA** or ^A	Any single letter or number	[function]
- or ^B or ^-	Any but next single character (represents NOT)	[function]
' or ' or **WC** or ^C	Carriage return character [Ascii 17, ']	[function]
€ or ^E or ^+ ??	Any single sentence separator (full stop/period, question mark, exclamation point)	[function]
??□or ^F	Line Feed Character	[function]
L or **WL** or ^L	Any single letter A-Z	[function]
N or **WN** or ^N	Any number 0 through 9	[function]
O or ^O	Allows search for more than one string	[function]
P or ^P	Regular or Alternate paragraph return	[function]
@ or ^R	Regular paragraph return	[function]
□	Carriage return+linefeed wildcard (Enter with 'func WC')	
S or **WS** or ^S	Any single separator	[function]
□ or **WT** or ^T	Tabs	[function]
W or **WW** or ^W	Any string from 1 to 80 characters. Mmust be used with at least 1 other character. 'se /x^W/' works; 'se /^W/' doesn't.	[function]
X or **WX** or ^X	Any single character	[function]

Main Alphabetical List

[U	Begin deletion operation that will be saved to clipboard "undelete" stack	[function]
U]	End deletion operation that will be saved to clipboard "undelete" stack *NB:These two must be used in pairs.*	[function]
<<	Enters ® in program	[function]
>>	Enters ¯ in program	[function]
@0-9/A-Z	Insert contents of phrase key x or run program assigned to phrase key x.	[function]
&0-9/A-Z	Run program assigned with LDPM filename,&# or &x NB: don't load user programs on ampersand keys C, D, E, G, I, L, S, U, X, which are used by Nota Bene.	[function]
#1 - #9	Move cursor to window no. 1 / window no. 9 NB: In this definition, '#' means the '#' character; it doesn't stand for 'any number'	[function]
1A 0/1	Read past end of file character (1), or stop at it (0) {DF 1A=0} (Doesn't work for me	[default]

ab or abandon	Abandon file	[immediate command]
ab/nv	Abandon file without verification (not necessary if Prompts, Abandon is unchecked in Tools, Preferences)	[immediate command]
abort	Abandon file	[immediate command]
AC	Turn Auto-Check on and off. Function string AC,AZ,AZ in a keyboard table or program turns off auto-replacement. When Auto-Check is on, command VA $AC returns 1.	[function]
AD	Append to macro	[function]
AH	Allow hyphenation (on/off=1/0) {DF AH=0}	[default]
AK	Accelerator: Move to a specific item in action bar or dialog box. Works only if assigned to the relevant Alt key; doesn't work in XPL programs.	[function]
AL 0/1	Automatic leading (line spacing) off/on {DF AL=1}	[embedded command, default]
AN 0/1	Toggles command brackets between ® ¯ (0) and « » (1) {df AN=1}	[default]
AOP	Backup path. {DF AOP=c:\nbwin\bak}	[default]
AOT	Min, max Autosave time in minutes {DF AOT=2,2}	[default]
apfil	Append text to file on disk. 'apfil x, now is the time' appends 'now is the time' to file x. (No prompt; it just does it.)	[immediate command]
append	Append one file to another	[immediate command]
apt	Append one saved file to the top of another saved file [See 'Apt' in CPG Appendix]	[immediate command]
AR	Execute Expand Abbreviation (NB: *toggle* Expand Abbreviation is AZ) (This is now, in combination with function XH, on the Ctrl key in all keyboard states; you can expand an abbreviation by pressing Ctrl. It also works, with or without XH, on other keys. When Auto-Replace is on, command VA $AR returns 1.	[function]

AS Move cursor between the two windows last displayed. [function]
AS Argument string [embedded command]
attrib Change file attribute (read only option) [immediate command]
au Toggle Auto-Uppercasing on and off [immediate command]
AZ Toggle Auto-Replace on and off. [function]
Function string AC,AZ,AZ in a keyboard table or program turns off auto-replacement.

BB Breakable block (end of non-breakable block) [embedded command]
BC Break column (marks point where column breaks in snaked columns) [embedded command]
BC Clear command line and move cursor to start of command line. [function]
BD Delete previous character. [function]
BF Move cursor to bottom of file. [function]
BF 0/1 Bottom footnotes - 0 = footnotes immediately below text; 1 = at bottom of page {DF BF=1} [embedded command, default]
bg Background color (d bg=#,#,#, where ## are red/green/blue values) [immediate command, default]
BK Stop command currently in progress; stop user program. [function]
BK 0/1 Backup off / on {DF BK=1} [default]
BL 0/1 Blank lines at top of page not suppressed / suppressed {DF BL=1} [embedded command, default]
BL Jump to left edge of current balanced pair of command brackets (current=where cursor is located). Lets you see whether you are at beginning or end of an embedded command or variable, so that e.g. you could DeFine it. Cursor must be adjacent to command bracket. [function]
BN Buttons Face {DF BN=0,18,18,12,1} [default]
argument= ButtonType,
Toolbar Width
Toolbar Height
button size
toolbar indent
(Type is: 0=Picture, 1=Text, 2=Both)
[Changing the arguments in NB.DFL does not seem to affect the button faces.]
BO Border - defines borders(see NB Help) [embedded command]
BR Jump to left edge of current balanced pair of command brackets. (see BL) [function]
BS Move cursor to bottom of screen. [function]
BT Bottom margin: footer, nominal, minimum, maximum: [embedded command, default]
footer is distance from bottom of page to bottom of running footer.
nom. is normal number of inches (or other default measurement) allowed for bt.
min and max are minimum and maximum number of inches.
{DF BT=0IN,1.1IN,.3IN,1.3IN}

BX Blind Execute - execute command without putting it on command line [function]
BX is not limited to length of the command line; and it does not blank the command line. [Can't be used in change-invisible commands.]

BZ Current button set (last argument in string) [default]
(not tested) {DF BZ=-,Button Sets,Main}

C0 - C14 Counter 0 - Counter 14 (11-14 not available from command line - use keyboard) [embedded command]

ca or call Open file [immediate command]
Ca/100 - call in Show Codes View
Ca/1 - call in Draft view
Ca/4 - call in Page Layout view
(See 'dt' below for other switches - except that /0 doesn't work with 'ca'; you must use /100.)

caf Open file without displaying graphics [immediate command]

cap Call program file (valid but unneeded in NBWin; use 'ca') [immediate command]

CB Move through windows in the reverse order to that in which they were opened. [function]

CB 1/0 Spell check beep on/off {DF CB=0} [default]

CC Toggle cursor between command line and text. [function]

cc Change case (of character under cursor, or of defined block) [immediate command]

CD Move cursor down one line. [function]

cd or chdir Change directory ('cd' on cmd line, with nothing after it, goes directly to NB main directory) [immediate command]

ce | cev Clear redlining edit - 'ce/v'=verify each change [immediate command]

cf Capitalize first letter of word (of character under cursor, or of defined block) [immediate command]

CF 0/1 Set footnote separator format status, to use either separator 1, 2, or 3 as needed (1), or only separator 1 (0) {DF CF=0} [default]

CH Delete the text on the command line without moving cursor [function]

ch | cha Change / change absolute (i.e., if case matches) (In NB Win, ch/cha are the same as ci/cia. For switches, see ci) [immediate command]

CI Switch to Overstrike mode (from Insert). [function]

ci | cia Change, invisible / change, invisible absolute (i.e., if case matches) [immediate command]

Switches:
ci/e |string| changes in elements (as well as body of file)
ci/w |string| finds string only if it's a self-contained word.
ci/s |string| limits the change to selected text (defined block).
If cursor is outside block, change is made from beginning to end.
If cursor is inside block, from cursor location to end of block.
ci/t |string| starts search at top of file
ci/# |string| 'ci/3 |string| finds the third instance of string (you can substitute any number).

CK 1/0 Spelling checker: ignore words with number (1=ignore) {DF CK=3} (What are 0 and 2?) [default]

CL Column location (position) [embedded command]

CL	Move cursor left one space (to previous line if at beginning)	[function]
clrasg	Clear all &@ phrases (XyWrite) (NB: use with great caution - clears ***all*** keyboard '&+letter' assignments for the session, so that, for instance, F9 will give message 'No macro assigned' instead of going to cmd line	[immediate command]
clrlib	Clear all phrase-library phrases from memory	[immediate command]
clrsgt	Clear all phrase-library phrases (XyWrite, same as clrlib - works in NB)	[immediate command]
CM	Toggle between draft and expanded (show codes) view.	[function]
CO	Insert a comma on cmd line or in text (use in keyboard file to enter an actual comma character into text)	[function]
copy	Copy a file	[immediate command]
copy/nv	Copy a file without verification), overwriting existing file if necessary	[immediate command]
copy/mv	Move file from one directory to another	[immediate command]
CP	Text cursor position	[embedded command]
CP	Copy currently selected block of text to cursor position.	[function]
CR	Move cursor right one character; wrap to next line.	[function]
CR	Sets cursor values {DF CR=1,0,0,4} a=blinking (0) or non-blinking (1) b=not used in Windows c=not used in Windows d=the number (0-5) specifies the width of the insertion point in Page Layout View	[default]
CS:5	The series of symbols that can be used instead of numbers in footnotes (not tested: I haven't tried to change them) {CS:5} * † ‡ §	[default]
CT	Create cellular table (see NB Help, Cellular Tables)	[embedded command]
CU...»	Count up operator (execute a segment of code a specified number of times)	[embedded command]
CU	Move cursor up one line.	[function]
CV 0/1	Prompt user before executing change (0=No 1=Yes)	[default]
cv	Change with verification (switches as for ci/cia)	[immediate command]
cva	Change with verification, absolute (match case) (switches as for ci/cia)	[immediate command]
d or default	Default - 'd xx=#' (on cmd line only - in NB.DFL the usage is 'DF xx=#')	[immediate command]
d:	Drive - e.g., 'c:' changes to drive C:	[immediate command]
D1	Undelete clipboard. DF D1=50,5 saves 50 clips to the NB clipboard, each 5 letters or more long	
DA	Date code in text (updates)	[embedded command]
DB	Move cursor to beginning of selected block.	[function]

DC Define counter (set counter numbering) [embedded command, default]
dc 0=1 Decimal numbers (default)
dc 0=I Uppercase Roman numerals
dc 0=i Lowercase Roman numerals
dc 0=A Uppercase letters
dc 0=a Lowercase letters

DC Begin selecting a column of text. [function]

DD End selecting block and delete it. If no selection, delete character. [function]

DE Move cursor to end of selected block. [function]

DE Soft carriage return {DF DE=,} [default]

default Same as 'd' [immediate command]

del or delete Delete a file. Use switch /nv to delete without verification (or uncheck 'Delete' in Tools, Preferences) [immediate command]

delall WARNING. Deletes ALL files in current directory, without verification [immediate command]

DF Begin or end selecting a block of any size. [function]

DF Dump footnotes (puts all footnotes at marker location) [embedded command]

DF 1/2/3 Dump footnotes, set 1, 2 or 3 [embedded command]

DH Conditional hyphen {DH=} (go to Show Codes View to see character) [default]

DI Directory defaults, command line {DF DI=1,6,0} [default]
Affects long directory listings (see command 'dirl'). In:
d di=x,y,z
x is filesize divisor (if x is more than 1, actual filesize is displayed divided by x. E.g., divide by 1024 to display filesize in kilobytes.)
y is number of lines displayed
z can be 0 or 1. 0 retains CRs, 1 removes them.

dir Directory - Switches: [immediate command]

+	lists all files in the specified directory and any associated subdirectories, e.g.: 'dir c:\nbwin\work+*.nb'
/fi	filenames and file information only (i.e., no subdirectories listed
/pa	subdirectory names only
/su	file summary information
/na/fi	filenames only
/na/pa	list of subdirectories in current directory, + list of available drives

dirl Display directory; show first few lines of file [immediate command]

DL Select line of text the cursor is on. [function]

DM Extend (or shrink) a block of selected text to cursor position. [function]

dm Restore all defaults (XyWrite) - NB: ***not*** tested; use with great caution) [immediate command]

dmfont Change default draft view and cmd line font. NB: use with caution. Changes display of « » on cmd line to ® and ¯. To restore to « », delete file C:\Windows\swlocal.ini. [immediate command]

DN Delete selected text without saving it on the delete stack. [function]

DO Turn on display (complement of DX). [function]

do Run DOS program with extension of .COM or .EXE while NB still running (not tested) [immediate command]

Dorothy Display images and text about Dorothy Day [immediate command]
dos Opens a DOS window at the current directory [immediate command]
DP Select paragraph the cursor is on. [function]
DS Select sentence the cursor is on. [function]
dsort Sorted directory (same as 'order') [immediate command]
Switches:
f to sort by filename
e to sort by extension
d to sort by last saved date and time
s to sort by size
p to sort by path name
r to sort in reverse order (use in addition to other modifiers)
h to add a header on top of directory
You can use more than one switch, separated by commas
DT Views: {DF DT=4} [immediate command, default]
DT=0 Show Codes View ("100" in "CA/#|ED/#|ME/#|RE/#")
DT=1 Draft View without page breaks
DT=2 Draft View with page breaks
DT=4 Page Layout View
DT=9 DT=1 with markers hidden
DT=10 DT=2 with markers hidden
DT=12 DT=4 with markers hidden
DT=17 DT=1 where only markers affected by scoping rules are hidden
DT=18 DT=2 where only markers affected by scoping rules are hidden
DT=20 DT=4 where only markers affected by scoping rules are hidden
DW Select word the cursor is on. [function]
DX Freeze display (complement of DO). [function]
DY m,n Printout color (m=foreground, n=background; but NB6 does not support printing non-white background colours) [embedded command]
m and n can be:

1	Black	6	Cyan	11	Orange
2	Blue	7	Magenta	12	Red
3	Brown	8	Maroon	13	Violet
4	Charcoal	9	Neutral	14	White
5	Green	10	Olive	15	Yellow

DZ End selecting a block if selection is in progress. [function]
DZ Set date format {DF DZ=d Mmmm, yyyy } [default]

EB 1/0 Error beep beep on/off {DF EB=0} [default]
EC Move cursor to end of current cell [embedded command, function]
ED Select current row of cells in a table [function]
edp Call file cursor is on. 'edp x' calls file x in directory [immediate command]
EE Delete a row of entries in a table. [function]
EE [+] # Element end - offset from bottom margin [embedded command]
+ sets offset from bottom margin to top of capital letter in last line of text
# is vertical offset
EF Cancel window (e.g., editing window) and close [function]

EH 0/1 Error Help off/on. If off, prevents programs from incurring "errors" deliberately. For example, if you call a network file, in order to learn whether you're connected to a network, an error message might pop up and pause or halt your program.'d EH=0' prevents that. [default]

EI End if - ends an «if» statement [embedded command]

EL # Extra leading - #=amount of vertical space; affects only current line [embedded command]

EL Move cursor to far left of line, then to left end of previous line [function]

EN Edit next: opens next file that matches a global filename specification. First you must *call* the first file that matches the filespec. E.g., if the filesec is '*.NB, do: 'ca *.NB'. Then do 'func en' to call the next .NB file; and continue till all files with that filespec have been called. If Quick Open is used to open a sequence of files of a particular type (that action by itself opens the first file matching that type), EN (on CS F9) opens the next file in the sequence. [function]

EP Error prompt (set from Tools, Preferences, Command Prompts) [default]
erase
print selection
print dir
abort
func SA
del markers
font mismatch
prompt message for Correct command
{DF EP=0,1,1,0,0,0,1,0}

ER Error flag [embedded command]

ER Move cursor to right end of line, then to end of next line. [function]

erase Delete file [immediate command]

ernv Erase, no verification (NB: will erase saved version of on-screen file) [immediate command]

es 0/1 Error suppression off/on (on=1) (es 0 not needed in NB Win) [immediate command]
ES 1 should go at the head of programs - errors can cause delays, even if error beep is turned off.

ES Release selected text [function]

ES # Enable scoping rules {DF ES=0} [default]
Specifies the boundaries, or scope, of formatting commands:
es=0 - apply formatting cmds from cursor position forward
es=1 - apply them from beginning of current para, overwriting any other occurrences of the command within the para. Stays in effect until overridden by another occurrence of same cmd in a subsequent para
es=2 - 'previous cmd forward'. Changes any previous occurrence of the command to the new value. If no previous occurence, inserts new value at cursor.

ET [+] # Element top margin (sets offset from top margin) [embedded command]
+ (optional) sets offset from top margin to top of a capital letter in the first line of text
# is vertical offset

EU	Sets language dependent parameters {DF EU=.,:,;} (not tested)	[default]
EX	Exit program	[embedded command]
EX1	Exit all running programs	[embedded command]
exist	Test for existence of file (used in programming)	[immediate command]
FA	Framed area (see NB Help, Frames, Inserting Frame, Command Line)	[embedded command]
FC	Force center (centres text)	[embedded command, function]
FD	Form depth (page length) {DF FD=117DI}	[embedded command, default]
FD	Compare files in current and adjacent windows - stop where no match	[function]
FF 0/1	Form feed off/on	[embedded command, function]
FF	Force display to refresh. (I don't know what this does in NB Win)	[function]
fg	Foreground (text) color (d FG=#,#,#, where #,#,# are red/green/blue values)	[embedded command, default]
FH	Format bar (overall height of bar, height of multistate controls inside, indent: button size) (not tested) {DF FH=17,17,1,10}	[default]
FI	Field identification (Ibidem, and XyWrite Mailmerge only)	[embedded command]
find	Find file on drive (can be slow unless path or file specification included)	[immediate command]
findl	Same as 'find', but displays first few lines of file	[immediate command]
FL	Flush left - text flush with left margin	[embedded command, function]
FM	Compare files in current and adjacent windows - stop where files match	[function]
FM 1/2/3	Footnote format, sets 1-3 - define footnote format	[embedded command]
FN1/2/3	Footnote, sets 1-3 - inserts footnote	[embedded command]
fo	Print to disk (file FO.TMP)	[immediate command]
format	Print to file FO.TMP (same as 'fo') (Worth using with caution, and definitely not in a DOS window!)	[immediate command]
FP	Final page (see NB Help, Page numbering, command line)	[embedded command]
FQ	Items that go on the format bar (not enabled in NB6.1) {DF FQ=-,format sets,main}	[default]
FR	Flush right - text flush with right margin	[embedded command, function]
FS 1/2/3	Define footnote separator 1/2/3	[embedded command]
FT	Define footnote transition	[embedded command]
FT	Set line height to fixed instead of automatic {DF FT=.166IN} (.166 is 1/6" or 1 line) If auto-leading is on by default, it must be turned off with 'd al=0' for FT to work.	[function]
FU	Fill units, used in NBDOS for estimating space of refs. {DF FU=3,5} (not tested)	[default]
func	Insert/test/execute function (e.g., 'func bc' inserts a 'begin column counter' code	[immediate command]
FW 1/2/3	Define footnote wrap separator, set 1/2/3 - for notes that continue on a second page	[embedded command]
FX	Field separator {DF FX=⇥}	[default]
FZ	Date format for directory display {DF FZ=d.m.yyyy}	[default]

GC General citation [embedded command]
GH Move cursor to command line without clearing command line. [function]
GL...» Go to label [embedded command]
go #-# Go to page # (and, optionally, line #) [immediate command]
gofile Go to open file ('gofile x' goes to file x if it's open) [immediate command]
GT… Get contents of macro (phrase key) [embedded command]
GT Move cursor to text area. [function]
gtsgt Insert phrase-library phrase on command line or in text [immediate command]
(XyWrite but works in NB). To insert phrase in text, put cursor in text before pressing F10.
GU Gutter - inside,outside (white space beside columns, [embedded command, default]
cellular tables and frames {DF GU=2DI,0}
GW {DF GW=0} (in NB.DFL) (What does this do?) [default]
GX (in NB.DFL) (What does this do?) [default]
{DF GX=0,0,255,255,0,0,255,0,0,255,0,255,0,128,0,0,255,255,255,255,0,
128,0,255,0,255,0,128,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

H@ Open NB Help [function]
HB 0/1 Left-to-right or right-to-left text entry - also reverses [embedded command]
orientation of all words from beginning/end of current line to end of file.
hc Move cursor to beginning of current cell [immediate command]
HI Define whole file [function]
HK 0/1 Begin link (HK 1); end link (HK 0) [function]
«HK1Web|[web address]»[link name]«HK0»
«HK1Email|[email address]»[link name]«HK0»
«HK1NB File|[name of file]»[link name]«HK0»
«HK1Auto|[name of file in other program]»[link name]«HK0»
«HK1Paint|[name of bitmap file]»[link name]«HK0»
«HK1Jpeg|name of .jpg file]»[link name]«HK0»
HM Move cursor to top of the screen. [function]
HV Sets hyphenation rules {DF HV=6,3,3} [default]
HY 1/0 Automatic hyphenation on/off {DF HY=1} [embedded command, default]

IC Interrupt command (makes running header start on next page, [embedded command]
not current page, if placed before the running header.
IE End of linked text (see NB Help, Indexing) [embedded command]
IF… If conditional (must be paired with «EI» [embedded command]
IG Include graphic (merge graphic into file) (not tested) [embedded command]
IL Insert at Line (formerly known as Index label) [embedded command]
enable specification of an alternate sort sequence for an entry (line, paragraph, Ibidem field). from NB, insert via Tools, Set Item Order. From Ibidem, insert via Edit, Set Field Sort Order
IM|IE Index marker. Subheadings are: [embedded command]
IT=heading|subheading|sub-subheading Index Term (to be indexed)
CA=** (2-character mnemonic for category) Category -- of index term
SC=** (2-character mnemonic for subcategory) Subcategory

RT=** (2-character mnemonic for reference type) Reference Type
GT=** (2-character mnemonic for get-text type) Get Text Type
SO=text or category+subcategory| same | same Sort Under
CR=heading|subheading|sub-subheading Cross-reference term

IO	Turns on/off document information {DF IO=0}	[default]
IP	Indent paragraph {DF IP=4DI,0,0}	[embedded command, default]
IR	Open auto-check/auto-replace pair dialog	[function]
IS…	Insert text phrase	[embedded command]
IT	Insert a tab on command line or in text	[function]
IV	Invisible comment (not visible even if show markers is on) [See Function IV on p 164.	[embedded command]
JC	If cursor is on a marker, move right past all markers (until next text) (not in Page Layout View)	[function]
JD	Jump to dialog box (Instances in NB.DLG)	[embedded command]
JM	Display dialog box or run menu routine with specified keyword. Must refer to valid framename in loaded DLG file (e.g., NB.DLG) E.g., JMCustDispFrameQ2;*;	[function]
jmp #	Jump to character # in file	[immediate command]
JR	Journal {DF JR=1} Not supported in NB8.0 (for future use)	[function, default]
JU	Justification on {DF JU=}	[default]
JU 1/0	Justify on/off (on=1)	[embedded command]
jumplb	Jump to label. 'jumplb x' jumps to label x	[immediate command]
KF	(On Ctrl 1, Esc - which activates Window Start menu systemwide)	[function]
L1	Command Line colors: current path	[default]
L2	Message Line colors: 1 status indicators 2 normal message 3 page number-clock 4 wait-for-user message (|) 5 warning message (ascii 12) (L0, L3, L4 have no effect in Windows) {DF L1=112} {DF L2=112,112,112,176,224}	[default]
la	Codepage - language command: lets you work with files created in code page 437 (US) or 850 (multilingual). Default is 437. .(not tested)	[immediate command]
LB 1/0	Determines whether selection will give error if complete framework element is not selected	[default]
LB	Label - insert label in file	[embedded command]
LB	Move cursor to far left of line and no farther.	[function]
lc	Lowercase (of character under cursor or defined block)	[immediate command]
LC	Hard carriage return {DF LC=¶}	[default]

LD — Leadering - insert row of characters (any single character), move existing text flush to margin(s) [embedded command]

LD — Line Down - move cursor directly down one line. [function]

ldkbd — Load keyboard file [immediate command]

ldhelp — Load user help file, e.g., XYWWWEB.U2 [immediate command]

ldlib — Load phrase library (NB: doesn't load the associated .LIX comment file [immediate command]

ldpm — Load program on phrase key - 'ldpm [name],[key]' [immediate command]

ldsgt — Load phrase library (XyWrite; works in NB - same as ldlib) [immediate command]

LE — Move cursor to far right of line and no further. [function]

LH — Set super/subscript (low-high) {DF LH=50,117,22} [default]
a=percent size
b=percent up
c=percent down

LJ 0/1/2 — Line justify - align one line of text left, (0), centre (1) or right (2) All text below that line remains aligned as it was preceding the lj command. [embedded command]

LL — Move cursor left one character; don't wrap to previous line [function]
[Doesn't work in Page Layout View. Use function CL in programming.]

LL — Line leading (only works if auto-leading is turned off, either by default or with 'd al=0' {DF LL=0LI,0LI} [embedded command, default]

LM # — Left margin (obsolete) [embedded command, default]
is no. of inches (or other default unit) for margin {DF LM=0in}

load — Load keyboard file, NB.DFL, spell file, etc. (XyWrite/NB) [immediate command]
In XyWrite the following types of file can be loaded with 'load'(;XX; = file ID, at top of file):

Printer file	;PR;
Default file	;PR;
Help file	;HL;
Menu file	;MN;
Dialog box file	;DG1;
Personal spelling dictionary*	;SP;
Sort file	;SO;
Keyboard file	;KB;
Hyphenation file	;HY;
Command override file	;U2;
Soft font file	;SO;

Of these, ;SP;, ;KB; and ;HY; files exist in NB Win.
*NB: 'load + [fileB.spl]' adds the definitions of [fileB.spl] to those of any already loaded .SPL file, for the current session.

LR — Move cursor right one character; allow to move past carriage return. [function]
[Doesn't work in Page Layout View. Use function CR in programming.]

LR 1/0 — Enter text from left to right (1) or right to left (0) [embedded command]

LS — Line spacing (in lines, inches, etc. - only works if auto-leading is turned off, either by default or with 'd al=0') [embedded command]

LU	Move cursor directly up one line.	[function]
LV 0-14	Counter 0 - Counter 14, for Table of Contents	[embedded command]
M0	Type in mode at cursor, or make selected text match mode at cursor.	[function]
M1	Type text in normal mode, or make selected text normal.	[function]
M2	Type text in bold, or make selected text bold.	[function]
M3	Type text in underline, or make selected text underlined.	[function]
M4	Type text in reverse mode, or make selected text reverse (doesn't work: inserts a non-functional MDRV)	[function]
M5	Type text in bold underline, or make selected text bold underlined.	[function]
M6	Type text in bold reverse mode, or make selected text bold reverse. (doesn't work: inserts a non-functional MDRV)	[function]
M7	Type text in superscript, or make selected text superscript.	[function]
M8	Type text in subscript, or make selected text subscript.	[function]
M9	Type text in italic or make selecte text italic.	[function]
MB	Display messages in message boxes vs. status line {DF MB=0}	[default]
MC	Mark column - select cell at cursor location in a table	[function]
MD	Scroll text and cursor down one line.	[function]
MD	Type style:	
	md bi (bold italics)	[embedded command]
	md bo (bold)	[embedded command]
	md bu (bold underline)	[embedded command]
	md dn (strike through)	[embedded command]
	md in (double underline)	[embedded command]
	md it (italics)	[embedded command]
	md nm (normal)	[embedded command]
	md sb or md sd (subscript)	[embedded command]
	md su (superscript)	[embedded command]
	md ul (underline)	[embedded command]
ME	Reports on memory management (and see 'mem')	[function]
me or merge	Merge file	[immediate command]
mem	Reports on memory management	[immediate command]
MF	Enter certain characters (in keyboard table)+ 4 hex digits Cursor must be in file when function is executed, otherwise character appears on command line. E.g., F9 func MF [go to text, type 'n', press F10, then numerals 7461. Tilde appears over 'n'. [Nota Bene only, not XyWrite]	[function]
MG	Current message - not activated in NB 6 {DF MG=}	[default]
MI	Switch from Overstrike to Insert mode until a cursor key is pressed.	[function]
MK	Toggle display of format markers and line ending markers.	[function]
mkdir	Make directory	[immediate command]
MR 1/0	Metric ruler on/off {DF MR=0}	[default]
MS	Designate that a mouse is installed - Assigned to key 105. (why would one want to change/use this?)	[function]
MT	TOC marker	[embedded command]
MT 1/0	Military time on/off (military time=24-hour clock) {DF MT=0}	[default]
MU	Scroll text and cursor up one line.	[function]

MV Move currently selected block of text to cursor position. [function]

MW Microsoft Windows functions (do 'func mw', then enter 2-letter code): [function]

- ac Cascade all text windows
- ah Split all text windows horizontally
- ar Tile all text windows
- av Split all text windows vertically
- cb Display contents of Windows Clipboard
- cl Close text window
- cp Copy selected text to Windows Clipboard
- cu Cut to Windows Clipboard
- hh Display help on using Help files (Windows Help)
- hi Display Help Index (Nota Bene Help)
- mn Minimize NB screen
- pa Past text from Windows Clipboard
- mv Display 4-headed arrow to move NB screen (minimizes NB at top lhs of screen; dragging enlarges it)
- mw Move window
- mx Maximize NB screen
- pa Paste text from Windows Clipboard
- pl Paste link (doesn't seem to do anything)
- pr Display information about Windows printer driver
- ps Paste special
- qu Quit
- rm Restore text window to maximum size
- rs Restore NB screen to previous non-max|min size
- rw Restore file
- sf Repaint the screen (doesn't seem to do anything)
- sl Scroll left
- sr Scroll right
- sw Size document window
- sz Display 4-headed arrow to move text window
- wf Make current text window full screen
- wi Minimize text window

MX Type in mode at cursor - same as M0, but does not get inserted in programs. [function]

MY Magnify (specifies point size & typeface in dialog boxes) {DF MY=8,Helv} [default]

MZ Type text in bold italic, or make selected text bold italic. [function]

NB Designate selected block of text as unbreakable. [embedded command, function]

NC Move cursor to next character. [function]

ne or new Create a new file (to open an unnamed file, use 'ne' without argument) [immediate command]

Switches:

- ne/100 opens new file in expanded view
- ne/# where # is any recognised display type, opens it in that display type (but ne/0 doesn't work: use ne/100 instead) (See 'dt' above for other switches)

nep — New program file (valid but unneeded in NBWin; use 'ne') [immediate command]

NF — Move cursor to first line of next printed page. [function]

NF 1/2/3 — No footnotes, set 1/2/3 - turn off printing of footnotes [embedded command]

NI — No index - prevent index from printing [embedded command]

NI — Prevent key from being passed to DOS (used in some keyboard assignments in NB DOS, but not in NB Win) [function]

NJ — Justification off [embedded command]

NL — Move cursor to start of next line. [function]

NM — No markers - hide format markers and line ending markers [function]

NM 0/1 — No modification: protected block off / on. Inserts NM1 at beginning of block, NM0 at end. To make block unprotected again, delete codes in Draft or Show Codes View [embedded command]

NN x — Generic Wild Card - the next character is the wild card. [function]
Wildcard is inserted at cursor position, either on command line or in text. To put it in text, go to command line, type 'func nn' + letter or number, then place cursor in text and press F10
Switches:

- (minus sign)	negation wildcard
numbers 0 through 9	numeric (repetition) wildcards
A full stop (ascii-46)	sentence separator wildcard- E (finds only full stops)
Ascii-17	Ascii-13 (carriage return) wildcard ' (view in Show Codes View)
Ascii-25 down arrow	Ascii-10 (linefeed) wildcard -
Ascii-27	CrLf (carriage return+linefeed) wildcard (also produced by executing func WC); Ascii 27 produces a B, which turns into a left arrow on the command line, but does not find CrLfs. Func WC produces a □ (view in Show Codes View), which also produces a left arrow on the command line, but does find CrLfs.
A	alphanumeric wildcard
L	letter wildcard
N	number wildcard
O	logical OR wildcard.
S	separator wildcard
W	variable-string wildcard
X	variable-character wildcard (also produced by executing funcs WA, WL, WN, WS, WW and WX)

NO — No operation - used in keyboard files as dummy assignment when beginning a key assignment that inserts text [function]

now — Time as text in file [immediate command]

NP — Move cursor to start of next paragraph. [function]

NS — Next style (invokes next style when working with styles) (needs testing by someone who uses styles) [embedded command]

NS — Move cursor to start of next sentence. [function]

NT — Move cursor to the next tab position without inserting a tab (not in Page Layout View) [function]

NT Annotation [embedded command]
Switches:
/0 General (NT/0)
/1 Comment (NT/1)
/2 Instruction (NT/2)
/3 Argument (NT/3)
/4 Drafting Tip (NT/4)
/5 Query (NT/5)
/6 Ibidem (NT/6)
/7 Orbis: Status (NT/7)
/8 Orbis: Keywords (NT/8)
/9 Style Manual (NT/9)

NU Delete selected text, without saving it for possible later undelete. [function]

NW Move cursor to start of next word. [function]

NW # Automatic windows {DF NW=3} [default]
To change permanently, change the default in NB.DFL.
'd NW=0 and ''d NW=3': calling file removes on-screen directory; abandoning a file or directory does not leave blank window.
'd NW=1' and 'd NW=5' make directory persistent when file called; abandoning a file or directory does not leave blank window.
'd NW=2' and 'd nw=4' make directory persistent; abandoning a file or directory leaves blank window (not untitled file) on screen.

NX Move cursor successively through all open windows. [function]

O1 0/1/2 Specifies how NB handles screen/printer font mismatches in [default]
Page Layout View {DF O1=1}
0=use printer font widths and do error correction between words
1=use printer font widths and do error correction between characters
2=use screen font widths and do error correction at the end of the line)

OB 1/0 Overstrike beep on/off {DF OB=1} [default]

OD 0-7 Offset display. 'd od=0' hides onscreen margins; 'd od=2' displays them [default]
'd od=4 displays margins as grey hatching.
{DF OD=32} (My default is 32, but the information I found (perhaps written for XyWrite) gives only 0-7)

OF Offset for right and left pages {DF OF=1IN,1IN} [embedded command, default]

OL Enter outline level, 1-9 [embedded command, function]

oln Change outline to Outline View [immediate command]

OP Access the previously accessed Menu/Help/dialog frame (works with some frames, not others) [function]

OP # Orphan (min. no. of lines of a para allowed at bottom of page) {DF OP=3} [embedded command, default]

OR 0/1 Orientation portrait / landscape [embedded command]

order Sort a directory (e.g., 'order d,r' to sort in reverse date order) [immediate command]
Switches
f to sort by filename
e to sort by extension

d to sort by last saved date and time
s to sort by size
p to sort by path name
r to sort in reverse order (used in addition to other modifiers)
h to add a header on top of directory

OS 0/1	One-sided printing off / on	[embedded command]
outline	Change outline to Outline View	[immediate command]
p	Pause (approximately 1 second)	[immediate command]
PC	Move cursor to the previous character.	[function]
PD	Scroll down one screen.	[function]
pe | pev	Undo redlining, without or with verification	[immediate command]
PF	Put field (Ibidem .FOR files and XyWrite mailmerge) (not tested)	[embedded command]
PF	Move cursor to first line of previous printed page.	[function]
pfunc or pfun	Enter function code into file from command line	[immediate command]
PG	Start new page Switches: PG #IN (or LI) - forces break at x IN/LI, etc.) PG E/O - new page if on even page / odd page	[embedded command]
PL	Page length (e.g. 'PL32LI')	[embedded command]
PL	Move cursor to start of previous line.	[function]
PN	Page number	[embedded command]
PP	Move to start of previous paragraph.	[function]
PR…	Prompt	[embedded command]
print	Print Switches (note compulsory commas): ,#-# page range, e.g., 'print ,3-6' / broken page range, e.g., 'print ,2-10/16/30-50 - (at end of last number) print to end of file, e.g., 'print ,2-' /# multiple copies, e.g., 'print/2' to print 2 copies ,e and ,o even and odd pages, e.g., 'print ,e' to print all even pages Switches can be combined, e.g., 'print ,2-10/15,30-,e'	[immediate command]
print @	Print a group of files (see NB Help)	[immediate command]
printf	Write printer file FO.TMP to disk	[immediate command]
program	New program file (unneeded in NBWin; use 'ne') (XyWrite)	[immediate command]
PS	Previous style (invokes previous style when working with styles)	[embedded command]
PS	Move cursor to start of previous sentence.	[function]
PT	Move cursor to previous tab position.	[function]
PU	Scroll up one screen.	[function]
PV…	Put value of text macro	[embedded command]
PW	Page width {DF PW=85DI}	[embedded command, default]
PW	Move cursor to start of previous word.	[function]
PX	Character used for visible page break {DF PX=45}	[default]

Q2	Finish command started with BX, or finish call to Help routine started with functions BX, JM or JH	[function]
QC	(in NB.DFL) {DF QC=0} Flag setting cursor movement in Hebrew, Arabic, etc. (whether arrows move left/right or next/previous)	[default]
QL	Move cursor left one space (to previous line if at beginning)	[function]
QR	Move cursor right one character (to next line if at end)	[function]
quit	Quit NB (prompts to save unsaved files)	[immediate command]
qs	Change directory (same as cd)	[immediate command]
R0-9	R0 to R9: Enter the ascii character associated with the number(s). If using more than one R+ number in keyboard table, note that multiple R# assignments require a terminating func NO: ##=R2,R6,R5,NO	[function]
RB	Delete the word before the word the cursor is on.	[function]
RC	Read character (allows user input from keyboard in programs	[embedded command]
RC	Delete character under the cursor.	[function]
RD	Delete selected block of text.	[function]
rd or rmdir	Remove directory (it must be empty, and not the current directory)	[immediate command]
RE	Delete text from cursor to end of line.	[function]
re or read	Open file for reading only	[immediate command]
REC	Refer to chapter number	[embedded command]
REC #	Refer to chapter number #	[embedded command]
red on/off	Toggle redlining on/off	[immediate command]
REF #	Refer to footnote or counter number #	[embedded command]
REL	Refer to label	[embedded command]
remove	Remove contents of one phrase-library phrase	[immediate command]
ren or rename	Rename a file	[immediate command]
REP #	Refer to page number #	[embedded command]
RF	Running footer	[embedded command]
RFA	Footer, all pages	[embedded command]
RFE	Footer, even pages	[embedded command]
RFO	Footer, odd pages	[embedded command]
RG	Case:	[embedded command]
	RG or RG 0 upper and lower as typed	
	RG 1 caps, ignore shift state	
	RG 2 lower, ignore shift state	
	RG 3 small caps	
	RG 4 caps and small caps	
RH	Running header	[embedded command]
RHA	Header, all pages	[embedded command]
RHE	Header, even pages	[embedded command]
RHO	Header, odd pages	[embedded command]
RK	Read Key - Toggle Record Keystrokes mode on and off (executes each command recorded, unlike function TS.)	[function]

RK	Read character (Note: not 'read and uppercase', as in NB DOS). Goes with RX.	[embedded command]
RL	Delete line the cursor is on.	[function]
RM #	Right margin (obsolete) (# is inches or other unit from margin)	[embedded command]
rmdir	Remove directory (it must be empty, and not the current directory)	[immediate command]
rmvdup	Removes duplicates in sorted list	[immediate command]
RN	Round off numbers - {DF RN=0} (of line count on status line)	[default]
RO 1/0	Turn redlining on/off.	[function]
RP	Delete paragraph the cursor is in.	[function]
RS	Delete sentence the cursor is in.	[function]
RS	Record separator {DF RS=□}	[default]
RT 0/1	Relative tabs off/on (establishes tabs relative to left margin and gutter	[embedded command]
run	Run XPL program	[immediate command]
RW	Delete word the cursor is on.	[function]
RX	Execute the last set of keystrokes you recorded.	[function]

S-	Displays last command on command. line	[function]
S1	Acute accent (func s1,[letter to be accented], e.g., 'func s1,e')	[function]
S2	Grave accent	[function]
S3	Umlaut	[function]
S4	Circumflex	[function]
S5	º accent	[function]
S6	Tilde	[function]
S7	Underline (doesn't work in NB Win)	[function]
SA	Save file	[function]
sa or save	Save file	[immediate command]
sa/ne	Save under new name, switch to new version (old version remains on disk)	[immediate command]
sa %x	Saves contents of phrase x to a file named X.SAV. (See Appendix of CPG, 'SA%'	[immediate command]
sad	Save selected (defined, highlighted) text to new file: 'sad [filename]' Same as 'savedef','savesel' and 'sas'	[immediate command]
sad/ne	Save define and switch to new version of file	[immediate command]
sas	Save defined text (same as 'sad')	[immediate command]
salib	Save phrase library (NB: doesn't save associated .LIX comment file)	[immediate command]
savedef	Same as 'sad'	[immediate command]
savesel	Same as 'sad'	[immediate command]
SC	Superscript mode for footnotes {DF SC=SU}	[default]
SD	Space between text/footnote {DF SD=xIN, or xLI, etc.}	[default]
se | sea	Search - sea= absolute (match case)	[immediate command]

Switches:

se/e |string|	searches in elements (as well as body of file)
se/f |string|	puts the cursor on the first letter of the found string.
se/w |string|	finds string only if it's a self-contained word.

se/s |string| limits the search to selected text (defined block).
If cursor is outside block, block is searched from beginning to end.
If cursor is inside block, block is searched from cursor location to end of block.
se/t |string| starts search at top of file
se/# |string| 'se/3 |string| finds the third instance of string (you can substitute any number).

searcha Search absolute (same as 'sea') [immediate command]
searchb Search backwards (same as 'seb') [immediate command]
searchba Search backwards absolute (same as 'seba') [immediate command]
se[/c] range|string| Search directory - searches through a series of file names separated by commas (range) for the text (string). 'searcha' and 'sea', 'searchba' and 'seba', can also be used [immediate command]
You must do search from blank window; do:
F9 ne F10 before executing search command.
Switch:
/c tells program to count number of times string appears, but not to stop at each match.
(Doesn't work for me)
seb | seba Search backwards - seba= absolute (match case) [immediate command]
sec Inserts fixed time in text with hours, minutes *and* seconds [immediate command]
SF 1/2/3 Set footnote style and number in set 1/2/3 [embedded command]
sf#,1 decimal numbers (default)
sf#,I uppercase Roman numerals
sf#,i lowercase Roman numerals
sf#,A uppercase letters
sf#,a lowercase letters
SG x or # Insert text or run program from phrase key x or # (XyWrite) [function]
sg1926=4((in NB.DFL) [default]
sg1927=3 (in NB.DFL) [default]
sg1928=1 (in NB.DFL) [default]
sg1700=0 Define type (in NB.DFL) [default]
sg1701=1 Quote type (in NB.DFL) [default]
sg1984=1 (in NB.DFL) [default]
SH # Snake height (sets depth of columns for snaking columns; # is the depth) [embedded command]
SH Show Help. Displays the top-level menu (On key Right Alt—98) (examples in XYWWEB.U2) [function]
SI Switch to Insert mode (from Overstrike). [function]
SK Show single phrase-library phrase ('func sk'; then, when prompted, strike alphanumeric for phrase whose contents you want to know) [function]
SL Save all open files in all windows. [function]
SM Add the number the cursor is on to the total [function]
SN #,#,# Snaking columns - 'sn x,y,z', where x,y,z are the locations (in your default measurement unit) where you want columns of text to begin, e.g., 'SN 0,3.25,5' [embedded command]
SO Check spelling of a single word. [function]
sort Sort selected text. [immediate command]

sortd	Sort list in [filea] alphabetically, and put sorted list in [fileb], leaving [filea] intact - 'sortd [filea],[fileb]'	[immediate command]
SP	Set page number (sp #) Arguments: is this right word? sp#,1 decimal numbers (default) sp#,I uppercase Roman numerals sp#,i lowercase Roman numerals sp#,A uppercase letters sp#,a lowercase letters	[embedded command]
SP	Switch to page-line view.	[function]
spell	Spell check Switches - to check: all elements /e all footnotes /fn specific footnote series /fn=1 or /fn=2 or /fn=3 all notes /nt specific note series /nt=1 or /nt=2 or /nt=3 etc. headers /rh footers /rf	[immediate command]
SS	Save style	[embedded command]
SS	Turn on Program mode.	[function]
ST	Set numeric keypad to numbers (doesn't work in NB Win)	[function]
ST 1/2	Show tab character - 1=expanded view, 2=draft view, 3=both {DF ST=0}	[default]
st or store	Store file	[immediate command]
st/nv	Store file without verification	[immediate command]
SU…	Subroutine	[embedded command]
SV…	Save value as literal	[embedded command]
SV	Save selected block of text to phrase-library key of next character typed	[function]
SX…	Save expression	[embedded command]
SY	Display a list of synonyms for the word the cursor is on.	[function]
SZ	Type (font) size - e..g., 'sz 12pt' {DF SZ=13PT}	[embedded command, default]
TB 0/1	Tabs-to-spaces on/off - 0 converts tabs to spaces when printing, (i.e., tabs are spaced as they appear on screen); 1 closes up the gap, so that tab(s) appear as one space. Most users will want TB 0.	[function]
TC	Tabs clear (from marker forward)	[embedded command]
TE	Insert a new row of entries in a table	[function]
TF	Move cursor to top of the file.	[function]
TF	Top of form setting {DF TF=0}	[default]
TG	Toggle between expanded view and the view previously displayed.	[function]
TI	Toggle between Insert and the active Overstrike mode.	[function]
TL	Move cursor one column to left in table.	[function]
TM	Time in text (code; updates)	[embedded command]
TN	Toggle numeric lock (doesn't work in NB Win)	[function]
TO	Toggle between Character Overstrike and Insert mode.	[function]
today	Date in text (hard text, does not update)	[immediate command]

topcmd Places embedded command at top of file, [immediate command]
e.g., 'topcmd ts 1in,2in,3in'

TP Toggle between Page Layout View and view previously displayed [function]

TP Top margin for header text, body text [embedded command, default]
{DF TP=3DI,7DI}

TR Move cursor one column to the right in a table. [function]

TR Tab reset (resets tabs to user's default settings) [embedded command]

tree Displays directory tree of drive , as in NB DOS, (but with [immediate command]
rows of 'Ä's instead of the lines that displayed in NB DOS)

TS Toggle Program mode [function]

TS Tab set [embedded command, default]
{DF TS=.5in,1in,1.5in,2in,3in,4in,5in,6in,7in,8in,9in,10in}

TW Switch between Insert mode and Word Overstrike mode. [function]

TW Text width (e.g. 'tw 4.5in') NB DOS command;; [embedded command, default]
NB Win version is PW. {DF TW=0in}

TX +/- mark changes when Track Changes is on. TX+ indicates an [embedded command]
addition/insertion. TX- indicates a deletion.TX ends either region

UA 0/1 How defined text is handled - 0=NB DOS-type persistent selection; [default]
1= Windows-type transient selection {DF UA=0}

UB Use border (around text) [embedded command]

uc Uppercase character under cursor or defined block [immediate command]

UD Restore last text deleted, or activate undelete stack dialog box [function]

UF Use Typeface {DF UF=Times New Roman} [embedded command, default]

UH Sets horizontal unit of measure {DF UH=in,in} [default]

UI Turns on buttons, format bar, and other user interface features: [default]

1 CommandLine
2 StatusLine
3 Buttons
4 FmtBar
5 Ruler
6 MenuBar
7 HorizScrollBars
8 VertScrollBar
9 CommandLinePosition
10 StatusLinePosition
11 ButtonsPosition:
{DF UI=1,2,1,1,0,1,0,1,1,2,0}

UL 0 Underline everything [embedded command]

UL 1 Underline all but tabs [embedded command]

UL 2 Underline all but tabs and spaces [embedded command]

UL 3 Underline only text [embedded command]

UM 0/1 Unhide/hide mode markers. 'd UM=1' makes them appear [default]
as triangles in Draft view and squiggles in Page Layout View.
{DF UM=0}

UN Paste copy from clipboard. (In NB Lingua, an invalid [function]
«XAEnglish» code is inserted as well as the text.)

UP	Use page border	[embedded command]
UP	Delete spaces between cursor and the first non-space character to the left	[function]
US	Use style (as set by ss command)	[embedded command]
UV	Sets vertical unit of measure {DF UV=li,li}	[default]
VA #	Value of variable - shows current value for variable/setting # (see variables list)	[embedded command]
VB	Invokes a Visual Basic routine? (appears in NBMAIN-X.AUX)	[function]
VD	Scroll down one screen	[function]
VU	Scroll up one screen.	[function]
WA	Wild alphanumeric - any single letter/number	[function]
WA	Length of time tooltips are displayed (18=1 sec) {DF WA=36}	[default]
wait	Wait for process to finish (programming). Forces program to wait until preceding command finishes before continuing to execute. Used after commands such as PRINT, SAVE, COPY, which can generate disk activity of an indeterminate duration.	[immediate command]
wc | wcb	Word count forwards/backwards	[immediate command]
WC	Carriage-return wildcard (can be used in searches)	[function]
WD	Widow - minimum number of lines of a paragraph allowed at top of page {DF WD=3)	[embedded command, default]
WF	Makes text wraps to fit window (with DT=0, 1 or 9) {DF WF=1}	[default]
WG	Switch text to [pre-NB8] Draft View (no Page-Line indicators).	[function]
window #	'Window #' goes to window no. # (0-9) 'window #,[left top,width, length]' goes to window no. # and defines its size: left is the column number of the left border (0-80) top is the line number of the top border (0-22). width is the number of columns wide for text (1-80). length is the number of lines of text (1-22).	[immediate command]
WL	Wild letter - any single letter	[function]
WN	Wild number - any single number	[function]
WN 0/1	Auto-renumber window: 0=transient window numbers; 1=fixed window numbers, NB DOS style {df WN=1}	[default]
WO	Word overstrike {DF WO=1}	[default]
WS	Wild separator (any single separator)	[function]
wt	Wait (same as 'wait')	[immediate command]
WT	Line weight (borders)	[embedded command]
WW	Wild within - find up to 80 characters (must be used with at least one character; see Operators section.	[function]
WX	Wild any character - any single character letter, number, separator	[function]

WX Windows extended characters {DF WX=Dutch,Swiss,Courier10,,} [default]
serif
sanserif
monospace
script
decorative

WZ Switch to Page Layout View

X1-X9 TOC/index markers #1 - #9 [embedded command]

XC Execute command that is currently on command line. [function]

XD Release selected block of text, or close and save command window [function]
(func xd will close a footnote or formatting window - same as striking F3)

XD Sets directory to read-only {DF XD=0} [default]

XH Remove any currently displayed Menu or Help screens from view. [function]
(This is now, in combination with function AR, on the Ctrl key in all keyboard states; you can expand an abbreviation by pressing Ctrl.)

XM Move cursor to middle of line [function]

XM Display page-line number and time on status line {DF XM=*PL*TI} [default]

XN 1/2/3/4/5/6 Transpose text - 1, 5 &6=character; 2=word; 3 sentence; 4=paragraph [function]
Transposing characters:
XN1 - If cursor is on a character, transposes current and previous characters
If cursor is on a separator, transposes the two characters preceding the cursor
XN 5 - transpose current and previous characters
XN 6 - transpose the twocharacters preceding the current character.
These can be set in a keyboard file, e.g., ##=XN,5

XP Switch text to expanded view. [function]

XS… Extract (parse) string [embedded command]

XT 1/0 Display/hide message when cursor is on a marker {DF XT=1} [default]

XT Remove entire contents of on-screen file [function]
(Is this what this is meant to do, or just a by-product?)

YD Release selected text, but don't close a command window [function]
(contrast XD)

ZC 1/0 Allow/don't allow upper and lower case for spelling {DF ZC=1} [default]

ZM Zoom page to ##% width {DF ZM=100} [default]

zoom ## Zoom - enlarge/reduce window by ## percent [immediate command]

ZS Point size (sets point sizes displayed in list box for scalable fonts) [default]
{DF ZS=6,7,8,9,10,11,12,14,16,18,20,22,24,30,36,48,72,96,120,144}

Compendium ofXy4/XyWin/NBWin Variables

R.J. Holmgren 5/6/98 LastRev.3/31/06

Note for NB users:This list comes from the file XyWWWeb.INF, which is part of the big XyWrite programming library XyWWWeb.U2. You can download the latest version of XYWEB###.ZIPat the XyWWWeb site: http://www.serve.com/xywwweb/
Files in the zip were written by Robert J. Holmgren and Carl L. Distefano; Robert Holmgren compiled the VAriable list. He has has kindly given me permission to post this standalone version of the VAriables Compendium.
I have not tested these variables in Nota Bene, except for those I myself use in programs, but Robert has removed those that he knows cause trouble in NB.
Note that the second column shows you the current settings of your variables.
It is easier to read this file onscreen in Draft View, without markers: Shift F9, then Shift F10 to toggle through views. ——Mary Bernard April 2006

You can obtain formatting information or the value of DeFaults by entering the appropriate VA command on the command line. E.g., to see the "value" (name) of the current typeface:

VA/NV UF<cr>

The value of << VAUF>> (VAriable UseFont), i.e. the name of the typeface at that cursor position, is displayed on your PRompt line.

Some VAriables report both DeFault values, i.e. those specified at startup, and current values, e.g. << VANW>> and << VA$NW>> (VAriable NewWindow). However, many others use the same two-letter identifier but have different meanings, e.g. << VAET>> (ElementTop) and << VA$ET>> (ElapsedTime)

A few current VAriable values are not displayed, because they are illegal in one or more of the word processors in the XyWrite family, and trigger serious problems.

VA @100	-1	Contents of S/G 100...
VA @1999	-1	...Contents of S/G 1999
VA @*1	-1	Current **<<** CP**>>** CharPos [*0001]
VA @*5	-1	Last CoPied|MoVed Text [*0005]
VA @*11	-1	Recorded Keystrokes (func RK) [*000B]
VA @*26	-1	First (or most recent) running program or frame [*001A]
VA @*27	-1	Second running program or frame [*001B]
VA @*29	-1	? Next running program or frame [*001D] (increments)

```
VA @*31      -1                    ? [*001F]
VA @*213     -1                    Second LDPM program not
                                    assigned to a key
                                    [*00D5] (decrements)
VA @*214     -1                    First LDPM program not
                                    assigned to a key
                                    [*00D6]
VA @*248     -1                    &0 LDPM program [*00F8]
                                    (increments to...
VA @*257     12          BC se //CL □       ...&9 LDPM
                                    program [*0101]
VA @*265     -1                    &A LDPM program [*0109]
                                    (increments to...
VA @*290     -1                    ...&Z LDPM program
                                    [*0122]
VA @*13841   -1                    Current running program
                                    frame [*3611]
VA @*13843   -1                    KBD layout [*3613]
VA @*13844   -1                    First MeNu file selec-
                                    tion [*3614]
VA @*13845   -1                    Second MeNu file selec-
                                    tion [*3615] (incre-
                                    ments)
VA @*13853   -1                    ? [*361D]
VA [it       0           Additive MoDe On (e.g."IT") at
                                    current << CP>>
VA \902      "Select a style first." Error Message 902
VA \@01                 Error Message from S/G, e.g.
                                    << SX01,<< VA$ER>> >>
                                    << PR\@01>>
VA !01       255         Flag Value (initialization type)
                                    for S/G: 0=$tring
                                    (SV|SX), 2=SUb, 4=SX
                                    value, 16=expression
                                    evaluates FALSE,
                                    24=expression evaluates
                                    TRUE, 255=nonexistent
VA ”01       0           S/G Contains Only Numbers 0|1=Yes
VA |01       -1          Size of S/G
VA *ul       1           MoDe number of MoDe mnemonic, e.g.
                                    << MDUL>>
VA _bc       Ctrl+E      (First) Location of keyboard
                                    assignment, e.g. BC
VA œ13       thirteen    (Xy4 adds " dollars")
VA <m1#u1                Unit of Measure Conversion
                                    (<measure#unit), e.g.
VA <PT#2IN   144         PoinTs/2 inches (72pt/in)
VA <m1]va                VAriable expressed in
                                    Measure(<measure]variab
                                    le) ("]" = system
                                    default), e.g.
VA <PT]SZ    13,13       PoinT SiZe, 1st and 2nd elements
```

VA <*m1va*	VAriable expressed in Measure(<measurevariable) (without "]"=*current value*), *e.g.*
VA <PTSZ1 12	PoinT SiZe, 1st element only *or*
VA <INIP 0,3.6,0	(whereas standard **<<** VAIP**>>** =0,3.6,0 expressed in Deci-Inches [=**<<** VA<DIIP**>>**])
VA *command#* 3.6	Value of Command Element, where "command"=embedded command or default, "#"=element within command, e.g. **<<** VAIP2**>>** =3.6
VA (*cmdname,var*	COURIER NEW Value of Nested Command: "cmdname"=BOlabel|FAlabel|FM12or3|IGfilename|SSstylename,"var"=variable|element|keyword to solicit, e.g. **<<** VA(SSCompendium,UF **>>** ="COURIER NEW"
VA □*dfbit*	Bit status within default setting, where "df"=default name, "bit"=value to check; 0|1=On, e.g. **<<** VA□HD4**>>** =1
VA {*var* è	GC variable value
VA =*filename,$tring=*	**<<** VA=C:\AUTOEXEC.BAT,SET PATH=**>>** Search String: returns text between <cr>search$ and next <cr> (EOL), e.g. "**<<** VA=G:\XY4\XYWWWEB.REG,Comspec_W2K=**>>** " returns "C:\WINNT\SYSTEM32\CMD.EXE"
VA ^*mm*	Document Information: Summary Item
VA ^AU	- Author
VA ^CD	- Creation date
VA ^CM	- Comments
VA ^CT	- Creation time
VA ^KY	- Keywords
VA ^LG	- Last revisor
VA ^MD	- Modified date
VA ^MT	- Modified time
VA ^PJ	- Project Number
VA ^RP	- Retention Period
VA ^RV	- Revision Number

```
VA 1A       0           Ignore End-of-File Marker
                          0|1=ignore EOF byte
                          (Ascii-26)
VA $1A      0           Ignore End-of-File Marker
VA 1O       0,0         ? (NB)
VA 1X       3           ? (NB)
VA $1X      3           ? (NB)
VA 3D       1           Three-Dimensional Effects:
                          appearance of dialog
                          boxes 0=2D|1=3D
                          (Windows)
VA $3D      1           Three-Dimensional Effects
                          (Windows)
VA $AC      0           Auto-Correct 0|1=On
VA AE       1           ? (NB)
VA $AE      1           ? (NB)
VA AF       0,0         ? (NB)
VA AH       0           Allow Hyphenation
VA $AH      0           Allow Hyphenation
VA AL       1           Automatic Leading
VA $AMMoDe              Available MoDes (BI|BO|IT) in cur-
                          rent type family
                          (0|1=available):
VA $AMbi    0           Bold Italic
VA $AMbo    0           BOld
VA $AMit    0           ITalic
VA $AN      1           NBWin: Display command brackets as
                          0=Registered symbol
                          {&reg} and long macron
                          {&macr} (i.e. ANSI 1252
                          codes
                          174/175)|1=<< >> (ANSI
                          171/187) [immediate
                          command D AN=#]
VA AOP      C:\NB\AUTOSAVE\AUTOSAV1.TMP     AutOsave Path
VA AOT      1,1         AutOsave Timer (min[,max
                          default=min+5] in
                          minutes)
VA AP       0           Auto-Pause
VA $AR      0           Auto-Replace 0|1=On
VA $AT      0           ATtribute value returned by last
                          ATTRIB command
                          (0=R/W|1=RO)
VA AX       256         ? (NB)
VA $AX      256         ? (NB)
VA AZ       0           Counter Numbering Style: 0= ... x
                          y z aa bb cc ... xx yy
                          zz aaa bbb ...; 1= ...
                          x y z aa ab ac ... ax
                          ay az ba bb bc ..
VA $AZ      0           Counter Numbering Style
VA BC       0           ?
VA $BD      0           BaD Words
VA BF       0           Bottom Footnote
```

```
VA BG       255,255,255  BackGround color
VA BI       0            Beep Inhibit: 0|1=Display any
                                 format□ without error
                                 beep
VA $BI      0            Beep Inhibit
VA BK       1            BacKup Files 0|1=keep BAcKups
VA BL       28           BLank Lines; NB: Base Line
VA BM       1            ? (NB)
VA $BM      1            ? (NB)
VA BN       0,0,0,0,0    B u t t o N  D e s c r i p t i o n :
                                 F a c e , W i d t h , H e i g h t
                                 (where Face is 0=Pic-
                                 ture, 1=Text, 2=Both)
                                 (Windows); ? in Xy4
VA $BN      0,0,0,0,0    ButtoN Description (Windows); ? in
                                 Xy4
VA $BQbo    0            Border Query 0|1=definition is
                                 present
VA BS       1            Backspace Control
VA $BS      1            Backspace Control
VA BT       .5,.5,.5,.5  Bottom Margin
VA $BT      0            Black and White Trace: value of BW
                                 command
VA BW       26707        Black and White (for CGA monitors)
                                 0|1
VA $BW      1            Monitor Type: Black and White
                                 0|1=Color
VA BX
                                 0,0,5,63856,0,0,0,0,1,1,0
                                         Window Border
                                 Colors
VA $BX
                                 0,0,5,63856,0,0,0,0,1,1,0
                                         Window Border
                                 Colors
VA BZ       0            Select Displayed Button Set
                                 (Windows)
VA $BZ      0            Select Displayed Button Set
                                 (Windows)
VA $C#      0            NB: C#=0-9 CodePage default(?)
VA $CA      (none)       Cartridges currently loaded
VA CB       0,4096       Correction Beep: Xy4 values are
                                 frequency,duration; NB
                                 0|1=Off
VA $CB      0,4096       Correction Beep
VA CF       1            Change Footnote Separator: 0Äuse
                                 series 1 separator
                                 (even if no series 1
                                 notes); 1Ästart with
                                 separator for first set
                                 of notes used
VA $CF      1            Change Footnote Separator
VA CH       0            ?
VA $CH      0            ?
```

```
VA CK       3             Spelling Checker: 0|1=ignore words
                                that contain a number
VA $CK                    Spelling Checker
VA $CL      C:\NB\QSHAXBUC.TMP              Command Line
                                last issued (40 chars
                                max)
VA $CM
                                                 Current con-
                                tent of CoMmand Line
                                (80 chars max)
VA $CN                    Cartridge INstalled (Xy4)
VA CO       0             COlumns
VA $CO      0             COlumns
VA $CP      850           Operating System Code Page (cf.
                                LAnguage DeFault)
VA CR       1,0,0,5,255,0,0        Cursor Type
VA $CR      □ □           Carriage Return character(s)
VA CT       0             ?
VA $CT
                                ,,,,,,,,,,,,,,,,,,,,,,,,"
                                        Column Style:
                                name of style used for
                                current column (if
                                specify << VA$CT5>> ,
                                returns style used in
                                5th column)
VA CV       0             Change Verification Prompt
                                0|1=Confirm CHange com-
                                mands
VA $CV      0             Change Verification Prompt
VA CW       60            Value of Margin Units MU (set in
                                PRN)
VA $CW      60            Value of Margin Units MU (set in
                                PRN)
VA $CX      18            Cursor Column Position
VA $CY      0             Cursor Row Position
VA $CZ      0             DDE Conversation Number: value of
                                highest conversation
                                currently active
VA D1       0,5           Delete Stack: # of
                                entries,min.chars con-
                                sidered deletable unit
VA $D1      0,5           Delete Stack
VA $DAd.Mmmm.yyy          16.July.2006        Embed in text;
                                displays date in
                                specified <<
                                VA$DAformat>>
VA DB       0,0           Debug a Program
VA $DB      0,0           Debug a Program: Stop on
                                0|1=<< IF|2=<< LB|4=JM
                                |8=<cr>|16=<< ER>> ,Ign
                                ore 0|1=<cr>|2=|3=both
VA $DC      0             Define Column currently selected
                                0|1=yes
```

```
VA DD       8             Display Selected Blocks
VA $DD      8             Display Selected Blocks
VA DE       ?D            Define "Soft" End-of-Line Charac-
                                  t e r s  V A D E 1
                                  ( ? ) = a c t u a l | V A D E 2
                                  (D)=visual
VA $DE      0             Define Ended 0|1=yes
VA $DF      0             DeFine Status 0|1=currently
                                  selected
VA $DG      C:\NB\NB.DLG DialoG File location
VA DH       -             Discretionary "Soft" Hyphen
VA $DH      -             Discretionary "Soft" Hyphen
VA DI       1,6,0         Long DIrectory Display x,y,z
                                  (x=filesize divisor;
                                  y=lines of text dis-
                                  played; z=0|1 (remove
                                  <cr>s)
VA $DI      0             DIrectory Type in current window
VA $DK      1023932928    NB: Date of "K"reation, current
                                  file: octal date+time
VA $DL      C:\NB\USERS\DEFAULT\NB.DFL      DeFauLt File
                                  Location
VA $DN      0             Define ENd << CP>>
VA $DO      0             DOcument Information attached to
                                  file 0|1=yes
VA DP       .             Decimal Point (USA="." Europe=",")
VA $DP      C:\NB\INBOX\CPG\CPG.NB D i r e c t o r y  P a t h  i f
                                  << VA$WS>> =2; other-
                                  wise=<< VA$FP>>
VA DR       C:\NB\        DRive:\Path\ for Temporary Files
VA $DR                    F i l e  a t  c u r s o r  p o s i t i o n  i n
                                  c u r r e n t l y - d i s p l a y e d
                                  DiRectory
VA $DS      0             Define Start << CP>>
VA DT       4             D i s p l a y  C u r r e n t  T y p e
                                  (0|1|2|4|8|9|10|12|17|1
                                  8|20)
VA $DT      4             Display Type
VA DU       60            Display Units
VA $DU      60            Display Units
VA $DV      C             Current DriVeletter
VA DY       0,0           Dye: activate color printer con-
                                  trol codes
VA DZ       d Mmmm yyyy   Date Format of DA and TODAY com-
                                  mands
VA $DZ      d Mmmm yyyy   Date Format
VA EB       0,36864       Error Beep
VA $EB      (none)        Error Beep
VA EC       0             ?
VA $ED      C:\NB\SWSYS.DLL       XyWrite Editor location
VA EE       0,0           Element End Margin Offset
VA EF       0             Special EFfects (immediate com-
                                  mand)
```

```
VA EG       0             IBM EGA Control: 0=25 lines, 1=43
                                lines B&W, 2=43 lines
                                Color
VA $EG      0             EGA Control
VA EH       0             Error Help
VA $EH      0             Error Help
VA EJ       0             Eject Last Page: 0Äleave last page
                                in printer/1Äeject last
                                page
VA $EJ      0             Eject Last Page
VA EL       0             Extra Leading on current line, in
                                INches*PoinTs/INch
                                (e.g. 0*72)
VA $EL      0             ELement ID: internal ID of the
                                last element clicked on
VA EP       0,1,1,0,0,0,0,0,0      Error Prompt: DEL, TY
                                DeFined-block, TY dir,
                                ABort, func SA, del
                                deltas, screen:printer
                                font mismatch [Added in
                                NB: replace with COR-
                                RECT command in batch
                                spell, Search/Replace]
VA $EP      0,1,1,0,0,0,0,0,0      Error Prompt
VA $ER      214           Last ERror Number
VA ES       0             Enable Scoping (apply format com-
                                mands from <CP>=0, cur-
                                rent paragraph=1,
                                replace previous=2)
VA $ES      0             Enable Scoping
VA ET       0,0           Element Top
VA $ET      8345950       Elapsed Time since ZT issued, else
                                current time in format
                                hh:mm:sec.hundredths.
                                Broken in NB
VA EU       .,:,;         EUropean Punctuation
VA $EU      .,:,;         EUropean Punctuation
VA $EX      NB            Current File .EXtension
VA F2       0             ? (NB)
VA $F2      0             ? (NB)
VA $FA      0             Frame Attribute: internal ID of
                                last frame clicked on
VA $FB      1             File Begin: cursor at TOF 0|1=yes
VA FC       FL            Current value of FL|Flush Cen-
                                ter|FR
VA $FC      0             Font Count: how many fonts avail-
                                able in current printer
                                file
VA FD       11            Form Depth
VA $FE      0             File End: cursor at EOF 0|1=yes
VA FF       0             Form Feed
VA FG       0,0,0         ForeGround color
VA FH       0,0,0,0       Format Bar Height (Windows); ? in
                                Xy4
```

```
VA $FH      0,0,0,0      Format Bar Height (Windows); ? in
                                 Xy4
VA $FI[/F]  CPG.NB       Current FIlename [/F=long filename
                                 NB]
VA FL       FL           Current value of Flush Left|FC|FR
VA $FM                   Forms Mode: is current file a
                                 form? 0|1=yes
VA $FP      C:\NB\INBOX\CPG\CPG.NB Drive:\Path\Current
                                 Filename
VA FQ                    Format Bar Queue: items to be dis-
                                 played on format bar
                                 (Windows)
VA $FQ                   Format Bar Queue: items to be dis-
                                 played on format bar
                                 (Windows)
VA FR       FL           Current value of FL|FC|Flush Right
VA $FR      0C           Last FRamename called
VA $FS      1            File Status: 0=no files open; non-
                                 zero=at least 1 file
                                 open
VA FT       0            Footnote Transition
VA FU       1,3          Footnote Unit
VA $FU      1,3          Footnote Unit
VA FV       0            ? (NB)
VA $FV      0            ? (NB)
VA $FW      0            Full Screen Window: status of
                                 window (0=not FS|1=FS)
VA FX       ?            Field Separator in data files
VA $FX      1            Fixed Pitch=0|Proportional=1 (cur-
                                 rent font)
VA $FY      sixteen      Font FamilY for current font
                                 1|2|3|4|5
VA FZ       DDD.MMM.YY   File Date format in DIRectories
VA $FZ                  Field Separator
VA GA       MD           Graphic Adapter
VA $GA      MD           Graphic Adapter
VA GB       C:\nb\support\debug\SWGS.LIB    Global Library
                                 file location (Windows)
VA $GC      0            GCI Status
VA GG                    Location of U5 file (General
                                 Counsel)
VA GH       6            ? (NB)
VA $GH      6            ? (NB)
VA $GMformat             Mode Status in format, e.g.
VA $GM1     1            Printing a file
VA $GM2     0            Waiting for a + to continue print-
                                 ing
VA $GM8     1            Insert mode On
VA $GM16    0            Waiting for printing
VA $GM32    0            Printing suspended
VA $GM64    1            Message is displayed
VA $GM128   0            Spell or search has highlighted a
                                 string
```

```
VA $GM256    0              Automatic Uppercase On
VA $GM2048   0              Format bar being built
VA $GM4096   0              Sets internal flag for redisplay-
                                  ing a page
VA $GM8192   0              Running program
VA $GM16384  1              Executing BX function
VA GP                       Graphics Drive:\Path\ (Xy4)
VA $GP                      Graphics Drive:\Path\ (Xy4)
VA GU        0,.1           GUtter (current value)
VA GV        0,0,0,0,0,0,0          Graphics Variable (set
                                     in PRN file)
VA $GV       0,0,0,0,0,0,0          Graphics Variable
VA GW        0              ? (NB)
VA $GW       0              ? (NB)
VA GX
                                     0,0,255,255,0,0,255,0,0,2
                                           ?
VA $GX
                                     0,0,255,255,0,0,255,0,0,2
                                           ?
VA GZ        0              ? (NB)
VA $GZ       0              ? (NB)
VA HB        0              Header Wildcard color (3rd element
                                  of L1 command)
VA $HB       112            Header Wildcard color
VA HD        764            Xy4: Fixed drives; NB: unknown
                                  numeric value
VA $HD       764
VA HI        %F             Header info
VA $HI       << VA$HI>>     Header info
VA HL        0              Help Construction
VA $HL       (none)         Help File location
VA HM        0              ?
VA HN        7              Header Normal display mode (2nd
                                  element of L1 command)
VA $HN       7              Header Normal display mode
VA $HP                      Hewlett-Packard Printer File
                                  loaded 0|1=Yes (Xy4)
VA HR        112            Header Reverse display mode (for
                                  CM) (1st element of L1
                                  command)
VA $HR       112            Header Reverse display mode
VA HS        16             ? (NB)
VA $HS       16             ? (NB)
VA HT                       Header title (DocInfo) (Xy4)
VA $HT                      Header title
VA HV        6,3,3          Hyphenation Values
VA $HV       6,3,3          Hyphenation Values
VA HY        1              Hyphenation 1=On (responds only to
                                  embedded << HY1>> , not
                                  to "d hy=1")
VA $HY       C:\NB\USERS\DEFAULT\MAIN.HYP    Hyphenation
                                  dictionary file (Xy4)
VA ID                       ? (NB)
```

```
VA $ID      0             ? (NB)
VA $IG      3             Import Graphics: number of IG com-
                                mands in current file
VA II       0             I t a l i c  I n f o r m a t i o n
                                font=0|attribute=1
VA $II                    I m a g e  I n f o r m a t i o n :  returns
                                information about an
                                image (compression,
                                color, depth, width and
                                height)
VA IM                     Image Mode Printing
VA $IM      2             Image Mode Printing
VA $IN      0             Cursor Inside Define 0|1=yes
VA IO       0             D o c u m e n t  I n f o r m a t i o n :
                                0|1=On|2=Enter comments
VA $IO      0             Document Information
VA IP       0,3.6,0       Indent Paragraph; 3rd param (Xy4)
                                is right indent
VA IT       519,1543,8193,264,1543,8193,0,1  Insert Cursor
                                Type
VA $IT      519,1543,8193,264,1543,8193,0,1  Insert Cursor
                                Type
VA IU       0             Information MenU 0|1=store file
                                after clearing DocInfo
                                dialog box
VA $IU      0             Information MenU
VA IW       0             ? (NB)
VA JB       0             Send PC Codes at Job Begin
VA $JBPCCode#             0         Job Begin PC code# set
                                by a JB command, e.g.
                                << VA$JB34>> returns
                                0|1=specified
VA $JC      1             Number of Journal entries in file
                                (Windows)
VA JE       0             ? (NB)
VA JL       1             Justify UnderLine Characters
VA $JL      1             Justify UnderLine Characters
VA JR       1             Journaling: 0=update existing
                                j o u r n a l s | 1 = m a i n t a i n
                                j o u r n a l s  b u t  d o  n o t
                                save|3=create journals
                                for new files (maintain
                                & save)
VA $JR      1             ? (NB)
VA $JS      31            Size of Journal in bytes (Windows)
VA JT       0             Justification Type
VA $JT      0             ? (NB)
VA JU       JU            JUstification|NJ
VA JZ       1             Job End 0|1|2|3
VA $JZPCCode#             0         Job Element 0|1=active
                                s e t t i n g  f o r  P C  C o d e
                                r a n g e ,  e . g .
                                << VA$JZ120>>  (Xy4); ?
                                NB
```

```
VA $KB      C:\NB\USERS\DEFAULT\NEW.KBD     KeyBoard File
                                            location
VA KC       0,0           Key Click
VA $KC      41            Key Code
VA KP       0             (In NB: Special KayPro laptop=1)
VA $KP      0             (as above)
VA $KR      0             Keystrokes Recorded: number of
                                            keystrokes saved
VA KS       0,0           Keyboard [cursor] Speed
VA $KS      0,0           Keyboard [cursor] Speed
VA L0       112,7,1,7     Menu bar color control
VA $L0      112,7,1,7     Menu bar color control
VA L1       112,7,112,15  Command line color control
VA $L1      112,7,112,15  Command line color control
VA L2       112,112,112,176,224    Status line color con-
                                            trol
VA $L2      112,112,112,176,224    Status line color con-
                                            trol
VA L3       7,112,7,15    Ruler line color control
VA $L3      7,112,7,15    Ruler line color control
VA L4       15,112,1,7,15          Pull-down menus color
                                            control
VA $L4      15,112,1,7,15          Pull-down menus color
                                            control
VA LA       English,A     LAnguage: Xy4 current Code Page
                                            value 437|850; NB lan-
                                            g u a g e   n a m e
                                            (e.g.<< LAEnglish>> )
VA $LB      0             ? (NB)
VA LC       ¶             Line End Character
VA $LC      ¶             Line End Character
VA $LE      0             ? (NB)
VA LF       0             (May be NB only & disabled in
                                            XyWrite; performs like
                                            VAWF)
VA $LF      0             (as above)
VA LG       1             Logic state (GC)
VA $LG      Mary          Logged-On User
VA LH       67,85,60      Low-High super/subscript control
                                            for Speedos
VA $LH      67,85,60      Low-High super/subscript control
                                            for Speedos
VA $LI      C:\NB\SUPPORT\DEBUG\TEXT.LIB    LIbrary file
                                            (NB)
VA $LK      0             ? (NB)
VA LL       0,0           Line Leading
VA LM       .1            Left Margin
VA LN       0             ? (NB)
VA $LN      9.43          Current Line Number in P-L mode
VA $LO                    Previous logical condition at same
                                            level as current level
                                            (GCC)
VA LQ       0             Letter Quality 0-9
```

```
VA LR       1              Left-to-Right Mode (for Hebrew
                                 etc) L2R=1;R2L=0
VA LS       .25            Line Space
VA $LT      1              Logon Notes Toggle
VA $LV      0              ? (NB)
VA LX                      Main LeXicon path (LEXAM) (Xy4)
VA $LX                     Main LeXicon path (LEXAM)
VA $LV      0              Current level of display (GCC)
VA LZ       Mmmm d, yyyy   Format Redlining Date
VA $LZ      Mmmm d, yyyy   Format Redlining Date
VA MA       40             # of chars to Find MAtch
VA $MA      40             # of chars to Find MAtch
VA MB       0              Message Box display location
                                 (0=status line, 1=mes-
                                 sage box) (Windows)
VA $MB      0              Message Box
VA MC       0              Minimum Size to Add Command to
                                 Stack: threshold mini-
                                 mum size (Windows)
VA $MC      0              Minimum Size to Add Command to
                                 Stack (Windows)
VA MD       NM             Current Character MoDe
VA ME       1              MEnu editing for deltas 0=command
                                 window|1=dialog box
                                 (Windows)
VA $ME      65525          Available MEmory
VA -M-                     Main Dictionary memory
VA -e-                     Expanded Dictionary memory
VA $M+      65437          Memory Used by XyWrite
VA $M+0     65437          All XyWrite memory
VA $M+1     65211          All Code memory
VA $M+2     65165          All Overlays memory
VA $M+3     47             Root memory (cseg)
VA $M+4     65391          Editor code data memory
VA $M+5     372            Data memory
VA $M+6     4              Save/Gets program memory
VA -P1      65475          Load program memory
VA -M1                     Load file memory
VA -P2      372            Math/Program program memory
VA -M2                     Math/Program file memory
VA -P3      10             Spell program memory
VA -M3                     Spell file memory
VA -P4      41             Help program memory
VA -M4                     Help file memory
VA -P5      1              Hyphenation program memory
VA -M5                     Hyphenation file memory
VA -P6                     Sorting program memory
VA -M6                     Sorting file memory
VA -P7                     Printing program memory
VA -M7                     Printing file memory
VA -P8                     Graphics program memory
VA -M8                     Graphics file memory
VA -P9      1              Directory program memory
VA -M9                     Directory file memory
```

```
VA -pa      1           Load printer program memory
VA -ma                  Load printer file memory
VA -pb      1           Search program memory
VA -mb                  Search file memory
VA -pc      1           Redline memory
VA -pd                  Box drawing memory
VA -pe                  Call/Save memory
VA -pf      3           Counters memory
VA -pg                  Memory manager memory
VA -ph                  Command table memory
VA -pi                  Menus memory
VA -pj      1           Error messages memory
VA -pk      1           WYSIWYG memory
VA -pl      1           Styles memory
VA -pm                  Soft fonts memory
VA -pn                  Image memory
VA -po      1           VGA memory
VA -pp                  HGC (Hercules) memory
VA -pq                  CGA memory
VA -pr                  Network memory
VA -ps                  GCI memory
VA -pt                  Scaling memory
VA -pu                  Rasterizer memory
VA -pv                  RFT:DCA Import memory
VA -pw      12          RFT:DCA Export memory
VA MF       255         Mode for Forms
VA $MF      255         Mode for Forms
VA MG                   Current MessaGe
VA $MG                  Current MessaGe
VA MH       59          ? (NB)
VA $MH      59          ? (NB)
VA MK       0           ? (NB)
VA $MK      0           ? (NB)
VA $MN      (none)      MeNu File location
VA $MO      0           File MOdified 0|1=yes
VA MR       0           Metric Ruler
VA $MR      0           Metric Ruler
VA MS       60          Microspace Factor
VA $MSmd                Mode Status:
VA $MS1     0           Document Information dialog box
                                displayed on STore|SAve
VA $MS2     0           Scroll Lock on
VA $MS4     0           No file open
VA $MS8     0           [Not used]
VA $MS16    0           Text selection started and NOT
                                ended
VA $MS32    0           Selected text is on screen AND
                                ended
VA $MS64    0           File open for read only
VA $MS128   0           Command window open
VA $MS256   0           Redlining on
VA $MS512   0           Directory displayed
VA $MS1024  0           Current file has never been saved
```

```
VA $MS2048   0          Current file has been edited since
                                last saved
VA $MS4096   0          Insertion point in file
                                (text=0|header=1)
VA $MS8192   0          Column selected
VA $MS16384  1          Current file includes Document
                                Info (0=Yes|1=No)
VA $MS32768  0          REVIEW.TMP file (created by
                                PRINTS|TYS) displayed
VA MT                   Military Time 0|1=Use MT
VA $MT       0          Military Time
VA MU        60         Margin Unit
VA $MU       60         Margin Unit
VA MW        0          Maximize Windows
VA $MW       0          Mouse Window number (current loca-
                                tion of mouse) (Xy4)
VA MX        67         RAM committed to DICT.SPL
VA $MX       0          Mouse X pixel|row position (Xy4)
VA MY        8,Helv     MagnifY Dialog Boxes: specify SZ
                                and Windows font
                                (Windows)
VA $MY       0          Mouse Y pixel|row position (Xy4)
VA $MZmd                Mode Status:
VA $MZ1                 Forms mode
VA $MZ2                 Put block cursor on menu
VA $MZ4                 Don't put accelerator on `TC'
VA $MZ8                 We are creating a new file
VA $MZ16                Set indicates we were editing a
                                previously entered com-
                                mand in a command
                                window. Cleared means
                                we are entering a com-
                                mand for the first time
VA $MZ32                Need to read the bottom of the
                                file
VA $MZ64                Tabular row define
VA $MZ128               Tabular column define
VA $MZ256               Screen in a menu or help screen
VA $MZ512               Menu is a sidebar
VA $MZ1024              No borders on screen
VA $MZ2048              Window has accelerators
VA $MZ4096              Radio button
VA $MZ8192              List box or list directory
VA $MZ16384             We can edit this menu
VA $MZ32768             Window is part of dialog box
VA $NA       0,0,0,0    Non-Printable Area
VA NB        0          ?
VA NC        1          Normal Carriage Return
VA $NC       0          ?
VA ND        65535      Network Drives
VA $ND       65535      Network Drives
VA NE        0          No Errors from Printer: 0|1=ignore
                                spurious errors
VA $NE       0          No Errors from Printer
```

```
VA $NF      0           ? (NB)
VA NI       0           No Index: suppress printing of
                                  indices
VA NJ       JU          No Justification|JU
VA NL                   Network Login Path
VA $NL                  Network Login Path
VA NM       0           No Modification Mode: 0|1=text
                                  marked with
                                  << NM1>> ...<< NM0>>
                                  commands can't be
                                  changed
VA NP       0,120       No Pause
VA $NR      1           No Ruler (Xy4)
VA $NU                  UNused Printer Memory (Xy4)
VA NW       1           New Window
VA $NW      1           New Window: 0=no auto windows;
                                  1=auto, no ABort if
                                  CAll in DIRectory dis-
                                  play; 3=auto, ABort if
                                  CAll in DIRectory dis-
                                  play
VA NX       0           ? (NB)
VA $NX      0           ? (NB)
VA O1       25601       Options: Error correction between
                                  screen<>printer fonts
VA $O1      25601       Options
VA OB       << VAOB>>   Overstrike Beep 0|1=On (Xy4)
VA $OB      << VA$OB>>  Overstrike Beep
VA OC       0,1707      OCtagon Control: define shape of
                                  radio buttons (Windows)
VA $OC      0,1707      OCtagon Control
VA OD       2           Offset Display 0-7
VA $OD      2           Offset Display
VA OE                   Open Editor on LAN 0|1=Keep Editor
                                  open after reading
                                  shared code (faster
                                  performance)
VA $OE      0           Open Editor on LAN
VA OF       1,1         OFfset
VA OH       1           ?
VA $OH      1           ?
VA OL       0           OutLine Fonts Drive:\Path\
VA $OL      0           OutLine Fonts Drive:\Path\
VA OM       31          Old Masticon (PCLEX=0;
                                  Microlytics=31)
VA $OM      31          Old Masticon (PCLEX=0;
                                  Microlytics=31)
VA $OO      0           Command Override On 0|1=On (Xy4)
VA OP       3           Number of lines for OrPhans
VA OR       0           ORientation
VA OS       1           One-Sided Printing (can also embed
                                  << OS0>> |<< OS1>> )
VA $OV      C           Overflow File Drive (if overflow
                                  file exists)
```

VA $P?*PCCode#*	0	P? returns 1 if specified PC code exists in PC table, e.g. **<<** VA$P?120**>>**
VA P.	14	Truncated Path
VA $P.	C:\nb\inbox	Truncated Current Drive:\Path Name (14 chars max)
VA $P\	C:\nb\inbox	Current Path, add Backslash "\" in Root Directories
VA $PA	C:\nb\inbox	Current Drive:\Path
VA $PB	1	Bottom Depth defines command controlling text length (0=Page Length PL|1=Bottom Margin BT)
VA $PC*PCCode#*	1	PC returns 1 if specified PC code exists in PC table, e.g. **<<** VA$PC34**>>**
VA $PC	1	PC Code returns 1 if specified PC code exists in PC table
VA PD	0	Pad Spaces
VA $PD	0	Pad Spaces
VA $PE	**<<** VA$PE**>>**	Page Elements: number of page elements in current file
VA $PF	(none)	Current Printer File (SETP) selection: $PF1=port#, $PF2=filename, $PF3=description
VA $PF7	1	Current SETP setting (number only)
VA PG	0,0,0	?
VA $PG	147	Current Page Number in P-L mode
VA PK	255	Page Break Color
VA $PK	255	Page Break Color
VA PL	10.5,10.5,10.5	Page Length
VA PM	0,0,0,0,100,0	Printer Memory (set in PRN file)
VA $PM	0,0,0,0,100,0	Printer Memory
VA $PR	(none)	Printer File location
VA PT	0	Numeric Print Type
VA PW	8.25	Page Width
VA PX	45	Page Break Character
VA $PX		Page Break Character
VA $PX*PCCode#*		*In Dialog Box only*: PC Code Explanation
Quote Type		NB.INI [General Settings]: 0|1="Smart" Quotes
VA QC	0	? (NB)
VA $QC	0	? (NB)
VA $QF	1	? (NB)
VA R2	8	Mouse Double Click
VA $R2	8	Mouse Double Click

```
VA $RA      0             Read Attribute displays absolute
                                number of current
                                character MoDe
VA RB       0             Reverse Buttons
VA $RB      0             Reverse Buttons
VA $RC      0             Resume Code (response to BR|WM
                                commands: 0=first valid
                                key, 1=second valid
                                key,...; 65533=Gray-
                                Enter; 65534=F9 or
                                Ctrl+Break; 65535=Esc)
VA RD       1,14,250,0,249          Redline Data
VA $RD      1,14,250,0,249          Redline Data
VA $RE      0             0|1=Read-Only File
VA RG       0             ?
VA RI       5             Mouse Repetition Interval
VA $RI      5             Mouse Repetition Interval
VA $RK      0             Record Keystrokes 0|1=On
VA RL       ??ʒ?pù????????□?       Ruler Markers
VA $RL      0             Redlining 0|1=On
VA RM       0             Right Margin
VA RN       0             Round Off Line Numbers to nearest
                                .5 in Page-Line display
                                0|1=On
VA $RN      0             Round Off Line Numbers
VA RO       17996         ? (NB)
VA $RO      LFDECOTFTEBF  ? (NB)
VA RP       1             ? (NB)
VA $RP                    ? (NB)
VA $RR      0             Return eRRor from DOS if shell
                                with DO command
VA RS       ??            Record Separator in data file
VA $RS                    Read Character at cursor position
VA RT       1             Relative Tabs
VA RX       8             Ratio for X Direction
VA $RX      8             Ratio for X Direction
VA RY       8             Ratio for Y Direction
VA $RY      8             Ratio for Y Direction
VA RZ       10000         Record SiZe (max chars for data
                                files)
VA $RZ      □ □           Record Separator in data file
VA SB                     ?
VA $SB      0             Scalable Fonts available 0|1=Yes
VA SC       247           Superscript Footnote Numbers
VA $SC      1577          Scan Code of last key pressed
VA SD       .166          Separator Depth (spacing between
                                text & note separator):
                                0-put out current line
                                spacing before separa-
                                tor [use only if you
                                have modified printer
                                driver]; 1-use fixed
                                (single) line spacing
                                before separator
```

```
VA $SD      D,R          Sort Directory
VA $SE       /           SEarch $tring last sought
VA $SF      (none)       Soft Font List File
VA SG       0            Assign Save/Get,$tring
VA $SG      C:\NB\USERS\DEFAULT\ORD.LIB     Macro LDSGT
                                 File location
VA SH       0,0,0        ?
VA SI       96,96        Screen Resolution
VA $SI      96,96        Screen Resolution
VA SK       1,80         Sort Key: 0=letter-by-letter
                                 ("Newark" before "New
                                 York"); 1=word-by-word
                                 ("New York" before
                                 "Newark"); 2=reverse
                                 sort; 4=delete dupes;
                                 n2=number of chars to
                                 sort
VA $SK      1,80         Sort Key
VA SL       25           Screen Length
VA $SL      << VA$SL>>   Screen Length (Xy4); Item in list
                                 box has been selected
                                 0|1=1 item (Windows)
VA SM       0            Show Menus
VA $SM      0            Show Menus: menu currently dis-
                                 played 0|1=Yes
VA SN       0            Snaking Columns
VA $SN      0            ? (NB)
VA SO       F1           Sort Order Setting
VA $SO      F1           Sort Order Setting
VA SP       1,0,49       ?
VA $SP      C:\NB\USERS\DEFAULT\SHORT.SPL   Personal Dic-
                                 tionary location
VA SQ       0            Sequential Page#: 0=respect SP
                                 command; 1=override SP
                                 and use actual pages in
                                 text
VA $SQ      0            Sequential Page#
VA $SSStyle#             IX       Style name defined in
                                 document, e.g.
                                 << VA$SS2>>  returns
                                 2nd style
VA $#S      COMPENDIUM   Current Style name
VA ST       3            Show Tabs (0|1|2|3)
VA $ST      C:\NB\NBSTART.INT     STARTUP.INT location
VA ?ST                   NB: Command STack
VA SW       0            Screen Width
VA $SW      0            Screen Width
VA SY                    SYmbol Set (Xy4)
VA $SY                   SYmbol Set for current font
VA SZ       .166,.166    SiZe: point size of font in use
```

```
VA TB       0             TaB Control 0=expand as
                                spaces|1=output
                                directly
VA $TB                  TaB Character
VA TE       0             ? (NB)
VA $TEn     0             Type Effect: status of different
                                bits set with the EF
                                printer file setting
                                0|1=On
VA TF       0             Ignore Top Margin
VA $TF      0             Ignore Top Margin
VA TL       C:\nb\support\debug\TEXT.LIB    Text Library
                                (NB)
VA $TL      1             ? (NB)
VA $TM      6:07 PM       Current TiMe
VA TO       0             (Time function of some sort)
VA TP       .3,.3         Top Margin
VA $TP      0             Current Cursor Position [NB]
VA TR       864,1800,2520,3240    Tab Settings
                                (1.2,2.5,3.5,4.5)
                                measured in points
                                (72/")
VA $TR      LB            TRiangle Mnemonic: display command
                                embedded in delta
VA TS       1.2,2.5,3.5,4.5       Tab Settings
VA TW       0             ? (Xy4); Text Width [NotaBene
                                (=<< VARM>> )]
VA $TW      0             Text Window command window
                                0=text|1=window|2=heade
                                r, footer, frame, foot-
                                note
VA TX       0             Triangle Suppress:
                                0=show|1=display con-
                                tiguous deltas as one
VA $TX      1             Cursor Location: 0=Header|1=Text
VA $U1      (none)        U1 File
VA $U2      C:\NB\XYWWWEB.U2      U2 File
VA $U3      (none)        U3 File
VA $U4      (none)        U4 File
VA $U5      (none)        U5 File (Windows)
VA $U6      (none)        U6 File (Windows)
VA $U7      (none)        U7 File (Windows)
VA $U8      (none)        U8 File (Windows)
VA $U9      (none)        U9 File (Windows)
VA UA       1             User Access: treatment of DeFined
                                text  0 = Xy4 -
                                DOS|1=Windows (Windows)
VA $UA      1             User Access
VA UB       65535         ?
VA ]UD1                   Use Main Dictionary (value of cur-
                                rent UD default)
VA ]UD2                   Use Supplemental Dictionary (value
                                of UD default)
```

```
VA UD                   Use Dictionary (current applicable
                                UD command)
VA $UDn                 Use Signature PCLEX Dictionary:
VA $UD1     0           Medical
VA $UD2     0           Legal
VA $UD4     0           Computer terms
VA $UD8     0           Special
VA $UD16    0           Any Supplemental
VA UF       COURIER NEW Use TypeFace
VA UH       IN          Units Horizontal (default measure,
                                e.g. IN)
VA $UH      IN          Units Horizontal
VA UI       1,1,1,1,1,1,0,1,1,1,0,0         User Interface
                                (Windows):
VA UI1      1           User Interface Command line is
                                visible
VA UI2      1           User Interface 1=Status line is
                                visible
VA UI3      1           User Interface 1=Button bar show-
                                ing
VA UI4      1           User Interface 1=Format bar show-
                                ing
VA UI5      1           User Interface 1=Ruler bar showing
VA UI6      1           User Interface 1=Menu bar showing
VA UI7      0           User Interface 1=Horizontal scroll
                                bar showing
VA UI8      1           User Interface 1=Vertical scroll
                                bar showing
VA UI9      1           User Interface 1=CMline Position
VA UI10     1           User Interface 1=PRompt line Posi-
                                tion
VA UI11     0           User Interface 1=Button position
VA $UI      (none)      XWUIF.UIF file location (Windows)
VA UL       0           UnderLine Setting
VA UM       0           Unhide MoDe Markers
VA $UM      0           Unhide MoDe Markers
VA UN       0           UNtitled File
VA $UN      0           UNtitled File
Unformatted Copy Clipboard                  N B . I N I
                                [Defaults]: 0|1=Text
                                only
Unformatted Paste Clipboard                 N B . I N I
                                [Defaults]: 0|1=Text
                                only
VA $UO                  Use Outline Fonts 0|1=supported
                                (Xy4)
VA UP       0           ?
VA UR       0           Use Rodent 0|1=Yes
VA $UR      0           Use Rodent
VA $US      0           U S - E n g l i s h  s w i t c h :
                                0=native|1=English
VA UV       IN          Units Vertical (default measure,
                                e.g. LI)
VA $UV      IN          Units Vertical
```

```
VA V3       0             ?
VA $V3      0             ?
VA $VA      0             ? (NB)
VA $VE      V4.1          Version Number
VA VF       0             Variable Forms (0|1=use multiple
                                  lines)
VA $VF      0             Variable Forms
VA $VH      0             ? (NB)
VA $VI      0             T y p e  o f  G C  v a r i a b l e :
                                  0=words|1=numbers|2=dat
                                  es|3 =calculations
VA $VL      0             ? (NB)
VA $VM      0             Vendor-Independent Messaging:
                                  0|1=V I M-compatible
                                  electronic mail system
                                  installed
VA $VN      0             ? (NB)
VA VO       1             ? (NB)
VA $VO      1             ? (NB)
VA VU       5,100,6       Vertical Units: min vertical move-
                                  ments in 1/6",screen
                                  lines,decimal places in
                                  vertical formats
VA $VU      5,100,6       Vertical Units
VA WA       36            Error Message Wait Time
VA $WA      2             Window Availability
VA WB       ??ʒ?pù????????□?       Window Border Characters
VA $WB      ??ʒ?pù????????□?       Window Border Characters
VA $WC      0             Word Count from SPELL|WC|WCB com-
                                  mand; number of CHanges
                                  made by CH|CI command
VA WD       3             Number of lines for WiDows
VA $WE      0             Where-Is-Error: displays location
                                  o f  e r r o r  " L e f t
                                  m a r g i n / i n d e n t  i s
                                  g r e a t e r  t h a n  r i g h t
                                  margin"
VA WF       1             Wrap-to-Fit (0|1=keep within cur-
                                  rent borders)
VA $WF      3             Status of UWF command (printing
                                  m o d e  a n d  f o n t s  i n
                                  effect) 0|1|2 (Windows)
VA $WH      4294965855    WHere: location of insertion point
                                  (after BE/wh command)
VA ?WI      1 C:\NB\INBOX\CPG\CPG.NB□       W I n d o w  l i s t
                                  ( < V A $ W N >  s  a n d
                                  d:\path\filenames) (NB)
VA $WI                    WIndow Parameters: dimensions
                                  (left,top,width,height)
                                  of the text window
VA WL       0             ?
VA $WL      0             ?
```

```
VA $WM                  ?
VA WN       1           Window Handling Style: auto-
                                renumber=0; fixed num-
                                bers=1 (NB)
VA $WN      63          Window Number
VA WO       0           Word Overstrike all except:
                                <cr>=0|...+space+tab=1|
                                ...+separators=2
VA $WO      1           Windows Open
VA ?WP      EPSON Stylus Photo 2100 on Ne01: NB: Current
                                Windows default
                                Printername
VA $WP                  Returns default printer driver, or
                                (with args) corresponds
                                to WPROF command
                                (Windows):
                                count,file,appname,keyw
                                ord (no args returns
                                default printer device)
?WS         windows sets
VA WS       0           Whole-Space Justification 0=par-
                                tial (micro) spaces|1
VA $WS#     1           Window Status (0=no
                                file|1=file|2=dir):
VA $WS1     1           Window 1 Status
VA $WS2     0           Window 2 Status
VA $WS3     0           Window 3 Status
VA $WS4     0           Window 4 Status
VA $WS5     0           Window 5 Status
VA $WS6     0           Window 6 Status
VA $WS7     0           Window 7 Status
VA $WS8     0           Window 8 Status
VA $WS9     0           Window 9 Status
VA $WT      0           ? (NB)
VA $WV#                 Windows Flag Values (Windows):
VA $WV1     19497       Protected mode
VA $WV2     19497       CPU 286
VA $WV4     19497       CPU 386
VA $WV8     19497       CPU 486
VA $WV16    19497       Standard
VA $WV32    19497       WIN286
VA $WV64    19497       Enhanced
VA $WV128   19497       WIN386
VA $WV256   19497       CPU086
VA $WV512   19497       CPU186
VA $WV1024  19497       Large frame
VA $WV2048  19497       Small frame
VA $WV4096  19497       80x87
VA WW                   Conversion Filters Path (Xy4)
VA $WW                  Conversion Filters Path
VA WX       0,0,0,0,0   Windows EXception Characters:
                                associate Speedos with
                                TT fonts
```

```
VA $WX#                 Windows Font Family: display Bit-
                                stream font used for
                                character substitution
                                (Windows):
VA $WX1                 name of serif Bitstream font
VA $WX2                 name of sans serif Bitstream font
VA $WX3                 name of monospaced Bitstream font
VA $WX4                 name of decorative Bitstream font
VA XA       0           ?
VA $XA      0           ?
VA XC       12          Space Constant
VA $XC      12          Space Constant
VA XD       0           Read-Only Directories: 0|1=R/O
VA $XD      0           Read-Only Directories
VA XE       174         ? (NB)
VA $XE      174         ? (NB)
VA XF                   EXtract Fields from data file with
                                SORTD
VA XI       0           Swap Italics (Windows)
VA $XI      0           ? (NB)
VA XL                   Selective Directory Listing
VA XM       *PL*TI      Transpose Messages: Order on
                                PRompt line
VA $XM      *PL*TI      Transpose Messages
VA XN       300         ?
VA $XN      300         ?
VA $XP      C:\NB\NBMAIN-E.AUX     ? (NB)
VA XR                   EXtract Records from data file
                                with SORTD
VA XT       0           EXpand Triangles when cursor on
                                delta=1
VA $XT      0           EXpand Triangles
VA $XW      6.25,6.25   TeXt Width
VA XX       0           ? (NB)
VA $XX      0           ? (NB)
VA XY                   SCRFONTS.BIN Location (default)
VA $XY                  SCRFONTS.BIN Location (Xy4); Full
                                Screen Window dimen-
                                s i o n s
                                (left,top,width,height)
                                of application window
                                (Windows)
VA $XZ      0           Status of Application Window:
                                0=window has been
                                restored to previous
                                size|1=minimized|2=maxi
                                mized
VA Y3       2           XyWrite 3+ compatibility mode=1
VA $Y3      2           XyWrite 3+ compatibility mode
VA YK       0           PRompt offset (Xy3+ only)
VA $YK      0           PRompt offset (Xy3+)
VA ZB       4           ?
VA $ZB      4           ?
```

VA ZC	1	Zero Capitalization 0|1=preserve case in spelling
VA $ZC	1	Zero Capitalization
VA ZD	0	? (NB)
VA $ZD	0	? (NB)
VA ZL	0	LaZer printer 0|1|2|3 (set in PRN file)
VA $ZL	0	LaZer printer
VA ZM	100	ZooM percentage
VA $ZM	100	ZooM percentage
VA ZO	0	?
VA $ZO	0	?
VA ZS		6,7,8,9,10,11,12,13,14,16 Point Sizes for Scalable Fonts
VA $ZS		6,7,8,9,10,11,12,13,14,16 Point Sizes for Scalable Fonts
VA ZX	1	Cancel eXpanded memory=1|0=use Xmem
VA $ZX	(none)	Cancel eXpanded memory (Xy4); ? (NB)
VA $ZZ	&□B\AUTOSAVE\AUTOSAV1.TMP	Miscellaneous functions, e.g. last questionable word found by speller or field name when attempt is made to delete a GC field

Dialog ListBox Lists:

?AG	All Save/Gets in list
?AR	ARea commands
?AS	All Symbol Sets
?BB	Button Barré (not used at 3/23/95)
?BM	Button Macros
?BO	BOrders
?BU	BUttons used
?CA	CArtridge list
?CD	Cartridge Directory
?CM	List of CoMmands
?DB	List of Delete Blocks
?DC	DDE Conversations
?DR	List of DRives
?DZ	Default siZe
?FB	All items in Format Bar
?FN	List of FuNctions
?FO	FOnts
?FR	Another FRame
?FU	Format bar being Used
?GR	GRoups in Global Library
?IB	List of commands
?IG	Includes

```
?IX         IndeX from a U1...U9 file
?JR         File JouRnal
?LS         LiSt of items from GC.INI
?MA         MAcro Save/Gets A-Z, 0-9
?MT         MaTch stuff for fonts
?OP         OPerators for calc
?PC         PCcodes
?PG         ProGram being run
?PP         Printer Files (SETP)
?PR         PRinter queue
?RG         Show ReGistration from OLE
?SB         SymBol set
?SD
?SG         Save/Gets
?SK         SK function (show a Save/Get)
?SP         SPelling errors
?SS         Styles
?ST         Command STack
?SY         SYnonyms
?SZ         SiZe
?VR         VaRiables for GC
?W2
?WB
?WI         WIndow list
?WP         Windows Printer driver
?WS         Windows Sets
```

Miscellany of XPL Information, chiefly by Carl Distefano

These are notes that I've excerpted over several years from an email correspondence with Carl Distefano about XPL programming. The information is his; the phrasing is sometimes mine, but more often his. A couple of entries are by others (noted). I have tested some of these suggestions, but not all.

Turn Auto-replace off

The function string AC,AZ,AZ in a keyboard table or program turns off auto-replacement. AZ turns it on again.

```
##=ac,az,az
##=az
```

or, in a program:

AC AZ AZ to turn it off; **AZ** to turn it on.

SG—Run all phrases from one key

Defining one key as

##=SG

enables you to run all your phrase-library keys from that one (##) key, freeing all the alphanumeric keys on your Shift+Alt keyboard for redefinition. You strike the ##=SG key, then the letter for the phrase you want, and the phrase is entered into your file (or run, if it's a program).

It also has the advantage that you do not remove whatever is currently on the command line.

Func XH at head of files

There's hardly a program you can write for NB5 that won't benefit from a prefatory **XH BX** es 1**Q2** . Func XH removes any displayed menu or dialog box.

And always use **BX** ES 1**Q2** . You don't need to use ES 0 too - NB cancels the error-suppression state when the program ends.

Value of the Wait variable (DF WA= in NB.DFL)

If you set it to 0 - programs with error messages execute faster. 'The only reason to set it higher (and then only temporarily) is to debug programs that generate error messages that flit by too quickly to read. (Or if, like me, you want time to read ordinary error messages.) Each unit of 18 (set in Tools, Preferences, Prompts, Time that other messages are displayed) = 1 second.

DX

Don't use DX as a matter of course [as one did in NB4], because it can destabilize some code (though this is pretty rare). If you need to use DX, test your program thoroughly before and after introducing this function to make sure that DX isn't causing errors.

CH and CI

Are identical in NB5, and mean ‘change invisible’.

Commenting string

Use the string:
;*;
to make comments within programs, not LaBels; and not comments after an «EX». A 500-word comment after an «EX» can slow a program up by 15%.

A comment is defined as anything from the ;*; symbol to the next carriage return. The ;*; symbol can also be used to create discrete "lines" of XPL (as <<LB CR>>s could in NB4).

Opening an unnamed file (e.g., as a temp file for programming purposes)

You don't need to name a file to open a temporary window. **BX** ne**Q2** (no filename) opens an Untitled file.
BX ne/100**Q2** opens it in eXPanded view.
BX ne/#**Q2** , where "#" is any recognized DT value, opens the window in that display type. (On this formula, ne/0 should open the window in eXPanded view, but for some odd reason it doesn't work. You need ne/100.)

Search Switches

/F - puts the cursor at the beginning, rather than the end, of the found string
/T - search from top of file regardless of cursor position
/S - confine the SEarch or CHange operation to selected text, i.e., a DeFined block
/n - (where n stands for any number) find the nth instance of the search string. Thus, SE/3 "man" finds the third instance of "man" (counting from the cursor position).
With Change commands (CH, CI, and CV), switch /n has a different meaning: it limits the number of changes to the specified number. In other words, CH/3 |man|men| will only change, at most, the next three instances.

Good idea to get used to using double quotes, " " as search delimiters, so that strings that include switches can be used.

Carriage Return wildcard

[View this section in Show Codes View]
□ or ^R. Use former for compatibility with Xywrite.
Input it by executing func WC

Negation wildcard
[View this section in Show Codes View]
- or ^B.

Example 1: se "c-x" will find "c" followed by any character but "x".

Example 2: se/f "-□ -□" will find any CR that is neither preceded nor followed by a CR, thus making it possible to search for and eliminate single CRs at line ends in email files, while retaining the CR CR sequences that end genuine paras.

You cannot use the 'CI' or 'CH' commands with the negation wildcard **-**. You have to use a 'SE' loop.

The keyboard file assignment [for the negation wildcard] is > "NN,-". <<

Guillemet [chevron/command bracket] wildcards - ® and ¯
To input them from the command line, do: func << and func >>. Or assign these functions to keys: nn=<< and nn=>>.

RK and branching
RK doesn't uppercase the input [as it did in NB4]; to do that, you need @upr(). The difference between RK and RC is significant. RK captures the first character or function call assigned to a key in the keyboard file and discards the rest of the key assignment; if you direct <RK> to a Save/Get (), the effect is to discard the *entire* .KBD file assignment associated with that key. You can then do whatever you want with the keystroke. Typically, you use the key code (41) of the pressed key to branch to different parts of the program, or to disable the keystroke altogether. Below, for instance, the program responds to Escape or Enter but nothing else:
[View this section in Show Codes View]
;*;
;*;
;*;
;*;

In contrast, RC captures the first character or function assigned to a key, but allows any subsequent characters or functions in the key assignment to execute -- unless you loop around and continue to capture (concatenate) them, something RK doesn't allow. RC is appropriate when you want to capture and *give effect to* the user's assigned key assignments. When you want to suppress them and substitute your own actions, use RK.

Operating on defined blocks in programs
[View this section in Show Codes View. But first, try highlighting some nearby text and see what happens to the 0s 2 lines down.]
DZ - closes a DeFined block if one is started, otherwise does nothing
0 - cursor position at start of DF block
0 - cursor position at end of DF block

This segment:
/s
inserted in an change/search clause, will confine the operation to a defined block, if there is one. E.g.,

BX ci/s "but"butter"**Q2**

will change all buts to butter in a define if there is one, or the whole file if not.

SA%
The basic command-line usage is SA %X, where X is a phrase that has content.
Without more, the contents of the phrase are saved to a file named X.SAV. (Any existing file named X.SAV is summarily overwritten.) However, you can specify a different filename, e.g., SA %A,MyPhraseA.txt (also overwritten if it exists).

Typically, in XPL, you'd test for existence of the file first (because, if it already exists, you may not wish to overwrite it!):

```
;*; Save the null string to phrase 01
«SV01,»
;*; Test for existence of file
BX exist d:\path\myfile.txtQ2
;*; If file does not exist
«IF«ER»
;*; Create the empty file and wait for it to be written to disk
BX sa %01,d:\path\myfile.txtQ2 BX waitQ2 «EI»
```

Appending to a phrase in programs
It's unnecessary to save text to a phrase in order to add it to an existing phrase. Simply "quote" the additional text. Thus:

```
<SX01,<IS01>+" gathers no moss.">;*; Full proverb
```

Echo phrase to prompt line
You can echo the contents of a phrase to the PRompt line:

```
<PR@01>;*; Proverb is displayed on PRompt line
```

Note: Default MB must be set to 0. If default MB=1, the contents will display in a Windows message box which will persist on the screen until you press Enter or click on "OK". This causes problems with programs that loop repeatedly through PRompt statements.

Prompt can mix text and phrase number

A PRompt may consist of text and a phrase number, as long as the phrase number comes last:

<PRHere is the proverb: @01>;*; Proverb with prefatory remark

---

Manipulate variables and values directly

System VAriables and VAlues can be manipulated directly. It's no longer necessary to save the VA to a phrase:

<IF<VA$WS>==0><PRNo file is open><EX><EI>;*; Test for open file

---

New extensions to VA operator

There are several extremely important extensions to the VA operator:

Don't use @siz(<IS01>) to get the length of a phrase. Use <VA|01>.

@siz(<IS01>) crashes if phrase 01 is uninitialized; <VA|01> doesn't; rather, it returns a value of -1 (useful information).

<VA" 01> tells you whether 01 contains a number (integer) or a string. 1=number; 0=string

<VA!01> provides important information about the status of phrase 01. If:

VA!01=0, phrase 01 contains a string
VA!01=2, phrase 01 contains SUbroutine
VA!01=4, phrase 01 contains a numeric expression
VA!01=16, phrase 01 evaluates to FALSE
VA!01=24, phrase 01 evaluates to TRUE
VA!01=255, phrase 01 is not initialized

<VA@01> returns the first 77 characters of phrase 01, the whole phrase if the length is 77 or less. An extension to VA@ allows easy extraction of segments delimited by a separator. Suppose this:

<SV01,Spring;Summer;Fall;Winter>;*; Data

Then <VA@01;1>=="Spring", <VA@01;2>=="Summer", etc.

ð Containment operator (replaces epsilon)

There's a new ð (Ascii-240) containment operator, which tersely reports whether one string "contains" another. (The operator is the Ascii-240 character itself, not the number in curly braces.) For example:

```
<IF"Spring"{240}"pr"><PRYes><EX><EI><PRNo><EX>;*; Reports "Yes"
;*;
<IF"Spring"{240}"Pr"><PRYes><EX><EI><PRNo><EX>;*; Reports "No"
```

NB: The Contains wildcard, ascii 240: unlike ASCII 238, there is no numeric value associated with 240. The statement <IS01>[240]<IS02> is either true or false; if true, it says nothing about the *position* of 02 in 01. If you need to know the position, use epsilon.

Count Up operator

There's a nifty Count Up operator that makes it easy to execute a segment of code a specified number of times. Using it, you can time your programs like this:

```
 <SX01,10000>ZT <CUa,01>NO <LBa>;*; Count Up
 ;*;
 <SX01,"Done - Elapsed time: "+<VA$ET>><PR@01><EX>;*; Report elapsed   time
```

In the first line, everything between the <CU> statement and LaBel "a" (<LBa>) -- in this case, a simple func NO (No Operation) -- is executed the number of times stated in phrase 01. CU requires two elements, a delimiting LaBel name and a phrase which must contain the desired number of repetitions. Note how compact the CU formulation is. Also, if you translate the examples into live code and run them, you'll see that CU is significantly faster (on my machine, by a factor of 135%).

GT

GT can be used to put text on the command line as well as in text. Thus,

```
<SV01,Hello, world!>BC <GT01><EX>;*; Put to command line
<SV01,Hello, world!>GT <GT01><EX>;*; Put in text
```

In Xy3 (possibly also NB3 and 4), PV was required to put text on the CMline.

Search for function codes

You can search for a particular function code by striking your Pfunc key (Ctrl ;) *twice*, then typing the 2 letters of the code you're looking for. For instance, to search for BC, strike Pfunc twice, then type bc. If there's a BC code in your file, the cursor will leap to it.

Save text to an sx phrase by enclosing it in double quotes.

«SX02,«VADR»+"COPY.TMP"» - you can save text to an sx phrase by enclosing it in double quotes. [VADR is the dir that NB uses to create temp files.]

Double quotes can be used to refer to any string (i.e., anything that can be saved to a phrase wih SV, e.g., <SV01,Hello, world!>) for purposes of concatenation or testing for containment with either epsilon or Ascii-240. It avoids the need to introduce new phrase numbers in such operations, with the result that code is easier to write and read. Of course, there are

still operations, notably parsing -- with either XS or VA@nn[separator][#] (there are examples of both in WILDFUNC) -- in which saving the subject string to a phrase remains mandatory.

NB (In VA@nn[separator][#], nn denotes a phrase containing a string, [separator] is the character used to delimit segments of the string, and [#] is the number of the segment to be parsed out.)

How to put your own programs into the XYWWWEB.U2 file:
Assign your program a Type-5 framename, enclosed in double curly braces. [Lots of examples in U2 itself.] Put the program code immediately beneath, enclosed between Ascii-2's. (Use any existing frame as a model.) If you want to be able to pass arguments to your program (i.e., with PROGRAM_NAME args<Helpkey>), use Phrase 50 as the argument holder. Again, see existing frames for examples. Add your programs at the *bottom* of the U2 file, then issue Command LH<Helpkey> to reload the file.

Drag files into NB from Explorer or PowerDesk
Drag the files not to the workspace for the file but to the rulers or toolbars on the edge of the NB window—any grey surround will do; or even the title bar. [Mary Bernard/others]

Keys available for User Keyboard Definitions (this dates from NB5.5)
[from Steve Siebert]
Esc and PrtSc/SysRq are not available in any keyboard state --Tab is not available in C, C+S, A, and A+S states --F4 is not available in C and C+S states --F6 is not available in C and C+S states as well as A and A+S states --Del is not available in C+A and C+A+S states --The following keys (harmlessly) pull down an NB menu, but can otherwise take assignments:
A+W
A+E
A+T
A+I
A+P
A+F
A+H
A+V
A+M
A+Space, A+S+Space
--Pause/Break is accessible as key #90 in C, C+S, C+A, and C+A+S states, and as key #69 in A and A+S states
But note that there are now no NB restrictions, other than the list of Alt keys noted above (and even that follows a Windows convention). That is to say, the keys that cannot be assigned by users are keys that are reserved by Visual Basic for Windows.

SUMMARY OF KEYS NOT AVAILABLE (same info as above in different format)

C+Esc
C+Tab
C+F4
C+F6
C+#69
C+PrtSc

C+S+Esc
C+S+Tab
C+S+F4
C+S+F6
C+S+#69
C+S+PrtSc

A+Esc
A+Tab
A+W/E/T/I/P/F/H/V/M/Space
A+F6
A+PrtSc
A+#90

A+S+Esc
A+S+Tab
A+S+Space
A+S+F6
A+S+PrtSc
A+S+#90

C+A+Esc
C+A+#69
C+A+Del
C+A+PrtSc

C+S+A+Esc
C+S+A+#69
C+S+A+Del
C+S+A+PrtSc

Append and APT (APpend to Top of file) commands

The command 'append' appends one file to the bottom of another. You can use it in the form:

F9 append [fileA],[file B] F10

To append the file in the current window to another file, you can cut out the '[fileA]' part and simply say:

F9 append [fileB] F10

If you use ‘apt’ in ts form, you’ll get an error message, 'diskette full'—and the current file is not appended to the top of the target file.

‘apt’ requires full syntax - you can't leave out '[fileA]'. What works is:

F9 apt [fileA],[fileB] F10

Function IV

opens an input window like a footote but leaves no trace in normal view. In draft view it is identified as "Invisible comment" in verbose mode, IV in brief mode.

Since NB does not print background, you can use white type on a dark background and it will be visible on screen but not print. In ver. 8 this is easy to do with control-7 and control-8. [from Joel Lidov]

BX and repeat commands

Note that you can't do bx command q2 q2 q2.... as you can with bc command xc xc xc. So, if you're substituting bx q2 for bc xc, don't do it in contexts where you have a succession of xcs executing the same command (an example might be if you are changing all double spaces to single spaces, and want to make sure you include any accidental triple or quadruple spaces. bc ci / / /xc xc xc xc would do it; bx ci / / /q2 q2 q2 q2 stops after the first q2, so doesn't catch triples.

BX notes, from Carl Distefano's BX tutorial:

If you find that BX misbehaves, I recommend downloading the full tutorial, at:
http://www.serve.com/ammaze/xy/BX.ZIP

Paired with function Q2 -- as in **BX** command**Q2** -- BX executes native commands *without* blanking and rewriting the command line.

BX is the standard way to execute commands in XPL programs running under Nota Bene for Windows, XyWrite 4 and XyWrite for Windows. It replaces **BC** command **XC** in most contexts. (The BC method still works, and is useful in special situations.)

When NB encounters a BX is encountered it the command that follows. It then waits -- indefinitely if necessary -- for a closing Q2. Thus, every BX **must** conclude with Q2 before other instructions can be executed. This is a major difference from funcs BC and XC, which are independent of each other

Wildcards: SEarch wildcards must appear as wildcards (reverse-video characters that look like a single character), not as functions:

The command

'BX se "**S**and**S**"Q2

will find the word 'and'.[**S** is the any-separator wildcard, input by doing F9 func nn F10, then pressing 's']

'BC se "**WS**and**WS**"XC [Here, **WS** is the any-separator wildcard, input with 'pfunc ws']

will also find it (**WS** gets translated to **S** on the command line). But:

'BX se "**WS**and**WS**"Q2

will not, because BX interprets the WS literally and searches for a string containing the function WS.

Command Brackets

Strings containing balanced pairs of 'guillemets' [XyWrite's term for command brackets] can be searched for literally:

BX se "«US0»"**Q2**

This searches for the embedded command «US0»

Thet search can also be done using the 'guillemet wildcards' ® and ¯:

BX se "®US0¯"

Guillemet wildcards are **required** when:

(a) the SEarch string includes unbalanced guillemets,

(b) using real guillemets would have the effect of embedding an undesired command,

(c) the SEarch string is an XPL expression.

BX se "®IP"**Q2**

SEarches for embedded IP command (unbalanced guillemet)

BX se "®IF**O**®EI¯"**Q2**

SEarches for IF *or* EndIf in XPL code

When in doubt about whether to use real guillemets or guillemet wildcards, use the wildcards.

Functions AK and SH in NB

On the other hand, the absence of AK or SH certainly doesn't mean you can't take advantage of the menus in your XPL programs. You can still execute any menu by making a direct call to the corresponding frame in NB.DLG.

The usage is JM framenameQ2 (where JM and Q2 are 3-byte funcs, of course). For example, to activate the Import Text Files menu, you'd write JM TextBQ2 . (Or, in the KBD file, nn=JM,T,e,x,t,B,Q2.) It's actually far superior to AK. Why traverse three or four menus to get to the one you want when you can go there directly? In this connection, you may want to take a look at the discussion in BX.TXT on "Companion Functions JM and JH" (BX.TXT can be downloaded at:

XyWWWeb, http://www.serve.com/ammaze/xy/BX.ZIP). Finding the frame to execute is usually just a matter of searching in NB.DLG for text that appears in the menu display. For example, to find TextB, I searched for "Import Text File". The framename is in double curly braces at the start of the frame -- here, {{K,TextB}}. It also helps to know that that all dialog boxes are Type K frames; i.e., the first char after the open curly braces is "K".

You'll find it opens up possibilities. It can also be used to make calls to Jumbo U2 routines (in which case the syntax would be JM 2.framenameQ2 ; the "2." points to the U2 file -- again, see BX.TXT). Frames described in the section of XYWWWEB.INF entitled "XyWWWeb.U2 Common Resources: Reusable Subroutines for Discrete Tasks" are especially well-suited for this purpose. In fact, the Jumbo U2 is built on this kind of interoperability -- routines calling other routines all over the place. So are the menus in NB.DLG. It's a new dimension to XPL that wasn't possible before Xy4|NB5.

Runcode

Another very useful testing/debugging tool is U2 frame RUNCODE. Have you tried it? It does several things, but the handiest usage is to run DeFined blocks of XPL code. Simply DeFine the code you want to run and execute RUNCODE<Helpkey>. You can even feed an argument to the code that accepts arguments, with RUNCODE [arg]<Helpkey>.

Time programs with function ZT

You can now time your programs. Func ZT ("Zero Time") resets the timer to 0. The Elapsed Time is reported by <VA$ET>. Here's how you would time how long it takes to count to 10,000:

<SX01,0>ZT ;*; Initialize 01 and reset timer ;*;
<LBa><IF<PV01><10000><SX01,<PV01>+1><GLa><EI>;*; Count up ;*;
<SX01,"Done - Elapsed time: "+<VA$ET>><PR@01><EX>;*; Report elapsed time

---

Func + wildcard on cmd line or in text
Issue func NN on the CMline, and then hit the minus key. That will produce the negation wildcard. And hitting any of the other keys I mentioned will produce the other wildcards. The wildcard will be inserted at the cursor position, either on the command line or in text. If you want it in text, you have to CC the cursor to text before executing func NN.

Func NN
Func NN takes an argument: the minus sign produces the negation wildcard;
the numbers 0 through 9 produce the numeric (repetition) wildcards;
an Ascii-46 full stop produces the sentence separator wildcard;
Ascii-17 produces the Ascii-13 (carriage return) wildcard;
the Ascii-25 down arrow produces the Ascii-10 (linefeed) wildcard;
Ascii-27 produces the CrLf (carriage return+linefeed) wildcard (also produced by executing func WC);
O produces the logical OR wildcard.
Also, NN followed by A, L, N, S, W or X will produce, respectively, the alphanumeric, letter, number, separator, variable-string and variable-character wildcards, which can also be produced by executing funcs WA, WL, WN, WS, WW and WX.

Close a prompt window—to close within a program (=F3) - 3 ways
1. The foolproof way to do it -- if you know positively that a command window is already open -- is (or should be) YD XD. Func XD closes a command window OR undefines text if any is defined.
Therefore the initial func YD is necessary to undefine any defined text, so that the ensuing XD is sure to close the window. (XD XD is a no go, because if text isn't defined in the command window, the first XD will close it and the second XD will undefine any defined text in the *main* window. No good.)
2. If you're not certain whether a command window is open, you can test for it with <VA$TW>, which returns 1 if such a window is open, else 0. Thus the absolute foolproof way is <IF<VA$TW>(>)0>YD XD <EI>, where < and > represent guillemets and (>) represents the greater-than sign.
3. Actually, since commands that open command windows can be nested, open command windows can themselves be nested (one open within another open within another, and so on). So, really, the absolutely, positively foolproof way is to have a loop that repeats until the VAlue of $TW tests 0, like this: <LBa><IF<VA$TW>(>)0>YD XD <GLa><EI>.

Functions list, from U2 file

@0 @1 @2 @3 @4 @5 @6 @7 @8 @9 @A @B @C @D @E @F @G @H @I @J @K @L @M @N @O @P @Q @R @S @T @U @V @W @X @Y @Z AD AS BF BK BS CC CD CH CI CL CM CN CP CR CS CU DC DF GH DL DP DS DW EL ER EX GT HM M0 M1 M2 M3 M4 M5 M6 M7 M8 MD MU MV NC NL NK NP NR NS NT NW PC PD PL PP PR PS PT PU PW R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 RC RD RE RL RP RS RV RW SD SH SI SK SM SN SS SU SV TF TI TN TS UD WA WC WL WN WS WX WW XC XD DT S1 S2 S3 S4 S5 S6 S7 SP BC LB LE NF PF TP BD MS NM LD LL LR LU UP FF YD DO DX MK SO OP WZ NX SW FD FM TL TR TE ED EE HC EC MC #1 #2 #3 #4 #5 #6 #7 #8 #9 $1 $2 $3 $4 $5 $6 $7 $8 $9 DR EN C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 EF IB NO NI CO $0 LS XP WG XM &0 &1 &2 &3 &4 &5 &6 &7 &8 &9 &A &B &C &D &E &F &G &H &I &J &K &L &M &N &O &P &Q &R &S &T &U &V &W &X &Y &Z HL $A $B $C $D $E $F $G $H $I $J $K $L $M $N $O $P $Q $R $S $T $U $V $W $X $Y $Z XX H@ VH MW QH DK SR SC TG H1 JH DZ DD DM LT RK NN MT ET ZT T1 TT << >> IT SL SF FL FR FC SY ME AC FS TW MI RO NB Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 TO IR AR AX DB DE HF SA OV TC TB JM SG XH FT BX MN CB M9 MZ ZZ RX ST KF JC AK TM NU B4 QP HG US XE ES RB S- S+ ** BN RU CF UI XS EA BT KD DN HI WH XN FX UN MX AZ

Making print mode changes work on words with apostrophes

Use built-in functions, thus:

```
#=YD,DW,BX,(,m,d, ,b,i,),YD
```

To exclude the trailing space (or other trailing separator) from the MoDe change, try this:

```
#=YD,DW,CL,DM,DZ,BX,(,m,d, ,b,i,),YD
```

Either of these should solve the problem of the excluded apostrophe.

Straight double quotes in programs

Double quotes can be used to refer to any string (i.e., anything that can be saved to a phrase with SV, e.g., <SV01,Hello, world!>) for purposes of concatenation or testing for containment with either epsilon or Ascii-240. It avoids the need to introduce new phrase numbers in such operations, with the result that code is easier to write and read. Of course, there are still operations, notably parsing -- with either XS or VA@[separator]nn (there are examples of both in WILDFUNC) -- in which saving the subject string to a phrase remains mandatory.

XYWWWEB.U2: if calling it in Page Layout View crashes NB

Call it in Show Codes view with F9 ca/100 xywwweb.u2 F10

Access NB menus from the keyboard

In Chapter 7 I described how to use a shareware macro program, such as Macro Express, to run 35 programs from one key. You can also use such a program to do something you cannot do via XPL: access items on Nota Bene's menus without a mouse. For instance, the dialog accessed by Tools, Page Indexes, Mark...does not have a keyboard shortcut. But Macro Express lets you do it, with a macro that reads '<ALT>TXM' (plus a couple of codes to slow the macro down a tad; otherwise it's too fast for NB). I use other macros to open Tools, Preferences and to do File, Maintain in Ibidem. [Mary Bernard]

Codes that Do Not Seem to Work in NB for Windows

These codes may work for others, but they do not for me. There are two lists; those in the second are probably only used in XyWrite. I include both lists for the sake of completeness.

Codes which are more likely to work:

AP — Auto-pause on (pause printing at end of each page) [embedded command]
(needs testing by someone whose printer supports it; mine doesn't.)

beep — Used in programming to produce a beep [immediate command]
(doesn't work for me) (XyWrite)

B4 — Display dialog box previous to the last one displayed. [function]

box — (XyWrite only - freezes NB) [immediate command]

BT — Toggle window borders on and off. (doesn't work for me) [function]

cart — Loads font information for the named cartridges. [immediate command]
(XyWrite only 'command not recognised' in NB Win)

cdl — Change directory (any difference between this and 'cd'?) [immediate command]

CF — Column Func (how does this work?) (1=Ins before; [function]
2=Ins after; 3=Del; 4=Select column; 5=Select table)

clrsum or cs — Clear sum in memory without inserting value into text [immediate command]
(Doesn't work for me on sum done with 'x+y=z')

clrxsgt — Clear all 3-digit programming phrases and current [immediate command]
'run' command (not tested, sounds drastic)

cm — (found this in a NB4 program that still works: 'BC cm dXC' [immediate command]
What does it do?)

correct — (XyWrite only) [immediate command]

cs — Same as 'clrsum' [immediate command]
(Doesn't work for me on sum done with 'x+y=z')

DK — (Does nothing) [function]

docbld — (XyWrite only) [immediate command]

DR — (Does nothing) [function]

EA — Open command window for editing text only. [function]
('place cursor on marker' - but doing so has no effect)

ed or edit — Call file (XyWrite) [immediate command] obsolete

EF # — Special printing effect (XyWrite only) (not tested) [embedded command]

Activates special printing effects for printers that support them. # is one the following values, or a combination of them:

1 Reverse	32 Not Assigned	1024 Double Underline
2 Outline	64 Not Assigned	2048 Overscore
4 Shadow	128 Double High	4096 Floating Underline
8 Inverse	256 Script Up	8192 Outline/Shadow
16 User Set	512 Script Down	16384 Wide

To activate more than one special effect at the same time, add values assigned to each effect. E.g., "ef 4224" activates double high c^Rharacters with floating underline (where 4224 is sum of two special-effect values (128+4024). This command also turns off any other special effects that were active.

(needs testing by someone whose printer supports it; mine doesn't.)

ET Compute amount of time elapsed since ZT function and insert in text. [function]
(programming) (not tested) (XyWrite)

EV Evaluate (XyWrite Mailmerge only) [embedded command]

FS Return cursor to last misspelled word and display [function]
spelling menu (XyWrite only)

HG 0/1 Display border around graphic area without displaying graphic. [function]
0 turns display of graphics off, 1 turns it on (doesn't work for me)

I1-I9 Set up index format, set 1 - set 9 (XyWrite) [embedded command]
(valid in NB Win?)

IB Index break (break between alphabetical entries) (in NB Win?)[embedded command]

IN Include text file - how does this work? (XyWrite only) [embedded command]
IN [filename], depth, where [filename] is printer-ready. Includes a printer-ready file (with name [filename]) at current cursor position. Measure the vertical depth (in inches) in the printer-ready file and type that number as the depth. Files that contain a file-end marker are not printer-ready

IW 0/1 (entered at top of file by 'savcln' what does it do?) (XyWrite)[embedded command]

ix # Index Extraction command (XyWrite only) [immediate command]

JH Display Help frame with specified keyword. [function]
(NB DOS)

KD key diagram?

kilprn Stops printing to a printer (XyWrite only) [immediate command]

ldprn Load printer substitution table (XyWrite only) [immediate command]

ldrk Load program (recorded with function RK) on phrase key [immediate command
(XyWrite only)

ldsort Load sort-order file (XyWrite only) [immediate command]

ldsub Load printer substitution table (XyWrite only) [immediate command]

LF 0/1 Turn off/on display of graphics (for me, this makes no [function]
difference, either in calling files containing graphics or in turning off graphics within an open file.)

LN Line numbering. [This may be XyWrite only, or DOS only.][embedded command]
Defines how line number of each text line is printed in the margin.
This command has three formats:

```
ln 0            Begin line numbering
ln 1            End line numbering
ln m1,m2,...mn  Define Line Number
```

When you define ln, it is automatically on. 'ln 0' lets you turn before reaching the end of the file. You can then use ln 1 to restart line numbering from where you left off.

m1, m2, etc.., are one or more of the following:

o#,e# Offset: How far from the edge of the paper you want numbers to print for odd and even pages (where # is the number of inches). If you omit e#, XyWrite uses the value defined for o#. You must specify a value for o#.

i# Initial value - Starting line number (#). The default is 1.

d# Divisor - Lets you print every other line number (or some other alteration) by specifying a divisor. The default is 1, which means that every number is printed.

c Continuous number - Count numbers continuously from page to page. The default is to restart on every page.
b Blank lines - Do not count blank lines. A blank line contains only a carriage return, formatting commands, or both, but no text or spaces. The default is to number blank lines.
h Headers - Include running headers in the count. The default is to omit line numbers from running headers.
f Footers - Include running footers in the count. The default is to omit line numbers from running footers.

logoff Close any open files and log off current user (XyWrite) [immediate command]

login/logon (XyWrite) Log on network user and load any default settings associated with that user. Runs any XPL commands (e.g. Load commands, etc.) stored in file [username].LOG located in directory «VANL» directory - akin to a NBSTART.INT file for each valid username). (username 8 chars max.) Usage: logon [username] [immediate command]

LQ # Letter quality (XyWrite) - activates this mode on printers that support it. # can be: 1=Draft, 2=Letter, 3=Letter II, 4=Letter III, 5=Near Letter Quality Gothic, 6=NLQ Courier, 7=Utility, 9=Draft II (needs testing by someone whose printer supports it; mine doesn't.) [embedded command]

LT Toggle suppression of display of captured redlining login information (I can't make this work) [function]

LT (XyWrite - link text) Converts text in other file formats into XyWrite format and merges it into the displayed file at the current cursor location. [embedded command]

LW ('Function not recognized') [function]

MN Use next 2 chars: CD=Cartridges, FO=Fonts, MT=Match Type, SZ=Size (printing, but is it relevant in Windows?) (needs testing by someone whose printer supports it; mine doesn't.) [function]

MT Multiply (*) or divide (/) accumulated sum by selected number. (Doesn't work for me on sum done with 'x+y=z') [function]

nef New form (XyWrite only) [immediate command]

NK (Does nothing) [function]

NP No pause (printing) - cancels auto-pause (XyWrite only) (needs testing by someone whose printer supports it; mine doesn't.) [embedded command]

NR (Does nothing) [function]

OV (Does nothing) (was NB4) [function]

PA Pause printing (not tested) (XyWrite only) (needs testing by someone whose printer supports it; mine doesn't.) [embedded command]

PB Page begin string - same as PC (XyWrite only) (needs testing by someone whose printer supports it; mine doesn't.) [embedded command]

PC Printer control string (XyWrite only) (needs testing by someone whose printer supports it; mine doesn't.) [embedded command]

PI Insert printer string (still relevant in Windows?) (needs testing by someone whose printer supports it; mine doesn't.) And in NB4 it used to give trouble. I think it's an outdated, NB3 command.) [embedded command]

PP Put paragraph (XyWrite only) [embedded command]

prf Write printer file FO.TMP to disk (same as 'fo') [immediate command]
(XyWrite)
prints Print to screen (XyWrite only - like 'review') [immediate command]
Q1 smart quotes?
QH Display the Menu/Help screen whose keyword precedes cursor [function]
QP (Does nothing) ('File is missing {{') [function]
r2x Converts RFT:DCA format files into XyWrite (XyWrite only)[immediate command]
review review file (XyWrite only) (like print preview - doesn't work [immediate command]
in NB Win)
rmvscr Closes current window (XyWrite only) [immediate command

rpllf Make all but first line of file flush to left margin [immediate command]
RR Repeat records (XyWrite only) [embedded command]
rs Remove (empty) screen - (XyWrite/NB DOS only - not [immediate command]
NB Win)
RU (Does nothing) [function]
rv Same as 'review' (XyWrite only - doesn't work in NB Win)[immediate command]
savcln Save file under new name, with formatting codes at top [immediate command]
(XyWrite) Some of the codes are different from those
put at top of file by Format, Page Layout, Write Defaults.
(What is difference between this and Write Defaults?)
saverk Saves program created in Record keystroke mode [immediate command]
to a program file (XyWrite only)
SC ### (gets input at TF with 'savcln'. What does it do?) [embedded command]
SC 0/1 Turn Auto-Check off/on [embedded command, default]
(Doesn't work for me: executing 'd sc=1' and then typing
'persnl' (for 'personal' doesn't result in beep. Should it?)
SN Set numeric keypad to numbers (doesn't work in NB Win) [function]
SR Set record (XyWrite only) [embedded command, function]
stsgt Store current phrase library (XyWrite) [immediate command]
stspell Store temporary spelling dictionary (XyWrite only) [immediate command]
SU Subtract number cursor is on from the total. [function]
(Doesn't work for me on sum done with 'x+y=z')
SY n,m Symbol set for HP LaserJets (XyWrite in this form)[embedded command, default]
n is name of symbol set, m the pitch for
monospaced (use 0 for proportional fonts)
{DF SY=23Z,10,0,3}
(Why the difference between NB.DFL and XyWrite forms?)
T1-T9 TOC format, set 1 - set 9 (Used in NBWIN?) (XyWrite) [function]

TM Move to column element (0 = next; 1 = previous; 2 = top; [function]
3 = bottom) (doesn't work in NB Win)
TX # Extract TOC from source file, save it to target file (XyWrite) [function]
(Used in NB Win?)
UD Use dictionary (+ dict. name - error message: [embedded command]
'dict. path not found') (XyWrite only)
unload Unload spell checker [immediate command]
(says 'done' but doesn't work on personal spell checker)

updatetx	(XyWrite only)	[immediate command]
WH	(Does nothing)	[function]
WM	Wait for message (what does this do)	[embedded command]
WS1	Whole-space justify on	[embedded command]
x2r	(XyWrite only)	[immediate command]
XE	(Crashes program)	[function]
xlate	Strip high-bit characters - (XyWrite only, doesn't work in NB Win; in XyWrite, imports WordStar file)	[immediate command]
XS	Toggle display of markers affected by scoping (what does this do?)	[function]
XX	Define floating accent (must also be entered in AC Table in	[function]
XX,/	Stroke/slash accent	[XYWrite]
XX,ε	Macron accent	[XYWrite]
XX,”	Double acute accents	[XYWrite]
XX,≈	Cedilla accent pair	[XYWrite]
XX,□I	Caron accent pair	[XYWrite]
XX,□K	Ogonek accent	[XYWrite]
XX,□O	Breve accent	[XYWrite]
ZT	Reset stopwatch function to zero and start timer (programming) func ZT 0 resets elapsed time («VA$ET» to 0:00:00:00 [broken!]	[function]

Codes from list compiled for XyWrite - very unlikely to work in NBWin, included for completeness

Those with no description do nothing, and give no error message when I do F9 [command] F10. Error messages are in quotes.
I found some of these in Tyson's *XyWrite Revealed*; they are probably relevant only to DOS versions of XyWrite.

addtbl	Doesn't add column of figures	[immediate command]
ats		[immediate command]
atx		[immediate command]
b2g		[immediate command]
be	'command not recognised'	[immediate command]
big5in		[immediate command]
big5out		[immediate command]
bldidx	indexing	[immediate command]
bldseq	'requires sequence no. + create/replace flag'	[immediate command]
box		[immediate command]
bpt		[immediate command]
caf	call form - useful in nb6? ('caf [filename]' calls a file)	[immediate command]
cart	Call program file	[immediate command]
chgal 0-9	'the editing position is not in an object'	[immediate command]
chgedt		[immediate command]
clean		[immediate command]
clnp		[immediate command]
clnprs		[immediate command]
clrsum or	Clear sum in memory without inserting value into text	[immediate command]

cn	Force numeric lock off (‘command not recognized’)	[immediate command]
correct		[immediate command]
ddeexecute	‘command entry error’ / ‘not recognised’’	[immediate command]
ddeinitiate	crashes program	[immediate command]
ddepoke	‘command entry error’	[immediate command]
dderequest	‘command entry error’	[immediate command]
ddeterminate	‘command not recognized’	[immediate command]
dgb	(opens window with B in it)	[embedded command]
dgt	(opens window with T in it)	[embedded command]
dgw	(opens window with W in it)	[embedded command]
dlg	‘command entry error’	[immediate command]
dll		[immediate command]
docbld		[immediate command]
edf	forms	[immediate command]
edp	Call file cursor is on. ‘edp x’ calls file ‘x’ in directory Is this what this command is meant to do - am I missing something?	[immediate command]
EF #	Special printing effect	[embedded command]

Activates special printing effects for printers that support them. # is one the following values, or a combination of them:

1 Reverse	32 Not Assigned	1024 Double Underline
2 Outline	64 Not Assigned	2048 Overscore
4 Shadow	128 Double High	4096 Floating Underline
8 Inverse	256 Script Up	8192 Outline/Shadow
16 User Set	512 Script Down	16384 Wide

To activate more than one special effect at the same time, add values assigned to each effect. E.g., "ef 4224" activates double high characters with floating underline (where 4224 is sum of two special-effect values (128+4024). This command also turns off any other special effects that were active.

ET	Compute amount of time elapsed since ZT function and insert in text.	[function]
EV	Evaluate (XyWrite Mailmerge only)	[embedded command]
form	forms	[immediate command]
FS	Return cursor to last misspelled word and display spelling menu	[function]
g2b		[immediate command]
gcn	puts 2 GC deltas in text with low-ascii codes between	[immediate command]
getglobal	‘specify group/style name’	[immediate command]
gtgb	‘specify group/style name’ (same as getglobal)	[immediate command]
help	‘press Esc to remove menu’	[immediate command]
hkdef	‘not within a help link’	[immediate command]
imageinfo		[immediate command]
IN	Include text file -	[embedded command]

IN [filename], depth, where [filename] is printer-ready. Includes a printer-ready file (with name [filename]) at current cursor position. Measure the vertical depth (in inches) in the printer-ready file and type that number as the depth. Files that contain a file-end marker are not printer-ready

ja		[immediate command]

jc		
jg		[immediate command]
jrcom		[immediate command]
jrngrp	'requires number greater than zero'	[immediate command]
jrprs		[immediate command]
jrrmv		[immediate command]
js		[immediate command]
kilprn		[immediate command]
kiltyp	Stop current print job	[immediate command]
kp	(Same as kilprn)	[immediate command]
kt	Stop current print job (Same as kiltyp)	[immediate command]
ldprn	Load printer substitution table	[immediate command]
ldsort	Load sort-order file	[immediate command]
ldsub	Load printer substitution table	[immediate command]
linktx		[immediate command]
loado		[immediate command]
login	'logoff accepted'	[immediate command]
logoff	'logoff accepted'	[immediate command]
logon	'logoff accepted'	[immediate command]
logout	'logoff accepted'	[immediate command]
lp	Load printer file	[immediate command]
mail	'SMI not enabled in WIN.INI'	[immediate command]
mp		[immediate command]
msepos		[immediate command]
mwin		[immediate command]
nef		[immediate command]
newscrn		[immediate command]
NP	No pause (printing) - cancels auto-pause	[embedded command]
objdef		[immediate command]
objdel		[immediate command]
objfget		[immediate command]
objget		[immediate command]
objloc		[immediate command]
objupd		[immediate command]
ole		[immediate command]
olndef	outline?	[immediate command]
olnmov	Outline - 'specify PO, NO...'	[immediate command]
oo	'done' - but no change	[immediate command]
oos	Prompt 'Cannot run cmd;' then opens DOS window	[immediate command]
opnoln	'File not found'	[immediate command]
ox	'done'	[immediate command]
PA	Pause printing [not tested]	[embedded command]
PB	Page begin string - same as PC	[embedded command]
PC	Printer control string	[embedded command]
PP	Put paragraph	[embedded command]
prints		[immediate command]
prn	'printing file "review.tmp"' (but it doesn't)	[immediate command]
prs	(Opens new screen) (same as prints)	[immediate command]
pstring		[immediate command]

putc		[immediate command]
r2x	Converts RFT:DCA format files into XyWrite files	[immediate command]
rer	(Opens note window)	[embedded command]
review	review file (like print preview)	
rex	(Opens note window)	[immediate command]
rfrp		[immediate command]
rmvscr		[immediate command]
rplfil	('Command entry error')	[immediate command]
RR	Repeat records	[embedded command]
rs	Remove (empty) screen	[immediate command]
rv	Same as 'review' s	[immediate command]
r2x		[immediate command]
saverk	Saves program created in Record keystroke mode to a program file	[immediate command]
se[/c] range|string|	Search directory - searches through a series of file names separated by commas (range) for the text (string). 'searcha' and 'sea', 'searchba' and 'seba', can also be used You must do search from blank window; do: F9 ne F10 before executing search command. Switch: /c tells program to count number of times string appears, but not to stop at each match.	[immediate command]
setdm	('Command entry error')	[immediate command]
setlgd	('Command entry error')	[immediate command]
setp	'Select Windows printer': dialog,XyWrite buttons	[immediate command]
setpos	(outlines)	[immediate command]
shohyp	Show effect of current hyphenation rules	[immediate command]
showdlg	'command entry error'	[immediate command]
shwpg	'requires source file and target file'	[immediate command]
sm	(a numeric command - 'Sum requires 1 number')	[immediate command]
SR	Set record	[embedded command, function]
stspell	'dict. path not found')	[immediate command]
stxlate	'command not recognised'	[immediate command]
testob	(opens new screen)	[immediate command]
tmbld		[immediate command]
tol	('Need: TOL digit, name, mnemonic1,...,mnemonicn. mnemonic1,...,mnemonicn')	[immediate command]
ty	Print to printer	[immediate command]
tyf	Print to file	[immediate command]
type	Print to printer	[immediate command]
typef	Print to file	[immediate command]
types	Print to screen	[immediate command]
tys	Print to screen	[immediate command]
undel		[immediate command]
unl	unload feature	[immediate command]
unload	unload feature	[immediate command]
updatetx		[immediate command]
uwf		[immediate command]

vbx	(‘Command entry error’)	[immediate command]
wordchk	‘no alternates for [any word]’	[immediate command]
wprof	(‘Command entry error’)	[immediate command]
xlate	Strip high-bit characters - (imports WordStar file)	[immediate command]
x2r		[immediate command]
xe	Search filelist for text	[immediate command]
xea	Search filelist for text, absolute	[immediate command]
	(These 2 freezeNB)	
xm	(‘Function requires 1 number’)	[immediate command]
xwdll	(‘Command argument not recognized’)	[immediate command]
zap	‘0 words, 1 questionable’ (or crashes program)	[immediate command]

Appendix: NB DOS XPL Error Messages, and List of XyWrite Error Messages

. **General Introduction**. XPL does not precompile, nor flag errors; it just tells you there is one (not always that). Its error messages are generic─they do not always tell you what the error in the particular case is; and they do not cover everything.

A distinction needs to be made between errors that can occur when writing a program and those that can occur during an attempt to run the program. The latter may indicate, not that there is something wrong with the program, but that it cannot perform its task on the particular occasion; e.g., a Search command that cannot find any instance, or any more instances, of the string being sought.

1. **Error Messages** Most errors, when they occur, do not display messages on the prompt line; the following are those that are commonly displayed:

'Invalid Format command'

An opening command bracket (with or without the closing bracket) has been entered in Normal mode. The remedy is to switch to Expanded mode, which should always be used when writing a program. No message is displayed in the case where there is a closing bracket that does not have a matching opening bracket. But there is a quick and easy way to detect a surplus closing bracket. In Expanded mode enclose the entire program in the IV command: insert «iv at the start of the program and » at the end; then switch to Normal mode. If the program's brackets are OK, the whole program file will disappear; if there is an unmatched closing bracket, it will be shown.

'No command' or **'Illegal command'**

A command has been incorrectly entered on the action line. E,g., **BC** ca myfile.doc**XC** (a space before 'ca'

BC camyfile.doc**XC** (no space between 'ca' and filename
BC ra myfile.doc**XC** ('ca' mistyped as a non-existent command 'ra')

'Command entry error'

i. A program call has been wrongly entered.

a. The wrong call was used, e.g.,
'pv' instead of 'is' (or vice versa)
'sv' instead of 'sx' (or vice versa)

See explanation of the difference between the calls in Chapter *XPLCALLS.DOC*

b. The call was mistyped, e.g.,
'=' instead of '=='
'$wn' instead of 'va$wn'

'$wn' contains the number of the active window, but it requires the value command (va) to display it, or to save it to a phrase, as in «sx01,«va$wn»»

ii. An attempt has been made to perform a *string operation on a numerical value*, or a *mathematical operation on a string*. String operators are used for manipulating strings of characters, whether literal or numerical, as in conjoining them (e.g., «sx01,«is02»+«is03»»), or in locating one string within another (e.g., «sx01,«is02»ε«is03»»). They cannot be used for evaluating them (e.g.,«sx01,«is02»/«is03»» - one string cannot be divided by another.)
«sx01,«pv02»/«pv03»» could be valid if what had been saved to phrases 01 and 02 had been numerical values.

'Mismatched operands'
An attempt has been made to compare a string to a value. E.g., «if«pv01»==«is02»»

'Label not found' The «lb...» has been omitted, or it does not exactly match the «gl...» in either spelling or case. Not every «lb...» needs a corresponding «gl...», but every «gl...» needs a corresponding and exactly matching «lb...»; otherwise the program cannot jump to the correct place.

'Need ID & expression' A syntactical error has been committed with 'sv' or with 'sx'. The comma may have been omitted, e.g., «sv01Yes» instead of «sv01,Yes». Or the phrase key has been identified, but the expression has not, e.g., «sv01». *Note*: there is one exception to this: if a block of text has been defined, «sv01» is correct and saves the block to phrase 01.

In some situations, if either an opening bracket or a closing bracket is missing, e.g. «pv01 or pv01», the error message will appear.

'No «ei»'
This is a common error. Every «if...» clause must be closed with an «ei» call. If it is not, **Nota Bene** has no way of knowing where the conditional clause ends, and cannot execute the program correctly.

'Too many program calls'
An endless loop has been created. E.g., you have included within a program the command to run itself.

'Repeat w/alphanumeric'
A label name is missing from a «gl...» or an «lb...» call. This is the same error message that you get if you try to assign a phrase to a regular phrase key by striking Alt+F2 and then strike any key other than a letter or a number.

2. **Identifying Errors**

There are two Error variables, which should not be confused with each other: ER and $ER. The former has only 2 values, the latter more than 300.

i. Whenever an error occurs ER is set to True, and as soon as the next command is given it is reset to False. Thus, for example, if a program contains a search command for a specified word, and if no instance (or no further instance) of the word can be found, the command generates an error; and the program can include an instruction what is to be done if an error is met. E.g., **BC** se elephant **XC** «if«er»»«ex»«ei»: if no instance (or no further instance) of 'elephant' is found, the program is to to terminate.

ii. Also, whenever an error occurs, $ER is set equal to an XPL error number. The number can be found, and the error can be identified, in either of two ways:

a. By executing the command **BC** va $er**XC**. That will display in your screen file (when in Normal mode) a delta followed by a number, looking like an inserted counter; the number is the number of that particular error. In the case of the above example, what you will see in the file is «VA$ER»10, 10 being the error number that means 'Not found'. If $ER has no value, the delta will appear in the text followed by a 0. In either case the cursor will be located immediately after the number, and a single stroke of the Backspace key will remove both number and delta (again, just as with a counter in the text).

b. In a program the value of $ER can be saved to a phrase, as in «sx150,«va$er»», which can then be displayed, discarded, or evaluated, like any other phrase.

CAUTION It is a feature of **Nota Bene** that the value of $ER is always reset to 0, when the next command is given. Therefore, unless the value is read or saved immediately after the error occurs, it will be lost; and, if the value is saved to a phrase, it should be saved to one that is higher than 100; otherwise it will be lost when you leave the program: «sx150,«va$er»» will survive the program in which it occurs: «sx50,«va$er»» will not.

XyWrite Error Messages Listed Numerically (from the XYWWWEB.U2 File)

1 Filename already exists.
2 Function requires an empty window.
3 |Do you want to delete marker? (Y/N)
4 Command entry error.
5 Application error .....
6 NB.DFL
7 Function requires an open file or filename.
8 Type password and press Enter.
9 □Disk(ette) is full.
10 Cannot find item.
11 There is no command on command line.
12 Command is not recognized.
13 Separator is missing.
14 Function requires selecting a block.
15 Help frame too large &
16 Printing file &
17 Press letter or number.
18 There is not enough memory to perform function.
19 There is no macro assigned.
20 Name
21 Press Esc to cancel the selection.
22 Too many mode entries.
23 Search string is not recognized.
24 Formatting command is not recognized.
25 Characters were lost during typing.
26 Source and target filenames cannot be identical. &
27 Cannot create temporary file.
28 Logon completed.
29 NBSTART.INT
30 --Shift+F1 to save.
31 Printer error.
32 Read error.
33 Cannot find specified field.
34 DIRECTRY.TMP
35 |Change ? Y=yes, N=no, S=stop here, O=one more3
36 Change/Verify command canceled.
37 Done
38 (none)
39 PATH=
40 COMSPEC=
41 Too many soft fonts.
42 Overflow file full, printing terminated.
43 JM
44 Type "+" to continue printing.
45 Default drive/directory 1 Directory of 2
46 1 Files 2 Char. 3 Free
47 Specify filename to save selected block.
48 REVIEW.TMP
49 FO.TMP
50 INDEX
51 TABLE
52 Cannot continue scrolling.
53 Cannot modify protected text.
54 Correct format is @dat(mm-dd-yy)
55 Cannot choose soft hyphen character.
56 Going to formatted view with page breaks.
57 Cannot print to file while printing.
58 Not enough disk space to print document to file.
59 |File open--exit anyway? (Y/N)
60 Logon is not recognized.
61 CLOSED
62 OK
63 a:QUIT1.TMP
64 Current command is canceled.
65 No merge records extracted.
66 Preparing to print. Please wait.
67 Document summary corrupted.
68 Translate command requires 2-character string.
69 MLCS
70 Error loading macros.
71 <DIR>
72 Function is not recognized.
73 ON
74 OFF
75 Cannot sort records larger than 3000 characters.
76 Math function requires equal sign.
77 Place cursor on a number to do arithmetic.
78 Not enough memory for new window.
79 Not enough memory. Counters are not accurate.
80 Mathematical result is too large.
81 No more windows are available.
82 Mnemonic for translation not found.
83 @UPR cannot convert numbers.
84 Cannot close last window.
85 There is an extra Start Command (E) in text.
86 There are too many values for the current command.
87 Tab settings are not in numerical order.
88 |Error writing quitn.tmp files--try again? (Y/N)
89 Error writing index file.
90 @SIZ cannot convert numbers.

91 Function requires one number.
92 Function requires numeric values only.
93 Cannot nest embedded commands.
94 Using a space as the leader character.
95 Specify mode.
96 Text saved.
97 .BAK
98 @INT only converts numbers.
99 Cannot move cursor outside command window.
100 Not enough memory for sorting.
101 Specify filename to run.
102 @C2X translate command requires string.
103 Help file is too big.
104 Use cursor keys to adjust window, press Enter to end.
105 Complete modification of tab ruler.
106 Error closing index file.
107 Cannot find index file.
108 Error reading index file.
109 Field does not exist.
110 Customization file requires a file label.
111 @CNV only converts 2 character strings.
112 Fill in this field.
113 |Do you want to exit? (Y/N)
114 on
115 Cannot print to screen while printing.
116 File corrected--save it?
117 Disk full writing to overflow file--close file without saving? (Y/N)
118 |Continue previous correction? (Y/N)
119 Unrecognized line in custom file:
120 Function is not available while printing.
121 Menu/Help routine is missing named &
122 Dictionary is not loaded.
123 Index item is too large.
124 Typethru is on.
125 Mode value is not recognized.
126 Press key to insert (or run) macro.
127 |Filename already exists--overwrite it? (Y/N) &
128 |Type Y or N.
129 |Stop program? (Y/N)
130 Load UIF file before setting BZ value.
131 Specify path name.
132 |Do you want to delete? (Y/N) &
133 Default format is not recognized.
134 Specify program name.
135 Cannot run command.
136 Not enough memory. Clear some macro keys and try again.
137 Cannot append another column selection.
138 Function is not available in forms mode.
139 Cannot copy selected text into command window.
140 Specify filelist|search string|.
141 Specify program name and macro key.
142 Too many program calls or program loops.
143 Mismatched logical or numeric operands.
144 More than one unary operator.
145 Go to Label command requires a Label command.
146 Cannot find "End If." for
147 Function requires ID and expression. &
148 Cannot use more than ten shift keys in keyboard file.
149 Cannot use more than 20 tables in keyboard file.
150 Cannot use more than six shifting states in keyboard file.
151 Cannot use more than four toggle definitions in keyboard file.
152 Read-only file on p
153 File is hidden on p
154 System file on p
155 Name is volume label on p
156 Name is subdirectory on p
157 Cannot be written to on p
158 Cannot copy to same file.
159 Sort record is too large.
160 No more files that match specification.
161 |Still printing--exit anyway? (Y/N)
162 Item is currently not available.
163 Disk is full writing to overflow file--free up space on drive c:ʒ
164 No alternates for &
165 Invalid default setting &
166 Column format is not recognized.
167 Cannot find style. &
168 File contains too many styles.
169 Enter number greater than zero.
170 Working...
171 Function requires source file and target file.
172 |Keyword Found, Action? C=continue, O=open, N=next file, S=stop
173 Checksum error.
174 Cannot mix relative and absolute values.
175 Relative values are not acceptable.
176 Frame type is not recognized.
177 File contains an incomplete frame definition.
178 File has an unbalance {{ }}.
179 File is missing {{.
180 Window specification is not recognized.

181 Screen length cannot exceed 70 lines.
182 Function requires two open files.
183 |Disk is full writing to overflow file--stop current operation? (Y/N)
184 AUTO
185 Definition is missing from printer file.
186 Width table is missing from printer file.
187 Substitution table is missing from printer file.
188 Inserted text found. Use filename to print to screen.
189 |Sort files by F=File Name,E=Extension,D=Date,S=Size,P=Path,R=Revision,H=Header,T=Use Tab,N=No tab3
190 |Enter drive letter to store on.
191 |Diskette full. Insert new diskette and type Y to continue.
192 |Save files as quitn.tmp? (Y/N)
193 Write protect.
194 Unknown unit.
195 Drive is not ready.
196 Unknown command.
197 Data error.
198 Bad request structure.
199 Seek error.
200 Unknown media.
201 Sector not found.
202 Printer out of paper.
203 Write fault.
204 Read fault.
205 General failure.
206 Save edits to &
207 Composition stopped--disk full writing to overflow file.
208 Table values are not recognized.
209 Value in counter is not recognized.
210 Counter value must be between 0 and 14.
211 Roman numeral is not recognized.
212 Place cursor on marker.
213 Invalid function. #
214 File not found. &
215 Path not found. &
216 Too many open files.
217 Access denied.
218 Invalid handle.
219 Memory control blocks destroyed.
220 |Type new character.
221 Invalid memory block address.
222 Invalid environment.
223 Invalid format.
224 Invalid access code.
225 Invalid data.
226 DOS error.
227 Invalid drive.
228 Cannot remove current directory.
229 Not same device.
230 No more files.
231 Function does not accept arguments.
232 Not enough memory. Page numbers are not accurate.
233 Not enough memory to calculate page breaks.
234 New Form requires new filename and filename of master form.
235 Unrecognized characters in filename.
236 Decimal point cannot equal numeric or argument separator.
237 Cannot look up double words.
238 No files selected.
239 Press Esc to return to Cartridge List.
240 RFT conversion failed.
241 Cannot insert row outside of table.
242 Substitution tables must be put inside printer file.
243 □Error loading overlay.
244 □Overlay mismatch.
245 □Not enough memory for overlay.
246 □Error reading EDITOR.EXE.
247 Personal dictionary is not loaded.
248 Function is not available while Track Changes is on.
249 Function canceled.
250 Function is not available in expanded view.
251 |Change? Y=yes, N=no, S=stop here, O=one more, U=undo3
252 Function canceled.
253 Cannot find redlined text.
254 Making temporary file. Do not remove x:3
255 OK to remove x:3
256 Overlay ID not recognized.
257 word.ovr
258 □Disk is full. Cannot create overflow file.
259 items processed
260 Type a character.
261 SPELL.TMP
262 |Terminate column select operation? (Y/N)
263 .SAV
264 [UNTITLED]
265 OEIDCBHFLRNS
266 Menu/Help file not loaded.
267 Esc 1,Single 2,Double 3,* 4,Move cursor 5,Delete 6,New character
268 Function is not available in tables.

269 Accent is not defined.
270 SHELL=
271 No text data in clipboard.
272 Press space to continue undeleting.
273 Composition stopped--too many page elements.
274 Do you want to perform this change? Y/N/C
275 GSLIB
276 SWGS.LIB
277 Logoff accepted.
278 lexis
279 Too many TE controls.
280 Not allowed in Ibid. record.
281 Specify group\style name.
282 Style is too big.
283 JANUARY
284 FEBRUARY
285 MARCH
286 APRIL
287 MAY
288 JUNE
289 JULY
290 AUGUST
291 SEPTEMBER
292 OCTOBER
293 NOVEMBER
294 DECEMBER
295 TRUE
296 FALSE
297 PCLEXAM.DLL
298 .OVR
299 AMPM
300 DMY
301 HVFBTLRNAXC‘ M
302 DCRL
303 YNI
304 OoIAsRXNSCrc
305 TGCDLRIHXZ‘ A
306 |Double word encountered--delete second occurrence? (Y/N) &
307 IXPNCH
308 NTS
309 CAPS
310 SHIFT
311 ALT
312 CTRL
313 ALNSXWO
314 Press Esc to remove menu.
315 words, * questionable
316 String too large for filter.
317 Reset &View
318 changes
319 |Cannot recover changes--proceed anyway? (Y/N)
320 PV cannot find macro.
321 Not currently implemented.
322 FRWNLAD
323 Nothing has been deleted.
324 PV error converting to string.
325 occurrences
326 Cannot compare number to string.
327 |Recording keystrokes.
328 No keystrokes have been recorded.
329 Table has not been loaded.
330 #X function requires letter or number.
331 Proceed
332 Sharing violation.
333 Lock violation.
334 Invalid disk change.
335 FCB unavailable.
336 XX
337 Close text command window.
338 Cannot read/write string.
339 Buffer at maximum size.
340 No memory to expand buffer.
341 Unable to open menu window.
342 Cannot get printer device.
343 Unable to start print job.
344 "JA R1,...,Rn Ri = >n | n | n1-n2"
345 JA arguments must be in descending order.
346 Show journal.
347 View
348 EB Script %s incorrect number of arguments
349 TEXT.LIB
350 Network request not supported.
351 Remote computer not listening.
352 Duplicate name on network.
353 Network name not found.
354 Network busy.
355 Network device no longer exists.
356 Network adapter hardware error.
357 Net BIOS command limit exceeded.
358 Incorrect response from network.
359 Unexpected network error.
360 Incompatible remote adapter.
361 Print queue full.
362 Not enough space for print file.
363 Print file was deleted.
364 Network name was deleted.
365 Access denied.
366 Network device type incorrect.
367 Network name not found.

368 Network name limit exceeded.
369 Net BIOS session limit exceeded.
370 Temporarily paused.
371 Network request not accepted.
372 Print or disk redirection is paused.
373 Error reading text library.
374 Too many different outlines.
375 More than one initial outline element.
376 Reference to undefined id.
377 Error in outline structure.
378 Unrecognized KY type.
379 Could not find specified outline.
380 File exists.
381 Could not write paragraph index/outline.
382 Cannot make directory entry.
383 Fail on INT 24.
384 Too many redirections.
385 Duplicate redirection.
386 Invalid password.
387 Invalid parameter.
388 Network data fault.
389 Read-only file.
390 Please log on.
391 File overflow. Not all files are listed.
392 File not found.
393 Cursor must be within a database paragraph
394 MOUSE
395 Cannot turn on mouse pointer. Need driver or mouse.
396 Not enough memory to use mouse.
397 Not enough keys for mouse.
398 Cannot move selected block from read-only file.
399 Cannot sort on a selected column.
400 About ...
401 EDITOR: Close
402 Ok to close window?
403 EDITOR: End Session
404 Save files before ending?
405 EDITOR: Error Message
406 untitled
407 EDITOR: Quit
408 Nota Bene HELP
409 Default drive/directory
410 Directory of
411 |Print selected block? (Y/N)
412 |Print directory? (Y/N)
413 |File was modified--abandon changes? (Y/N)
414 Specify filename.
415 |Press (B) to select soft font, Enter to save.
416 Use Style Manual or Citation Menu to edit program-generated content.
417 percent
418 tenths
419 hundredths
420 thousandths
421 Rules are not properly nested.
422 YES
423 NO
424 Subdirectories
425 |Save displayed file? (Y/N)
426 |File is already open: G=Go to Existing, O = Open Copy, R = Read Only,C=Cancel
427 Dialog items overlapped.
428 HVNU
429 Fonts are not loaded.
430 Cannot find graphic file. &
431 Not enough memory to load graphic file. &
432 Graphic is not in PCL format. &
433 Graphic is not in PCX format. &
434 Graphic is not in TIFF format. &
435 Cannot load PCL graphic file. &
436 Cannot load PCX graphic file. &
437 Cannot load TIFF graphic file. &
438 Scale is out of range.
439 XYIMAGE=
440 Graphic file is not bilevel. &
441 Remove filename from command line.
442 Unspecified image size &
443 UNLOAD command not recognized.
444 TYSIPAPOLBIO
445 CDMTSZ
446 Name Wt Sty Size Pitch Or Symbol Set
447 C S T F F N D N V E R V N A F SUPAFIFSSLMVNCLSNEOLX LCZ N T N L X Y R L S P U P W N H H O N-NASCNACLOAXFWSCEIWHOPDENE O S O A F P I D O V S M B CRLGGCRHRFLBVEAFRESEXALFIO NRSSXPALCLPRETXOXULI
448 Command switch is not recognized.
449 Frame type is not recognized.
450 Frame command requires semicolon.
451 |Select (P)erm, (T)emp, (B)uild, space to clear, Enter to create file.
452 Not enough printer memory to download fonts.
453 SPPRSUHLHYSOKBP+FOMNSFDGU1 U2U3U4U5U6U7U8U9DFXDEBUI
454 Loading font file: &
455 Do you want to truncate? Y/N

456 Too many cartridges selected.
457 INWTLTABCRLBRSIS
458 PGPEFAFNRHRFCTSNSTNO
459 Style name is not recognized.
460 Not enough memory for command.
461 Cannot find border style.
462 Value is too large.
463 SCRFONTS.BIN
464 Function requires style name.
465 [Command Window]
466 Font not available.
467 |Load fonts into the printer? (Y/N)
468 Move the cursor out of the selected block.
469 Bad font name.
470 Cannot nest embedded commands.
471 Command argument is not recognized.
472 Not enough memory to load screen fonts.
473 PTB
474 MDCGHGHPINEGMCVG
475 System cannot support graphic mode.
476 Cannot select from this menu.
477 Program mode is on. Press [Ctrl]+[Alt] + [:] to exit.
478 Not enough memory for printer matches. Reload printer file.
479 Not enough memory to load the width table.
480 Cannot select across page elements.
481 Cannot search for invisible commands.
482 Style name already exists.
483 Function is not available in command window.
484 Invalid line for dialog box &
485 Autosaving to TMP file...
486 Type EXIT to return to Nota Bene.
487 Number is out of range.
488 SX command requires a number.
489 Selected column is too large.
490 |Printer not ready--try again? (Y/N)
491 GCI-OGCI-RGCI-NGCI-P
492 Host communication error.
493 General Callable Interface control block is not recognized.
494 |Save on host? (Y/N)
495 |Print on host? (Y/N)
496 |Merge file from host? (Y/N)
497 AUTOSAV1.TMP
498 Function is not available in expanded view.
499 |Change case? L=lowercase, U=uppercase, F=first letter, O=OK
500 Font file is not recognized.
501 Zoom only available in graphic view.
502 |Press appropriate key to continue.
503 Application error .....
504 Application error .....
505 Application error .....
506 Not enough memory for dictionary.
507 Application error .....
508 Cannot find dictionary.
509 No synonyms in dictionary, alternates provided.
510 Application error .....
511 Invalid word construction.
512 Too many dictionaries are open.
513 Dictionary cannot be updated.
514 Personal dictionary is full.
515 Dictionary already exists.
516 Dictionary name is not recognized.
517 Cannot read dictionary.
518 Dictionary not initialized.
519 Invalid dictionary function.
520 Invalid token in dictionary list.
521 Invalid dictionary word
522 Illegal dictionary input flag combination.
523 Missing or illegal value in dictionary parameter area.
524 Dictionary passback area too small.
525 Cannot add to temporary dictionary.
526 Too many dictionaries.
527 Dictionary contains bad data.
528 Dictionary memory allocation error.
529 Dictionary file not found.
530 Dictionary path not found.
531 Cannot open dictionary file.
532 Dictionary access denied.
533 Dictionary sharing violation.
534 Dictionary general file I/O error.
535 Variable Name exists
536 File converted to RFT:DCA format.
537 Conversion canceled.
538 Source file is empty.
539 Cannot read source file.
540 Cannot write to target file.
541 File converted to Nota Bene format.
542 Conversion canceled.
543 Source file is missing RFT:DCA end unit.
544 Include unit requires text.
545 Cannot find outline directory.
546 Cannot download fonts to non-HP printer.
547 XXLPCPRPLCCCRCPCTM
548 XXTPCPBPTCCCBCPC
549 Text link is completed.
550 Cannot open link file.

551 Cannot read link file.
552 Cannot open conversion file.
553 Cannot write to conversion file.
554 Data conversion is not valid.
555 Cannot open exception file.
556 Cannot write to exception file.
557 Not enough memory for conversion.
558 Document is not recognized.
559 Not enough disk space for output.
560 Conversion document is too large for target.
561 There may be an error in conversion.
562 INDITWPIPTDPCMMMCIDDLI
563 Unrecognized page number value.
564 Error saving text macro.
565 New DR setting ignored (overflow file is open).
566 Cannot modify EG setting while in graphic view.
567 Wildcards must be in the same order on both sides of a change.
568 Function code is not recognized.
569 REVERSE.TMP
570 Function is not available in graphic view.
571 Cannot change display mode of read-only directories.
572 Printing done.
573 TYDESCCRIMRVRIRT
574 |Suspending Nota Bene--save files to AUTOSAV.TMP files? (Y/N)
575 Zoom to &
576 Mouse not supported for use in this window.
577 Cannot load printer file when printing.
578 Check margin settings (PW, IP, GU, LM, RM, etc.)
579 Function not available on command line.
580 Unrecognized macro id.
581 Top margin on this printer must be at least &
582 Line is too large for graphic view.
583 Function not available with column selected.
584 Graphic images are ignored for this printer.
585 Too many letters in file name.
586 Function requires selected text in current or previous window.
587 occurrences displayed.
588 Form field not found--no longer in forms mode.
589 Line is too large to be printed in image mode.
590 Cannot print read-only directory to screen.
591 Cannot open marker--change to expanded view.
592 Press hot key again to go into TSR--press Esc to return to Nota Bene.
593 Cannot create RFT:DCA document with .DOC extension.
594 Error in <BR> command--command ignored.
595 Error in <WM> command--command ignored.
596 |Edit port, printer file or printer name. F9 when done (Esc=exit).
597 |Enter screen/printer fonts. Press F9 when done (Esc to exit).
598 |Enter Printer Fonts. Press F9 when done (Esc to exit).
599 |Continue: Forward (Alt down-arrow), backward (Alt up-arrow) (Esc to exit).
600 |Edit dictionary file, then press F9 (Esc to exit).
601 |Type text for TOC, then press Shift+F1 (Esc to cancel).
602 |Type text for index, then press Shift+F1 (Esc to cancel).
603 |Position cursor and press F9 (Esc to exit).
604 |Select the text to protect, then press F9 (Esc to exit).
605 |Select text to keep together, then press F9 (Esc to exit).
606 |Move cursor to a footnote, then press F9 (Esc to exit).
607 |Position cursor for endnotes, then press F9 (Esc to exit).
608 |Place cursor on footnote to be labeled, then press F9.
609 |Move cursor to the counter marker, then press F9.
610 |(X1) F5=heading, F6=selection, F7=insert text, Esc=exit.
611 |(X2) F5=heading, F6=selection, F7=insert text, Esc=exit.
612 |(X3) F5=heading, F6=selection, F7=insert text, Esc=exit.
613 |(X4) F5=heading, F6=selection, F7=insert text, Esc=exit.
614 |(X5) F5=heading, F6=selection, F7=insert text, Esc=exit.

615 |(X6) F5=heading, F6=selection, F7=insert text, Esc=exit.
616 |(X7) F5=heading, F6=selection, F7=insert text, Esc=exit.
617 |(X8) F5=heading, F6=selection, F7=insert text, Esc=exit.
618 |(X9) F5=heading, F6=selection, F7=insert text, Esc=exit.
619 |(X1) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
620 |(X2) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
621 |(X3) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
622 |(X4) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
623 |(X5) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
624 |(X6) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
625 |(X7) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
626 |(X8) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
627 |(X9) F5=word, F6=selection, F7=insert text, F8=subentry, Esc=exit.
628 |Turn off: P=page borders, A=all other borders (Esc=exit.)
629 |Select directory. F1=display files,F2=change dir,Esc=exit.
630 |Unprotect this block? (Y/N) (Esc to exit.)
631 |Continue searching? (Y/N) (Esc to exit.)
632 |Continue searching from top? (Y/N) (Esc to exit.)
633 |Not found, continue searching? (Y/N) (Esc to exit.)
634 |Not found, continue searching from top? (Y/N) (Esc to exit.)
635 |Set bookmark: F=first, S=second, Esc=exit.
636 |Go to bookmark: F=first, S=second, Esc=exit.
637 |Allow this block to break across pages? (Y/N) (Esc to exit.)
638 |Delete? (Y/N) (Esc to exit.)
639 |Mark entry: T=toc, I=index. (Esc to exit.)
640 |Press K=keycodes, I=identify key, J=jump to table. (Esc to exit.)
641 |Press any key for its key code, Esc to exit.
642 Close command window before selecting this window.
643 Keyboard did not accept new repeat rate settings.
644 Invalid repeat rate settings--new values ignored.
645 Fields contain formatting commands--no longer in forms mode.
646 and
647 Cannot enter specified character in forms mode.
648 OK to edit file. F9=continue, Esc twice=exit.
649 |Find latest files: C=current dir,S=incl. subdirs,E=entire disk
650 Reformatting for new printer file or default setting...
651 Type size is too large--default size used.
652 Not enough space to print all line numbers.
653 Image mode printing is not allowed for this printer.
654 Cannot alter image printing state while printer is active.
655 Function not available for selected row/column in table.
656 Redlining not available when editing markers.
657 Page range not recognized--no page printed.
658 Screen/printer font mismatch--display is incorrect.
659 pronoun
660 verb
661 noun
662 adj
663 adverb
664 prep
665 interject
666 conj
667 Masc noun
668 Fem noun
669 verb pron
670 trans verb
671 intrans verb
672 plural noun
673 Endnote marker was inserted at the top of the file.
674 OK
675 Cancel
676 Options
677 Filename
678 Path
679 Help
680 Saving...

681 This is the help for "Custom Entry"
682 |Update Bitstream typeface fonts for Nota Bene? (Y/N)
683 |Update font list? (Y/N) (Soft fonts and cartridges will be unloaded.)
684 Application error .....
685 Application error .....
686 Application error .....
687 Application error .....
688 To restore the view, press Ctrl+Shift+V or click on the System menu bar.
689 One character only.
690 Not allowed. (Press Esc to exit.)
691 <TAB>
692 <CR>
693 Canceled.
694 File name extension used:
695 File name extension added:
696 |You typed a period with no extension--okay? (Y/N)
697 File will be saved without extension.
698 Specify outline counter or 0 to turn off outlining
699 Load either the main keyboard or the mini menus.
700 Select a file.
701 Creating SW.DFL failed. Cannot make permanent changes.
702 Creating SW.DFL...
703 Access to SW.DFL denied. Cannot make permanent changes.
704 File could not be created.
705 Edit Style,Define Style,Replace Style
706 copied from
707 DEFAULT
708 File contains default style.
709 Default style inserted.
710 This is a new untitled file.
711 .TPL
712 Nota Bene
713 Open File
714 Open Form
715 Press Ctrl+Shift+M to store this file.
716 Converting to native code page...
717 To view other windows, press F6.
718 Please select a file from list.
719 Full View
720 Punctuation not allowed in Style name.
721 Searching...
722 (A to Z)
723 (Most recent first)
724 (Largest First)
725 .PGM
726 Cannot convert formats and switch to new file.
727 Selected text saved as:
728 .RFT extension added to file name.
729 Conversion files not properly installed.
730 Cannot find conversion files.
731 Conversion file not installed for
732 Internal menu error.
733 File saved to
734 Please select a format.
735 Converting from
736 Condition longer than 55 characters must be saved to a file.
737 Converting to
738 Excel
739 Lotus
740 Unframed
741 Uncropped
742 Compressed
743 Gray-scale
744 Color
745 Could not find file:
746 Conversion file not found.
747 CVT.RES file is missing in
748 Converting to TIF file. This may take some time...
749 Framed
750 Title
751 Title for Framed Graphic
752 Caption
753 Caption for Framed Graphic
754 Must specify both dimensions.
755 Cropped
756 GFX
757 8 characters maximum in label.
758 &Protect
759 Put cursor on "Link text" (LT) marker.
760 Un&protect
761 Operation canceled.
762 Linked text protected.
763 Linked text deleted.
764 &Filename Size Date Time
765 Display doc &info (Selective files)
766 Display doc &info (All files)
767 No files found in the selected range.
768 Filename
769 Size
770 Saved
771 Time
772 Author

773 Saved-by
774 Cr-date
775 Cr-time
776 Proj-no
777 Rev
778 Reten
779 Comment
780 Keyword
781 Cannot display more than 4 fields.
782 Press Update to calculate final size.
783 Final size is based on cropping and scale.
784 Width scaled for frame width:* *less gutter.
785 Path already exists.
786 Error in deleting directory.
787 Directory was deleted.
788 Error in creating new directory.
789 New directory has been created.
790 Author name exceeded 40 characters.
791 Project Number exceeded 20 digits.
792 Document retention exceeded 4 digits.
793 Comment exceeded 44 characters.
794 Keyword exceeded 65 characters.
795 Word saved to Document Info.
796 Text saved to Document Info.
797 Cannot use numbers as labels - labels must include a letter.
798 Use a name that does not contain a comma or parenthesis.
799 Print file to local printer.
800 Print file to host printer.
801 Cannot print selected text or current page in different window or file.
802 Cannot chain displayed file, selected text or current page.
803 Print entire file?
804 Cannot chain selected page numbers.
805 Cannot print selected text to screen.
806 (Example) 1-3/6-8
807 Print to Screen
808 Print to File
809 Pause after pages
810 Application error .....
811 Multiple copies
812 Non-collate
813 Quality (Draft,...)
814 Reverse order
815 Simplex/Duplex
816 Printing to printer...
817 Printing to screen...
818 Printing to file...
819 Image mode turned on.
820 File does not exist:
821 Please close this file and try again:
822 Error opening file:
823 Printer file has been loaded:
824 Application error .....
825 Application error .....
826 Application error .....
827 Port
828 Printer File
829 Printer Name
830 Canceled (tried to close wrong file).
831 Please select a port and try again.
832 Please select a printer file and try again.
833 Description exceeded 34 characters. Extra characters were truncated.
834 No printer file found in printer path.
835 Please load a printer file and try again.
836 Cannot open printer file.
837 Internal menu error (tried to close wrong file).
838 No soft font files found.
839 The extension SFL was added.
840 The extension was changed to SFL.
841 .SFL
842 Close
843 All 9 windows are open.
844 Error - Printer file not found.
845 All 9 windows are open. Close a window and try again.
846 First select the text to cut, then try again.
847 Cut to Clipboard.
848 Appended to Clipboard.
849 First select the text to copy, then try again.
850 Copied to Clipboard.
851 Clipboard is empty. First use Cut or Copy.
852 First select the text to delete.
853 First select the text to move.
854 Application error .....
855 Please load a dialog box (DLG) file.
856 Change case on selection not available in redlining mode.
857 You cannot protect a selected row in a column table.
858 Working...Screen will blank momentarily.
859 Selected text is protected.
860 Cursor is not in a protected block.
861 Selected text is now editable.
862 Rest of file from cursor forward is protected.
863 Protected block is shown selected.
864 Application error .....
865 Type the text.

866 Not found.
867 Type the search text and replace text.
868 Bookmark is set.
869 Second bookmark is set.
870 First set the bookmark.
871 Application error .....
872 Default set to fast edit view (this session only).
873 Default set to graphic view (this session only).
874 Default set to page-line view (this session only).
875 Default set to expanded view (this session only).
876 Application error .....
877 Number must be at least 5.
878 Number must be no more than 400.
879 Default set to Show Page Breaks.
880 Default set to Hide Page Breaks.
881 Value should be between .1 and 250. Try again.
882 Application error .....
883 Block made non-breakable.
884 Selected text is already allowed to break.
885 Cursor is not inside a non-breakable block.
886 This block can now break across pages.
887 Copy existing style and modify it using menus.
888 Portrait
889 Landscape
890 Function requires an open file.
891 cpi
892 prop. width
893 normal
894 bold
895 italic
896 Please select a color from list and try again.
897 Please select colors from lists and try again.
898 Pick up format at cursor and modify it using menus.
899 Type the name of the style to apply.
900 This document contains no styles.
901 Edit the style definition.
902 Select a style first.
903 Style not found.
904 Please name a style.
905 Define style automatically using format at cursor.
906 Style definition inserted at top of file:
907 Styles for file
908 No more windows available.
909 Choose "Print" from the File menu to print this list.
910 Could not create new window.
911 Close this file when done.
912 Cannot save current settings permanently.
913 Saving settings...
914 Current page:
915 Envelope feed is located in center of tray.
916 Envelope feed is located at edge of tray.
917 No radio button was selected.
918 and
919 Cannot create snaking columns inside a column table.
920 Number of columns must be between 1 and 12.
921 Left margin is too small to fit left border.
922 Header
923 Create Header
924 Application error .....
925 Create Footer
926 Application error .....
927 Page
928 of
929 Not yet implemented.
930 To insert page number, choose "Insert Other" from Options.
931 Try inserting a footer again when document contains two pages.
932 Type text in this window for
933 Footer
934 Edit the running header by modifying the RH command.
935 Edit the running footer by modifying the RF command.
936 Left,Right,Inside,Outside
937 Application error .....
938 No alternatives, type in correction.
939 You need to load a personal dictionary first.
940 You cannot use "*" or "?" in a filename.
941 Error -- wrong file.
942 Changes not saved.
943 Spelling file reloaded:
944 Please choose a main dictionary.
945 The dictionary does not contain the extension DIC.
946 Please specify the filename for the dictionary.
947 You must close this file before batch spell-checking.

948 Choose file, get unknown words, and make corrections.
949 Apply corrections to the original file.
950 Put insertion point in column to move and press F9 (Esc to exit).
951 Put insertion point in row to move and press F9 (Esc to exit).
952 Put insertion point in column to move TO and press F9 (Esc to exit).
953 Put insertion point in row to move TO and press F9 (Esc to exit).
954 Appears only when word breaks at end of line.
955 Application error .....
956 Application error .....
957 Keyboard: Tilde key (E)
958 Keyboard: Ctrl-Shift-H
959 Keyboard: (Changed from standard key, see Defaults.)
960 For words that always require a hyphen
961 Keyboard: Hyphen key (located above "P")
962 For minus sign and proper nouns
963 Keyboard: Minus (on keypad)
964 Application error .....
965 Nota Bene
966 Number of words in entire document:
967 Number of words from cursor forward:
968 Number of words from start of file to cursor:
969 Number of words in selected block:
970 Redlining is ON.
971 Redlining is OFF.
972 Application error .....
973 Please choose two files.
974 Page break takes effect on next line.
975 Please check date and time and select a format.
976 Cannot choose "Final page number" with "Starting page no."
977 Cannot choose "Combination" with "Starting page no."
978 Special Characters
979 Latin/Germanic Based Accented Characters
980 Mathematical Symbols and Greek Characters
981 Lines, Corners, and Intersections
982 Punctuation and Accents
983 Standard
984 Cannot create a table inside a table.
985 Number of columns must be between 1 and 12.
986 Cursor is not inside a table.
987 Please select the text and try again.
988 Cannot convert if there are more than 12 tabs in a line.
989 Character added to User Set.
990 Character removed from User Set.
991 Character is already in User Set.
992 Type, copy, or move text into this window.
993 No borders defined.
994 Border not found:
995 This border name is reserved:
996 This border already exists. Try another name.
997 This border is reserved and cannot be applied:
998 Type the text for the footnote.
999 Edit the footnotes.
1000 Footnote Separator
1001 Footnote Wrap Separator
1002 No footnote format found before the cursor.
1003 Please specify the starting number and style.
1004 Enter the footnote separator.
1005 Enter footnote wrap separator.
1006 "No Footnotes" marker was inserted at top of document.
1007 Footnotes will be placed here.
1008 Marker inserted. Footnotes will be placed at this point.
1009 (continued)
1010 Press "Define" to append to a macro.
1011 No text selected. Select text and try again.
1012 Enter single-digit letter or number.
1013 Cannot add text to a program macro.
1014 Text added to key:
1015 Text saved to key:
1016 DICT.SPL is the main dictionary and cannot be edited.
1017 Application error .....
1018 Cannot edit. This key is empty.
1019 This key is already empty.
1020 Macro removed from key:
1021 Cannot print. This key is empty.
1022 No window available to set up printing.
1023 To print this macro, choose "Print" from the File menu.
1024 Macro set saved to file:
1025 Choose "Close" from File menu when done.
1026 Error loading file:

1027 Macro file loaded:
1028 Macros have been cleared from memory.
1029 [Empty]
1030 Program:
1031 [Program]
1032 Text:
1033 To print this list, choose "Print" from the File menu.
1034 Macros
1035 Application error .....
1036 Application error .....
1037 Outline Level
1038 You have not defined the outline format.
1039 I|A|1|a|1|a|i|
1040 1|a|i|1|a|i|1|
1041 ||||(|(||
1042 |||(|(|(||
1043 .|.|.|)|)|)|)|
1044 .|.|.|)|)|)|)|
1045 Outlining was not on.
1046 Original format restored.
1047 Enter new value...
1048 Marker inserted at cursor.
1049 Marker inserted at top of file.
1050 Sequence not found.
1051 Cursor is not on a footnote marker.
1052 Label added to footnote.
1053 Cursor is not on a counter marker.
1054 Label added to counter.
1055 Please select the text first.
1056 Selected text contains a style. Use F5 or select other text.
1057 Cannot find the marker for end of TOC.
1058 aardvark, 15|afghan hound, 18|ape, 27|baboon, 14|banana, 13
1059 Place TOC Marker
1060 Place Static TOC
1061 Automobile
1062 This is the TOC.
1063 Generating TOC for marker #
1064 Unexpected error in generating the TOC.
1065 This is index format.
1066 Must fill "Sub level1" if "Sub level2" is filled.
1067 Searching for an existing index...
1068 Cannot find the marker for end of index.
1069 Searching for an index format...
1070 Place Index Marker
1071 Place Static Index
1072 This is the index.
1073 Generating index for marker #
1074 Unexpected error in generating the index.
1075 The symbol ""#"" represents the letter separator.
1076 MEMOPAD
1077 Please type in a file name.
1078 Invalid file name.
1079 Try another file name.
1080 Type your comments. Then press "Ctrl+Shift+M" to store.
1081 ALL
1082 First select the lines to sort.
1083 This is the sorted file.
1084 (No message)
1085 No more stop codes.
1086 Application error .....
1087 Application error .....
1088 Could not get text from document.
1089 This document is built from:
1090 Error generating the file.
1091 Enter field names in the same order as in the Data File:
1092 The following field names already exist in Field ID:
1093 Field ID modified at top of file.
1094 Field ID inserted at top of file.
1095 Application error .....
1096 Cannot open data file. Close a window and try again.
1097 Error in reading the data file.
1098 Error - record separator is contained in field separator.
1099 There is no record separator in the data file.
1100 Application error .....
1101 Application error .....
1102 First select conditional text in Main file.
1103 Put text if:
1104 Put Text Conditionally
1105 Select records where:
1106 Select Records Conditionally
1107 Extract records where:
1108 Extract Records Conditionally
1109 Include text if:
1110 Include Text Conditionally
1111 Main|Cycle
1112 Select a field and relation from the lists, then try again.
1113 Value is ignored with the selected relation.
1114 Value is not a number. "Numeric" check box is ignored.
1115 Neither radio button was selected.
1116 A field is missing from the Field ID command.
1117 This is preview of first record.

1118 This is for preview only, not for editing.
1119 Application error .....
1120 All records
1121 .FRM
1122 Cannot create untitled form in different path.
1123 Error saving keystrokes.
1124 Error saving file to key.
1125 File saved to key:
1126 Keystrokes saved to key:
1127 Keystrokes saved to file:
1128 Error saving file.
1129 Macro key saved to program file:
1130 Application error .....
1131 File already exists:
1132 Error in creating file:
1133 Please select a category from the list and try again.
1134 Please select an entry from the list and try again.
1135 Cannot make permanent changes. File not found:
1136 Settings not recognized:
1137 Access denied. Cannot make permanent changes.
1138 New file SETTINGS.TMP could not be loaded from:
1139 SETTINGS.TMP
1140 Cannot use same separator for records and fields.
1141 Main dictionary path
1142 No printer file is loaded.
1143 Printer file does not exist.
1144 Printer file is open.
1145 Error in opening printer file.
1146 Error in writing to printer file:
1147 Printer file does not exist:
1148 Please specify the autosave intervals.
1149 Application error .....
1150 Error in loading the color set file.
1151 Error in saving the current set.
1152 In expanded view, markers always show.
1153 Cannot open Macro Set file -- it is binary, not ASCII.
1154 Cannot open file.
1155 Please specify the backup filename.
1156 Application error .....
1157 Cannot find tutorial. Refer to installation procedure.
1158 Tutorial directory not found:
1159 TBOOK.EXE SWTUTOR.TBK
1160 ALTMENU
1161 Cannot find ALTMENU.PGM or ALTMENU.MNU
1162 .INT
1163 Lost track of menu file to return to.
1164 Application error .....
1165 Loading
1166 Select fewer characters (only 75 allowed).
1167 |Search which drive?
1168 List all files on drive
1169 No keyboard table in this file.
1170 Application error .....
1171 Press any key for key code.
1172 not found.
1173 Keycode
1174 No equal sign (=) found.
1175 Wrong format for keycode.
1176 F
1177 No SHIFT table found.
1178 is key
1179 is not a letter or number key.
1180 .TMP
1181 .TPL
1182 .PRN
1183 .DSP
1184 .DFL
1185 .KBD
1186 .HLP
1187 .SPL
1188 .HYP
1189 Changes to this file have not been saved
1190 Changes to this file have been saved
1191 LOAD.TMP
1192 .MNU
1193 TEMP.MNU
1194 Saving to
1195 File must be open.
1196 PRINT.TMP
1197 |Print the selected block in expanded view? (Y/N)
1198 |Print the entire file in expanded view? (Y/N)
1199 Close all open files and try again.
1200 RESUME.PGM
1201 RESUME.SGT
1202 Cannot find file:
1203 Press F9 to execute.
1204 Error.
1205 This key is not currently assigned.
1206 Load either the mini keyboard or the main menus.
1207 A menu subroutine is missing. (Menu: $...Subroutine: $)

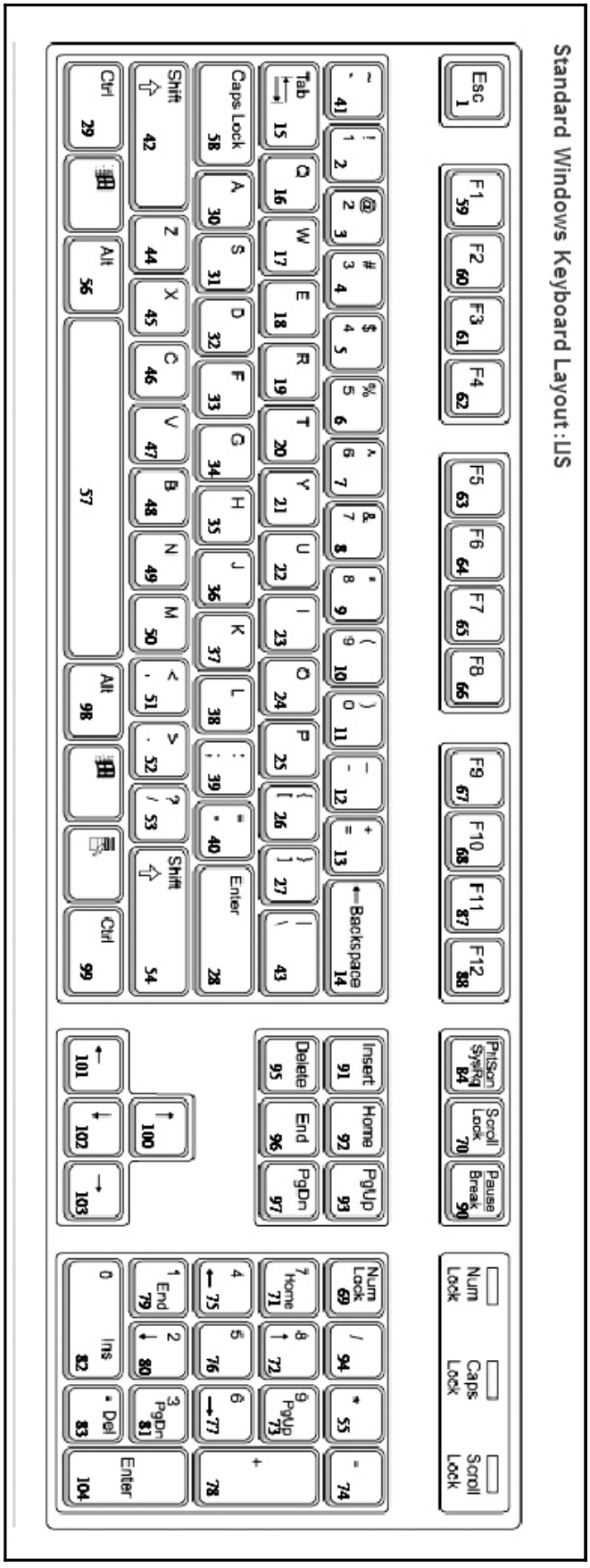
1208 Application error .....
1209 Choose one to RUN:
1210 Choose one to LOAD:
1211 Page Length
1212 Page Width
1213 Type
1214 Other
1215 |Delete marker for old header? (Y/N)
1216 Immediately
1217 At print time
1218 (Chapter number not available in header)
1219 Columns,Tables,Frames,Pages
1220 Create,Edit
1221 Height,Width
1222 F-WINDOWS
1223 only if Nota Bene cannot import source format directly.
1224 regardless of source format.
1225 Cannot draw with proportional font.
1226 RESUME.DEL
1227
TEXTIN.FLT,TEXTOUT.FLT,SPREAD.FLT,DATABASE.FLT,GRAPHIC.FLT
1228 Application error .....
1229 Print queue is full--please wait.
1230 PLFITILN
1231 Invalid format bar item &
1232 Spooler General Error
1233 Spooler, Printing canceled from program
1234 Spooler, Printing canceled from Print Manager
1235 Spooler, Out of disk space
1236 Spooler, Out of memory space
1237 Application error .....
1238 Printer setup error.
1239 XWRIM.DLL
1240 XWREX.DLL
1241 Unable to load RFT filter.
1242 .SG1
1243 .TP1
1244 .SP1
1245 .TM1
1246 .FM1
1247 SWBMP.DLL
1248 SWHELP.HLP
1249 CMHelp
1250 CR
1251 Error loading Windows printer fonts.
1252 Function not available using Windows fonts.
1253 Printer port not specified.
1254 TFGF
1255 Cannot find DLL.
1256 Cannot rename -- target file already exists.
1257 Error Number &
1258 DDE transaction timed out.
1259 DDE is out of memory.
1260 No DDE conversations established.
1261 DDE conversations established.
1262 DDE server died.
1263 DDE error.
1264 DDE server is busy.
1265 Cannot change number of columns in table edit.
1266 Column numbers are out of sequence.
1267 Invalid DDE conversation number.
1268 Unable to transfer DDE data.
1269 Left margin/offset on this printer must be at least &
1270 Bottom margin must be at least &
1271 0 (No font family)
1272 1 (Serif)
1273 2 (Sans Serif)
1274 3 (Monospace)
1275 4 (Script)
1276 5 (Decorative)
1277 DoCommand DoFunc PutString PutChar
1278 Invalid DDE item.
1279 Invalid DDE Execute subcommand.
1280 Invalid DDE topic.
1281 No DDE help available. (Need .DLG file with /E frame.)
1282 Requires 8 character string.
1283 Label
1284 Internal ATS error, save files and exit NOW.
1285 LPT
1286 Function not available under Windows.
1287 Too many color changes in one line.
1288 Change failed.
1289 [UNLABELED]
1290 Bad printer driver DLL.
1291 device
1292 No Nota Bene printer file loaded. Printing disabled.
1293 settings
1294 No screen fonts. Cannot go into graphics view.
1295 Cannot use mixed font mode. No Speedo fonts.
1296 Windows driver problem. Use Nota Bene driver.
1297 Using Windows device: &
1298 Using Nota Bene driver: &

1299 No Nota Bene driver loaded. SPEEDO fonts used.
1300 No printer file loaded. Cannot go into graphics view.
1301 When using Windows device drivers, SETP not allowed.
1302 No printer device with this number.
1303 Directory list
1304 AaBbCcXxYyZz
1305 Command not allowed when using Nota Bene driver.
1306 Job
1307 When using Windows device drivers, no reverse collation printing.
1308 Cannot open Untitled file.
1309 Two or more snaking columns occupy same position on page.
1310 Cannot minimize command window.
1311 Cannot print BMP graphic file with Non-Windows driver. &
1312 Bad printer font type.
1313 CWSPXR
1314 No width table for this font definition.
1315 Language Code not supported.
1316 Bad PR Keyword in printer file.
1317 zero
1318 one
1319 two
1320 three
1321 four
1322 five
1323 six
1324 seven
1325 eight
1326 nine
1327 ten
1328 eleven
1329 twelve
1330 thirteen
1331 fourteen
1332 fifteen
1333 sixteen
1334 seventeen
1335 eighteen
1336 nineteen
1337 twenty
1338 thirty
1339 forty
1340 fifty
1341 sixty
1342 seventy
1343 eighty
1344 ninety
1345 hundred
1346 thousand
1347 million
1348 billion
1349 dollars
1350 cents
1351 Indentation exceeds right margin.
1352 Article
1353 adj
1354 adverb
1355 adj & adv
1356 noun
1357 verb
1358 pronoun
1359 prep
1360 numeral
1361 exclamation
1362 interject
1363 conj
1364 SMI not enabled in WIN.INI - Email functions not available.
1365 Inches
1366 Centimeters
1367 Picas
1368 Points
1369 Ciceros
1370 Paragraph
1371 Document End
1372 Next Command
1373 Current Selection
1374 Ovr
1375 Ins
1376 Num
1377 Cap
1378 Bad Rule Syntax --&
1379 MISSING IF &
1380 MISSING LEFT (&
1381 MISSING RIGHT) &
1382 SYNTAX OR UNRECOGNIZED SYMBOL &
1383 DOMAIN ERROR IN FUNCTION &
1384 DIVIDE BY ZERO &
1385 MISSING QUOTE &
1386 MIXED TYPES &
1387 Please select an item.
1388 Must specify database file using the GF default
1389 Can't find Entry Point for EB script %s, error= %d
1390 Can't create Window for EB script %s

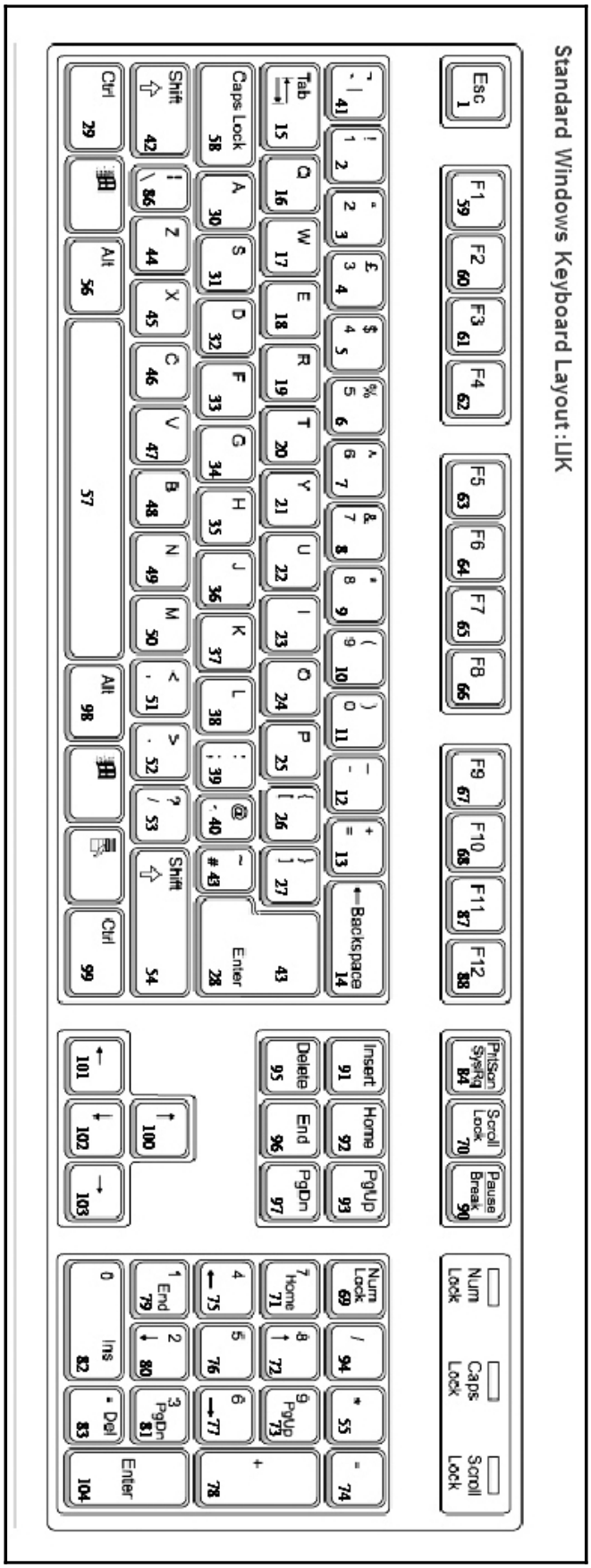
1391 EB Script %s Compile error %d at line %d,character pos %d
1392 No code to ECompile error on EB script %s
1393 Cannot create script %s, error= %d
1394 Cannot create thread for script %s, error= %d
1395 EB Script %s runtime error, code= %d, line= %d
1396 Error registering private application extensions.
1397 EB E{µE¡¦EÓ¤Ej¤, ELµEkªEõ°Eæ¦
1398 Eò¿E¦º BEGINEBX E³E¡¦
1399 Eò¿E¦º ENDEBX E³E¡¦
1400 El·E´·Eºª DLG EÎ© EBX EÉÀE×
1401 Nota Bene EBX Eï¹EÜ¸E°²Es½Eè¿Eû
1402 |Enter an arithmetic operator.\n +,-,*,/
1403 |Press any key to continue Debugging.\n*
1404 |Press any key to continue XPL.\n*
1405 |Press '0' to move down, '1' to move up, '2' move to top, '3' move to bottom.\n 0,1,2,3
1406 |Press '0' to insert IN FRONT, '1' to insert BELOW, '2-5' refer Doc\n 0,1,2,3,4,5
1407 |Enter '0' to turn off dispaly of graphics, '1' to turn it on. \n 0,1
1408 |Enter '0-7' for language selection.\n 0,1,2,3,4,5,6,7
1409 |Enter '0-9' for AutoSpell,AutoReplace or AutoTranslate argument.\n 0,1,2,3,4,5,6,7,8,9
1410 |Enter '0-9' for Bookmark number OR 'G' to Goto Bookmark.\n G,0,1,2,3,4,5,6,7,8,9
1411 |Enter '0-9' for BookMark number.\n 0,1,2,3,4,5,6,7,8,9
1412 |Enter '0-9' for Journal Restore level.\n 0,1,2,3,4,5,6,7,8,9
1413 |Enter '0-9' for Outline level.\n 0,1,2,3,4,5,6,7,8,9
1414 |Enter '0-9' or 'A-Z' for Window ID.\n *
1415 |Enter '0-9', Hyphen, 'O' ... for wild card specifier.\n O,0,1,2,3,4,5,6,7,8,9
1416 |Enter '1-6', transpose function type.\n 1,2,3,4,5,6
1417 |Enter a character for the XPL or function argument.\n *
1418 |Enter a character in the password.\n *
1419 |Enter a character for the Macro id.\n *
1420 |Enter '0' to response must use keyboard, '1' allow response from XPL.\n 0,1
1421 variables updated.
1422 999/999-99.99IN
1423 12:59PM
1424 Save Changes?
1425 |Enter '0' to reset elapsed time, '1' to initialize clock to timing mode.\n 0,1
1426 |Do you want to delete rule? (Y/N)
1427 Unrecognized outline option.
1428 Outline command skipped a level.
1429 Outline command missing level number or icon arguments or level too big.
1430 Please specify an object id.
1431 was not found.
1432 No objects to update.
1433 |Versions don't match, update anyway? (Y/N)
1434 was too short.
1435 Error updating library, update aborted.
1436 Paragraph was not marked as modified, update anyway? (Y/N)
1437 Label not found.
1438 Specify LG,GC,OB,OL,CR,CM,TO.
1439 RP:[Begin new session]:CL:Comment:CC:UC:LC:Insert:Delete:Group:
1440 Can't edit more than one command at a time.
1441 P T S IDH W D STPGF
1442 Get newest version? (Y/N)
1443 No miscellaneous subject specified--proceeding anyway.
1444 Object Explanation:
1445 View:
1446 Note:
1447 Comment:
1448 Caution:
1449 Stop:
1450 Drafting Tip:
1451 Explanation:
1452 Rule:
1453 Alternate Rule:
1454 Alternate Rule:
1455 Object:
1456 Variable:
1457 Variable Explanation:
1458 Q & A:
1459 Rule: (TRUE)
1460 Rule: (FALSE)
1461 Reserved:
1462 Reserved:
1463 Reserved:
1464 Processing Embedded Commands.
1465 Composing Pages.
1466 Refreshing Variables.
1467 GTSGT needs a save get id.

1468 |Enter '0 - 9' for VB keycode. \n 0,1,2,3,4,5,6,7,8,9
1469 Specify PO, NO, SF, EF (prev/next outline, start/end file).
1470 Specify 0M to move up, 1M to move down, 0C, 1C for copy.
1471 Accessing Database.
1472 Starting variable refresh.
1473 Specify command to put at top of file.
1474 BLDSEQ requires a sequence number and a create/replace flag.
1475 JRNGRP requires number greater than 0.
1476 SAVCLN requires an output file name.
1477 SAVCLN--could not open output file.
1478 SAVCLN--error writing output file.
1479 OLN--too many alternates.
1480 Unrecognized extended command.
1481 Unrecognized LOOP (LP) type.
1482 No QRY= keyword in LP command.
1483 Too many nested loops.
1484 Cursor is not in the range of a loop.
1485 |Selection contains unbalanced rule; extend selection to balance? (Y/N)
1486 |Selection contains a partial outline level; extend to full level? (Y/N)
(If you answer No and then delete, the document structure may be corrupted.)
1487 Component appears twice in object:
1488 Object missing close OB:
1489 Close OB without Open:
1490 Rule imbalance in object:
1491 Component(s) missing or out of order:
1492 Selection contains a partial framework component; extend to entire? (Y/N)
(If you answer No and then delete, the document structure may be corrupted.)
1493 Change will not take effect until next page.
1494 This is an authored template, and only an author may edit it. You may copy it and modify your copy.
1495 «OBO/LF»«USLEVEL@OL»|«OBO/CO»«USCOUNTER@OL»|«OBO/DE»|«OBO/TF»«USTITLE@OL»|«OBO/TI»|«OBO/TE»«USTITLEEND@OL»|«OBO/BF»«USBODYTEXT@OL»|«OBO/BT»|
1496 Too many different TOL items. Reuse an existing name
1497 Need: TOL digit,name,mnemonic1,...,mnemonicn.
1498 Not within a help link.
1499 User security level is less than the object security level.
1500 The editing position is not in an object.
1501 CHGAL requires a value between 0 and 9.
1502 The editing position is not in an valid object.
1503 Application error .....
1504 Application error .....
1505 Too many suppression strings.
1506 |Enter 4 hex digits
1507 Invalid keystroke
1508 Too many mode commands on one line
1509 changes...
1510 ÛÛÛÛ

US Keyboard



UK Keyboard



Index

❖ A ❖

❖ B ❖

❖ C ❖

❖ D ❖

❖ E ❖

❖ F ❖

❖ G ❖

❖ H ❖

❖ I ❖

❖ J ❖

❖ K ❖

❖ L ❖

❖ M ❖

❖ N ❖

❖ O ❖

❖ P ❖

❖ R ❖

❖ S ❖

❖ T ❖

❖ U ❖

❖ V ❖

❖ W ❖

❖ X ❖

❖ Z ❖