XyWrite
Programming Language User's Guide

XYQUEST

[Signature Programming Language User's Guide]

First Edition, January 1992
Publication Number 002409-01
ISBN 0-927923-24-6
Printed in the U.S.A.

XYQUEST Inc.
44 Manning Road
Billerica, MA 01821
(508) 671-0888

Written by Rose Intingaro with contributions from Tom Atwood, Tim Baehr, Jason Balliet, Lucas Gonze, Ruth Gray, Susan Koenig, Doug Kramer, Chris Madsen, and Claire Toupin; edited and additional material
R. J. Holmgren January 2006

http://www.serve.com/xywwweb/XPL.ZIP

Editor's Preface to *XyWrite Programming Language User's Guide*

XPL.ZIP contains a 156-page XyWrite Programming Language (XPL) user manual written by Rose Intingaro, with additional contributions by many XyQuestrians. Originally published in 1992 as the *Signature Programming Language User's Guide*, it was not bundled with the primary Signature package and therefore received very limited distribution. This is the only guide to XyWrite *IV-level* XPL in existence, and it represents the most ambitious, comprehensive programming manual that XyQuest produced for any version of XyWrite. Because Nota Bene seems unlikely or unwilling to publish anything on this scale (or, indeed, to produce any written manuals whatsoever), the present volume represents a unique asset, superseding Tony Woozley's sorely outdated *Customization and Programming Guide*, which was based on Nota Bene v4 [=XyWrite III] (XyWrite IV and NBWin are incomparably more powerful than XyWrite III). This guide is targeted at novice programmers, but it approaches XPL with a thoroughness that yields many insights to the experienced coder as well. Download **XPL.ZIP**: http://www.serve.com/xywwweb/XPL.ZIP

Signature, financed largely by IBM as the intended replacement for their proprietary *DisplayWrite* word processor, was the initial incarnation of what subsequently evolved into *XyWrite IV for DOS*, *XyWrite for Windows*, and *Nota Bene for Windows*. The kernels of all of these products were (and still are) designed and coded by one person, David Erickson, which gives them an extraordinary degree of internal consistency and continuity. Signature was flawed, as first versions usually are; but ***all*** of the bedrock principles of "Xy4-level XPL" established by Signature — of XPL as we know it today — persist through the entire lineage, to the latest Nota Bene for Windows (NBWin). 99% of XPL written for Signature or Xy4-DOS runs perfectly, without alteration, in NBWin v8; and conversely almost nothing of importance has been added to the language since 1992. Very few Notabenieri seem aware of the extent of NB's Xy4 inheritance, or of the near-identity of the underlying word processor engines. With an understanding of the differences imposed by the Windows environment and of the few (minor) enhancements|alterations to XPL ordained by NBWin, 100% compatibility and interoperability between Xy4 and NBWin can be realized.

The present text of the manual was derived from an optical character reader, and carefully proofed. Visual conventions of the original edition have been retained in large part (see Preface-2). The eight sample XPL programs that accompanied the original book on diskette are included, in **XPL.ZIP** subdirectory **PROGRAMS**. The ZIP also contains a keyboard file, **XY4.KBD**, which is compatible with the specific keystroke instructions in the text; **XY4.KBD** may be loaded under NBWin, although the keystrokes to which NBWin users are accustomed (using NB.KBD) will **not** be available (in particular, F5 clears the command line [function BC], and F9 executes commands [function XC]). It is not necessary to load this keyboard file to understand the manual.

I have taken a few liberties with the original text:

- references to Signature have been replaced by "XyWrite"
  (N.B.: References to "XyWrite" apply equally, with only cosmetic differences, to Nota Bene for Windows)
- references to SPL have been replaced by "XPL"
- a description of the character 240 "Contains" operator is added
- a description of the Count Up command is added
- the tables of Functions and Variables (Appendices B and C) have been updated (although not all new NBWin variables are tabled)
- miscellaneous corrections and additions

Two versions of the manual are provided in this ZIPfile: **XPL.TXT**, which is formatted for XyWrite and NBWin, and which contains executable code ("live" guillemets and functions); and **XPL.PDF**, a searchable, text-based version of the manual in Portable Document Format. *Adobe Acrobat* (or *Acrobat Reader*) v7.0.5+, or *Ghostscript* v8.53+ with *GSview* v4.7+, are recommended PDF viewers.

**** XPL.TXT is intended to be displayed in Expanded (Codes) View! ****

An included program, **CONVERT.PM**, transforms the "live" text and programming in **XPL.TXT** into a passive form, suitable for Draft or Graphics Views, or for printing to paper. Command **RUN CONVERT.PM**, with **XPL.TXT** in the current window, under either NBWin or XyWrite 4.

Robert Holmgren
January 2006

ORIGINAL PREFACE

XyWrite Programming Language (XPL) User's Guide provides an in-depth explanation of how to use XPL with XyWrite to produce effective programs for a variety of helpful applications. A special disk, XPL Program Examples, is also provided to demonstrate various concepts and is referenced throughout the document.

This document assumes you know some basic programming concepts. (If you are familiar with the BASIC programming language, refer to Appendix D for a comparison of XPL and BASIC.)

XyWrite Programming Language (XPL) User's Guide is organized into five sections and four appendixes:

- *Introduction* gives you an idea of the things you can do with XPL and provides step-by-step instructions for creating a simple program.

- *Part 1, Tools of the Trade*, describes principles you will use again and again in programming. Such topics as function calls, macros, flow control commands, and commands that return values are explored in depth. One or more examples and a complete explanation are provided for each topic.

- *Part 2, Common Applications*, combines many of the concepts introduced in Part 1 to demonstrate three commonly used XPL programming techniques — using IF statements, monitoring keyboard input, and displaying a selective directory.

- *Part 3, Program Examples*, contains eight complete XPL programs to illustrate various XPL concepts.

- *Part 4, Tips*, has fourteen useful hints to help you write effective, efficient XPL programs.

- *Appendix A* contains a chart of all the XPL commands and operators for quick reference.

- *Appendix B* has a complete list of XyWrite function calls.

- *Appendix C* lists all the variable mnemonics you can use with the VA command.

- *Appendix D* lists some major features of the BASIC programming language and describes their XPL equivalents. It will help someone who knows the BASIC programming language pick up XyWrite Programming Language (XPL) quickly and easily.

The following conventions are used throughout this document:

• All examples of program code in XPL.PDF are shown in Expanded ("Codes") View. The companion XyWrite file XPL.TXT (also in XPL.ZIP) contains executable code, designed to be displayed in Expanded ("Codes") View. XPL.TXT may be converted to a file (XPLPRINT.TXT) that can be viewed in Draft or Graphics View under either XyWrite IV or Nota Bene for Windows, and/or printed to paper, by running CONVERT.PM (included in XPL.ZIP).

• XyWrite function calls are represented in an UPPERCASE `fixed-space` **`bold`** `font`; for example, **`BC`** , **`XC`** , **`BX`** , or **`Q2`** .

• XyWrite commands are represented in a lowercase `fixed-space font`; for example, `call`, `search`, or `store`.

• XyWrite keystrokes are represented as symbols, for example, Ctrl, Shift, Caps, B, or 1. These reference the standard keystrokes assigned in file XY4.KBD. XY4.KBD is included in XPL.ZIP, for the convenience of Nota Bene for Windows users (it can be loaded in Nota Bene without error — command LDKBD XY4.KBD).

• The glossary on the next two pages defines some common programming terms as they are used in this document.

GLOSSARY

Argument - one or more variable values you can give to a command to specify characteristics of how you want that command to function. For example, the syntax for the RUN command is: `run` *programfile,string*. Both *programfile* and *string* are arguments of the RUN command that let you specify the name of the program file you want to run and a string of characters you want to pass into the program.

Branch - an instruction that causes the program to change direction and continue executing in another part of the program. In XPL, you use the GL (Go to Label) command to branch to another point in the program labeled with the LB (Label) command.

Concatenation - joining two or more character strings together.

Expression - a set of characters that can be evaluated. There are three types of expressions: logical, numeric, and string. *Logical expressions* are either true or false. In XPL, logical expressions are used in IF statements and often contain relational operators or the Error command. *Numeric expressions* usually contain some arithmetic operators. For example, in XPL the SX (Save Expression) command saves the result of a numeric expression or a value to a macro (`«SX02,(25+25)»`). *String expressions* contain string operators (for example, `+` for concatenation, `@UPR` for uppercase, `ε [238]` for "is contained in", `≡ [240]` for "contains", etc.).

Jump - another term for branching or moving to a different point in the program.

Macro - in its simplest form, a place in your computer's memory where XyWrite can store information. A macro can contain text, XyWrite commands, function calls, other macros, and even entire programs. You assign numbers or letters to these pieces of information so that XyWrite can keep track of them. For example, to store the phrase, "Today is Monday", in macro 01, you would include the following in your program file: `«SV01,Today is Monday»`.

Main program - a relative term used to distinguish between a program and its subroutines.

Program - a XyWrite file containing XyWrite function calls, commands, macros, or text. When it is executed, a program performs commands and inserts text as if you had typed them from the keyboard.

String - any character or series of characters that XyWrite saves exactly as typed, without any numeric interpretation or evaluation. Strings are also referred to as character strings or literals.

Subroutine - a set of instructions that are executed when the main program calls for it. In XPL, you use the SU (Save Subroutine) command to save program instructions to a macro that is executed by the main program.

INTRODUCTION

XPL is a high-level programming language developed by XYQUEST that combines the functions of a word processor with traditional programming tools. It is a specialized language that helps you create programs to emulate the way you interact with the computer as a user.

In its simplest form, XPL can be a series of keystrokes. You can simply store those keystrokes and reuse them repeatedly, or you can combine them with programming instructions to produce a variety of customized applications. For example, you can:

- Save values during program execution
- Insert values into the program from the keyboard, from XyWrite, or from another XPL program
- Perform arithmetic or logical (boolean) operations

Like all programming languages, XPL is sequential — program execution begins with the very first character of the program file and continues, character by character, or command by command. However, there are several things you can do to break out of the sequence, such as:

- Label a point in the program and cause execution to jump to that location from anywhere in the program
- Evaluate a statement to determine if it is true or false and control execution based on the outcome of the evaluation
- Designate a secondary set of instructions or a subroutine that executes when the main program calls for it
- Exit the subroutine or the program

Once you have created your program, you run it to automatically execute the commands (and text) as if you had typed them from the keyboard. There are several ways to do this:

- Execute the RUN command from the keyboard
- Assign a simple program to a key in the keyboard file
- Assign the program to a macro as a subroutine

The rest of this Introduction provides an example of a simple programming session. It describes all the steps you take to create a program: from planning through debugging. An optional step, assigning the finished program to a key, is also explained.

STEP 1: PLANNING THE PROGRAM

Since programming is simply preparing a list of instructions that tells your computer how to perform a specific task, you should first think through the steps you would take if you manually performed the task. Then, analyze the steps to determine what the program has to do to perform the task and in what sequence.

NOTE: For more complex tasks, you may find it helpful to sketch a flowchart while planning your program. (Refer to Part 3 for examples of flowcharts.)

For example, if you habitually hold down the Shift key a split second too long, you frequently end up with two letters capitalized at the beginning of a word instead of one. Assuming you typed the entire word and have not typed the next word, you would manually correct this mistake as follows:

1. Back up to the previous word.

2. Make the second letter lowercase by going to the command line and executing the LC (Lowercase) command.

3. Return to where you left off.

As an example of how to use XPL, we will show you how to write a simple program to fix this problem automatically. The following four sections describe creating, writing, saving, and testing a program named TWOCAP.PGM.

STEP 2: CREATING A PROGRAM FILE

You use the NE (New) command to create a new program file. For example:

Type: F5 `ne twocap.pgm` Enter

As with any other text file, NE creates a new program file in memory; therefore, you must use the SAVE or the STORE command to save it on disk as you do with any other file.

After you have stored your program file, you use the CA (Call) command when you want to display it again:

Type: F5 `ca twocap.pgm` Enter

STEP 3: WRITING THE PROGRAM

You use the Scroll Lock key to turn program mode on and off. When program mode is on, an "S" appears at the top right of the screen; and XyWrite records all your keystrokes as function calls in your program file.

You use the PFUNC command to enter function calls not assigned to a key. Before you do this, be sure program mode is off.

For example, use the following keystrokes to enter code for the TWOCAP.PGM program. (If you make a mistake when program mode is on, press Scroll Lock to turn program mode off, correct the mistake as you would any typing mistake, and press Scroll Lock again to continue recording keystrokes.)

1. Type: F5 `pfunc gt` Enter

2. Press: Scroll Lock (to turn program mode on)

3. Press: Ctrl ←

4. Press: →

5. Type: F5 `lc` F9

6. Press: F5

7. Press: Scroll Lock (to turn program mode off)

8. Type: F5 `pfunc gt` Enter

9. Press: Scroll Lock (to turn program mode on)

10. Press: Ctrl →

11. Press: Scroll Lock (to turn program mode off)

Result: Your program file now contains code that looks like this:

GT PW CR BC `lc`**XC BC GT NW**

NOTE: Depending on the keyboard file you have loaded, XyWrite may output the NI (Not IBM-Sensitive Key) function call with one or more key codes. This will not affect the way your program executes.

The program code functions as follows:

GT (Go to Text Area) moves the cursor into the text area.

PW (Previous Word) moves the cursor to the first character of the previous word.

CR (Cursor Right) moves the cursor one space to the right, which places it on the second character of the previous word.

BC lc**XC** blanks the command line (**BC**), enters the LC (Lowercase) command, and executes it (**XC**).

BC (Blank the Command Line) clears the command line.

GT (Go to Text Area) moves the cursor into the text area.

NW (Next Word) moves the cursor to the next word, which would be the point where you stopped to correct your mistake.

STEP 4: STORING THE PROGRAM

Remember, until you use the STORE (or the SAVE) command, your program file is stored in memory only, not on disk. You cannot run your program until it is stored on disk.

To store the displayed program file TWOCAP.PGM:

Type: F5 store Enter

Result: TWOCAP.PGM is stored on the disk and is cleared from the screen.

Now you are ready to test and debug the program.

STEP 5: TESTING AND DEBUGGING THE PROGRAM

Testing and debugging consist of locating and correcting errors in your program. Errors can cause XyWrite to display an error message or the program to produce either no results or incorrect results.

Debugging is the process of running all (or a portion) of your program and observing the results. ("Debugging Procedures", in Part 1, describes more complex debugging methods.)

If XyWrite displays an error when you try to run your program, often it is due to an error in syntax — you used one or more XPL commands improperly. See Part 4, *Tips*, for an explanation of four common errors.

If your program did not produce any results or produced incorrect results, it may be due to an error in logic. Review the procedure you used in planning your program.

To debug TWOCAP.PGM:

1. Create a test file. For example:

Type: F5`ne test`Enter

2. Type a word that begins with two capital letters instead of one. For example:

Type: `SIgnature`

(In order for the program to work properly, the cursor must be on the space or separator following the mistyped word.)

3. Run TWOCAP.PGM

Type: F5`run twocap.pgm`Enter

Result: TWOCAP.PGM changes the case of the I in SIgnature and positions the cursor for you to type the next word. If it did not, review the procedures in Steps 1 through 3.

The next step is optional, but it makes using your program easier.

STEP 6: LOADING THE PROGRAM TO A KEY (OPTIONAL)

Once you have written and debugged your XPL program, you can make it even easier to use by loading it directly to a macro key. You can then access it with one keystroke.

You use the LDPM (Load Program) command to load your program onto a macro key. You access programs you have saved to ordinary macros by pressing F2 and the key.

To load TWOCAP.PGM onto macro key T:

Type: F5 `ldpm twocap.pgm,t` Enter

Result: The program file is copied to the macro key in memory. You can now run TWOCAP.PGM by pressing F2 T.

NOTE: If you load your program to a macro key, you must reload it whenever you make any changes to the program. Therefore, it is a good idea to completely debug your program and make sure it works the way you want it to *before* loading it to a key.

If you want to keep the program file loaded on the macro key for use at future editing sessions (after you QUIT), use the STSGT (Store Macro Keys) command. The syntax for the STSGT command is:

`stsgt` *filename*

filename identifies the file on disk where you want the macro keys stored.

For example, the following saves the set of current macro keys to a file named PROPOSAL.SGT:

Type: F5 `stsgt proposal.sgt` Enter

You can use any filename you want, but we recommend that you use the SGT extension, and store the file in the same directory as EDITOR.EXE. If you follow these conventions, XyWrite will include the new macro file when it creates a list of macros.

NOTE: To print your program, call it to the screen and press Ctrl Alt Shift P. As part of the function, XyWrite copies the file to PRINT.TMP. Therefore, you can edit the version in PRINT.TMP and reprint it if necessary; however, you cannot run the PRINT.TMP version. (The PRINT.TMP file is overwritten by the next program you print and is erased when you QUIT XyWrite unless you save it to a file with a different name.)

Now that you have reviewed the basics of creating a very simple program, read Part 1, *Tools of the Trade*, to learn how you can add more power and functionality to your programs.

For example, XPL enables you to:

- Save values to 2000 macros (000-1999) during program execution
- Save strings, subroutines, or expressions in a macro
- Use the contents of a macro in an expression
- Divide the contents of a macro into parts, extract the parts, and then save them to other macros
- Insert the contents of a macro key at the cursor position
- Clear macros from memory
- Branch on a condition (IF Statement)
- Test for errors
- Stop to wait for keyboard input any time during program execution
- Pass values to a program as it starts
- Determine the current cursor position
- Make use of the current filename, path, page number, line number, and other XyWrite values and settings
- Label a point in the program
- Jump to a label
- Switch to the window that contains a specific file
- Use operators and functions to perform various tasks in your program
- Produce an audible signal at a point in your program
- Use the XPL debugger when testing your program to pause execution at certain points
- Execute a command from within a program without clearing the command line
- Display a selective directory

PART 1 • TOOLS OF THE TRADE

Part 1 describes the various tools that you can use when writing your XPL. programs. Including function calls in your program is discussed followed by information on macros, flow control commands, and commands that return values. Relational, arithmetic, logical, and string operators are also described. The last section discusses procedures you can use to debug your programs.

Many of the commands described in Part 1 are embedded commands, and you enter them in your program as you would any other embedded XyWrite commands. When you reach a point in the program where you want to insert an embedded command, turn off program mode by pressing Scroll Lock. After you press F9 to enter certain of these embedded commands in your program, XyWrite opens a command window so you can include an argument for the command. Press either F11 or Alt F1 to close the window when you have entered the appropriate argument. (To abort the command window, press Esc.)

Appendix A contains a quick reference chart of all the commands and operators you can use in XPL.

(XYQUEST plans to publish a book that includes all the XyWrite commands. For more information, contact XYQUEST.)

FUNCTION CALLS

Function calls are two-letter instructions that represent basic XyWrite activities. Some function calls are assigned to keys and some are not.

In a program, XyWrite function calls look like two bold letters followed by a space; however, they are unique, single-unit representations that must be created either by using program mode or by using the PFUNC (Put Function Call) command. Although function calls appear to have a space after them, the space is a part of the function call itself. If you try to delete it, you will delete the entire function call.

After you press Scroll Lock to turn program mode on, every key you press that is not an alphanumeric character causes XyWrite to record the function call assigned to that key into your program file. For example, if you press F5, XyWrite enters the function call **BC** (Blank the Command Line) into your program. If you press →, XyWrite enters the function call **CR** (Cursor Right). If you press Ctrl Shift 9, XyWrite enters the function call **#9** . (If you have loaded a customized keyboard file, XyWrite enters the function calls you associated with those keys.)

When you are not in program mode, you can use the PFUNC command to enter a XyWrite function call into your program. The PFUNC command gives you a lot of flexibility in creating your programs, because it lets you enter function calls not mapped to a key. The syntax for the PFUNC command is:

```
pfunc ##
```

is a two-letter function call.

For example, if you want to make certain the cursor is in insert mode, you can enter the function call **SI** (Set Insert Mode) into your program as follows:

Type: F5`pfunc si`Enter

Result: When you run the program, insert mode is activated at the point where the **SI** function appears in your program. (The **SI** function call forces insert mode unlike the **TI** function call that toggles between insert and overstrike mode.)

Appendix B of this document has a complete list of all XyWrite function calls. You can use all of those function calls in programs except **TS** (Toggle Scroll Lock). **TS** cannot be entered into a program file, since you use it to begin and end the recording of a program.

You can use either the SX (Save Expression) or the SV (Save String) commands to save a function call to a macro. If you use SX, you must enclose the function call in quotes. For example:

```
«SX01,"SI"»
```

Refer to "Macros" for more information on SX and SV.

The next four sections describe three function calls that are especially useful in programming and a method for searching for function calls.

BLANK THE COMMAND LINE (**BC**) FUNCTION CALL

The **BC** function call clears the command line and moves the cursor to the start of the command line. Just as you press F5 before entering commands on the command line, you include **BC** in your programs when you want to execute a command. For example:

```
BC ca testXC
```

In program mode, XyWrite enters **BC** in your program when you press F5. However, you can quickly enter the **BC** function call into your program by pressing Ctrl Alt Shift B.

BLIND EXECUTE (**BX**) FUNCTION CALL

The **BX** function call executes a command from within a program without moving the cursor to or clearing the command line. If the command and any arguments are enclosed in parentheses ( ), square brackets [ ], or curly braces { }, XyWrite executes the command automatically. For example:

```
BX (ca test)
```

Otherwise, if they are not so enclosed, you must use the **Q2** (Execute Too) function call with the **BX** function call to execute the command. **Q2** is useful for executing a command string that may contain parentheses, square brackets, or curly braces. For example:

```
BX se  (Q2
```

NOTE #1: You cannot use angle brackets < > to enclose a command specified by the **BX** command.

NOTE #2: Searching for a carriage return with **BX** is different than with **BC** . When you use **BX** , enter an actual carriage return or wildcard ←; but use ◙ (XyWrite character 10) or wildcard ← with **BC** .

NOTE #3: You cannot include function calls within the **BX** command.

ESCAPE (**ES**) FUNCTION CALL

You can use the **ES** function call to perform any function normally performed by pressing Esc (for example: remove the currently displayed menu or Help screen, release selected text, or cancel the currently displayed command window).

SEARCHING FOR FUNCTION CALLS

XyWrite provides the ability to search for function call mnemonics from the command line. To do this, use the /FN (Function Call) switch with the SEARCH command in the format:

`se/fn` /*text* ♂*mn text*/

text (optional) is any text that precedes or follows the function call you want to search for.

♂ (XyWrite character 11) instructs XyWrite to interpret the next two characters as a function call.

mn is the two-character mnemonic for the function call. (Do not include a space after the function call mnemonic unless you are actually searching for a space.)

For example, to search for the **BX** (Blind Execute) function call:

Type: F5`se/fn /♂BX/`Enter

If the function is assigned to a key, you can use an alternate method. For example, to search for the **BC** (Blank the Command Line) function call:

Type: F5`se /`
Press: Scroll Lock
Press: F5
Press: Scroll Lock
Type: /Enter

XyWrite inserts the **BC** function call in the search string on the command line. Since this procedure requires you to turn Scroll Lock on and off, you cannot use it to write a program in XPL that searches for function calls.

MACROS

Macros are storage areas, or buffers, you can use to save strings or values. XyWrite provides several types of macros: ordinary, programming, and additional programming.

You are probably most familiar with the 36 "ordinary" macros (A-Z and 0-9), usually accessed with the F2 key. You can also use ordinary macros in your program. The content of an ordinary macro remains intact until either you overwrite it or you QUIT XyWrite.

When you use XPL, there are also two thousand programming macros, numbered from 00 to 1999, which can only be used with an XPL program. You define programming macros within a program, and the macros you use depend on how long you want the contents to remain intact:

- The contents of programming macros 00-99 vanish at the end of each program.
- The contents of programming macros 100-1999 remain intact until either they are overwritten or you QUIT XyWrite.

If you use programming macros exclusively for programs, you can reserve your ordinary macros for on-the-fly use; since programming macros are not accessible from the keyboard.

NOTE #1: Ordinary macros 0 through 9 are distinct and separate from programming macros 00 through 09.

NOTE #2: There are also 1000 programming macros, numbered 1000 through 1999 that are reserved for use by XyWrite menus.

NOTE #3: Whenever you run a subroutine, XyWrite reinitializes macro 00 with the contents of the command line.

Additional programming (&A-&Z and &0-&9) macros differ from the other types of macros because they can only be used with the LDPM (Load Program) and the RUN commands. Refer to "Loading a Program to a Key" in this section for information.

You can use the VA (Value of Variable) command to display the contents of ordinary macros and programming macros. Refer to "Reading the Current Value of a Variable" for more information.

You can also display the contents of any macros used throughout a program while debugging. Refer to "Debugging Procedures" for more information.

The next six sections describe embedded commands you can use with programming macros. There are three kinds of information you can store in a programming macro: a numeric expression (value), a literal string of characters, or a program. The kind of information you are saving determines the command you use to save it and the command you use to access it. The table below provides a quick reference guide to the three kinds of information and the commands you use.

	Contents of Macro		
	Numeric Value	String	Function Call or Program
Save a value to a macro	«SX01,44»	«SV01,Yes»*	«SX01,"**BC** "»** «SU02,**BC** save**XC** »
Insert macro into text	«PV01»	«GT01»***	Not possible directly
Insert macro on command line	«PV01»	«PV01»	Not possible directly
Use macro in an expression	«PV01»	«IS01»	N/A
Evaluate an expression	«SX01,«PV02»+1» (Numeric expression)	«SX01,«IS02»+"x"» (Concatenation)	N/A
Use macro in an IF statement	«IF«PV01»==44»	«IF«IS01»=="Yes"»	«IF«IS01»=="**BC** "»**
Execute the macro	N/A	N/A	«GT01»

N/A Not Applicable

* Equivalent to «SX01,"Yes"»

** "**BC** " is a function call and not literal text.

*** «PV01» works almost the same as «GT01»; «GT01» inserts the contents of the macro into text all at once and «PV01» inserts the contents more slowly, one character at a time.

You use the SX (Save Expression) command when you want XyWrite to evaluate a character or series of characters and save the result to a macro. The character(s) can be a number (such as 88), a numeric expression (such as the result of 2 + 2), or a XyWrite variable (such as the current left margin setting).

The syntax for the SX command is:

```
SX #,exp
```

is the number or letter of the macro you want to save the value to.

exp is either a number, a numeric expression, or a XyWrite variable you want evaluated. (You use the VA command when you want to save the current value of a XyWrite variable.)

XyWrite evaluates the expression, determines a final value from it, and saves the final value to the macro.

For example, the following code saves the number 40 to macro 01:

```
«SX01,40»
```

As a second example, the following code adds 25 and 25 and saves the result, 50, to macro 02:

```
«SX02,(25+25)»
```

(Parentheses are not required around the expression above; however, they make it easier to read.)

As a third example, the following code saves the current value of the Left Margin setting to macro 03:

```
«SX03,«VALM»»
```

As a fourth example, the following code saves the current filename to macro 04. Although the filename is a character string, the SX command is used because XyWrite must *interpret* the variable.

```
«SX04,«VA$FI»»
```

You can also use SX to save a literal string if you enclose the string in quotes. For example:

```
«SX05,"Yes"»
```

SAVING A LITERAL STRING (SV)

The SV (Save String) command saves a literal string to a specified macro. A literal string is any character or series of characters that XyWrite saves exactly as typed, without any interpretation or evaluation. XyWrite simply dumps the string into the specified macro.

The syntax for the SV command is:

```
SV #,string
```

is the number or letter of the macro you want to save the literal string to.

string is a literal string. If you omit the comma and *string*, XyWrite saves any block of text selected when you run the program.

For example, the following code saves the literal string "YES" to macro 99:

```
«SV99,YES»
```

As another example, the following code saves the literal string "345" to macro 04 and treats it as a literal string, *not* as a numeric value:

```
«SV04,345»
```

As a third example, the following code saves the literal string «VALM» to macro 05. XyWrite saves the six characters as text without any evaluation of the left margin. If the SX command had been used instead, the actual value of the current left margin would have been saved.

```
«SV05,«VALM»»
```

As another example, the following code saves the selected block to macro 06:

```
«SV06»
```

However, the following code *clears* the contents of macro 06:

```
«SV06,»
```

(The contents of macro 06 are cleared regardless of how its contents were saved — SX, SV, or SU.)

SAVING A STRING AS A SUBROUTINE (SU)

The SU (Save Subroutine) command is similar to SV, but the string is marked as a program.

The syntax for the SU command is:

SU #,*string*

is the number or letter of the macro you want to save the subroutine to.

string is the string you want to save as a program.

For example, the following code saves the string "**BC** sa**XC** " — which executes the Save command — as a subroutine to macro 98:

«SU98,**BC** sa**XC** »

If the SV (Save String) command was used in the example instead of the SU command, XyWrite would *not* execute the functions within the string when the program called for it. Instead, XyWrite would place the characters on the screen as if they were text.

NOTE #1: You can use either the PV (Put Variable) or the GT (Get Macro) command to run the subroutine.

NOTE #2: Whenever you run a subroutine, XyWrite reinitializes macro 00 with the contents of the command line.

DISPLAYING THE CONTENTS OF A MACRO (PV and GT)

You use either the PV (Put Variable) or the GT (Get Macro) command to display the contents of a macro.

The PV command interprets the contents of a specified macro and then inserts the characters, one at a time, at the current cursor location, scrolling as it goes. The GT command is similar to PV, but inserts the text of the macro all at once. Therefore, GT is faster than PV and is especially useful for large blocks of text.

If overstrike mode is on, the text output by the PV command overwrites existing text; however, text output by the GT command does not overwrite existing text when overstrike mode is on.

The syntax for the PV command is:

PV #

The syntax for the GT command is:

```
GT #
```

is the number or letter of the macro you want to access.

Both GT and PV leave the cursor at the end of blocks of text they output.

You can display the contents of a macro either in the text area or on the command line, depending on where the cursor is located when you execute the command.

For example, the following code saves the current value of the left margin to macro 01 and outputs that value at the current cursor location:

```
«SX01,«VALM»»«PV01»
```

For example, the following code inserts the contents of macro 200 at the current cursor location:

```
«GT200»
```

As another example, the following code includes the contents of macro 01 as an argument of the Search command on the command line:

```
BC se /«PV01»/XC
```

NOTE: If you used the SU (Save Subroutine) command to save the contents of the macro, you can use either the PV or the GT command to run the subroutine.

USING A MACRO IN AN EXPRESSION (PV and IS)

You use either the PV (Put Variable) or the IS (Insert Macro) command to access the contents of a macro within an expression.

NOTE: Before you can use either IS (Insert String) or PV (Put Variable) in a program, you must have previously established the macro with SV (Save String) or SX (Save Expression). SV and SX not only save strings and results, they also identify the macros as valid elements within the program.

When you use the PV command inside an expression, XyWrite interprets the content of the macro as a number and combines it with the other parts of the expression to give a numeric result. The IS command simply "dumps out" the content of a specified macro as a literal string in an expression.

The syntax for the PV command is:

```
PV #
```

The syntax for the IS command is:

```
IS #
```

is the number or letter of the macro you want to access.

Two factors determine which command you use:

- Type of contents (numbers or character strings)
- Type of expression

If the content of the macro is numerical, you use the PV command in any numerical or logical expression.

For example, the following code compares macros 04 and 06. If the numeric value of macro 04 is greater than the value of macro 06, the program jumps to the label *a* to continue.

```
«IF«PV04»>«PV06»»«GLa»«EI»
```

As another example, the following code compares the value of a macro directly to a numeric value. If the numeric value of macro 04 equals 10, execution goes to the label *a* to continue.

```
«IF«PV04»==10»»«GLa»«EI»
```

You use IS if the macro contains one or more alphabetic (non-numeric) characters. (See the Note below.) You can use the IS command within an IF statement to compare the string contents of one macro either to another macro or to a literal string enclosed in double quotes to compare two strings in an IF statement.

For example, the following code compares macros 10 and 40. If the string in macro 10 is the same as the string in macro 40, execution goes to the label a to continue.

```
«IF«IS10»==«IS40»»«GLa»«EI»
```

As another example, the following code compares the content of macro 02 to the literal string "XYWRITE":

```
«IF«IS02»=="XYWRITE"»«GLa»«EI»
```

NOTE: If the expression contains any of the following string operators or string functions, you must use the IS command *regardless* of the contents of the macro:

+ (Concatenation)
ε [238] (Is contained in)
≡ [240] (Contains)
@UPR
@SIZ
@CNV

For example, if you want to concatenate two numbers, you use the IS command.

Refer to "Flow Control Commands" and to "Relational Operators" for more information.

You can also use the IS command as a regular formatting command to insert the contents of a macro into the text at printout. For example, to insert macro X:

Type: F5 `is x` Enter

Result: The IS command appears in draft view as:

▲IS:X

When you use the PRINT or PRINTF command, information in the macro is printed. You can also use PRINTS with the file stored. However, the content of an IS command is not displayed in graphic view; only the marker appears.

EXTRACTING PART OF A MACRO (XS)

Sometimes you may find it helpful to separate (parse) a long string into several smaller strings (for example, to separate the extension from a filename in order to evaluate it). The XS (Extract String) command enables you to divide the contents of a macro into parts and then extract and save the parts to other macros.

The XS command searches the specified macro for the character(s) you designate. If it finds a match, XS creates three new macros: one for storing the information that precedes the matched string, one for the matched string itself, and one for the information that follows the matched string.

The syntax for the XS command is:

XS *#1,#2,#3,#4,#5*

#1 is the number or letter of the macro that contains the string you want to *search*.

#2 is the number or letter of the macro that contains the *string* you want to match. (You can use wild cards in this string. See the Note below.)

#3 is the number or letter of the macro in which you want XyWrite to store the information that *precedes* the matched string.

#4 is the number or letter of the macro in which you want XyWrite to store the matched *string*. (Unless you use wild cards in the string you want to match, the contents of this macro will be identical to the contents of #2.)

#5 is the number or letter of the macro in which you want XyWrite to store the information that *follows* the matched string.

NOTE: XyWrite accepts wild card characters in the string you want to match (#2 above). Use the same wild cards as you use with the Search command. To enter them into the program file, press Shift Esc followed by the wild card letter you want. For example, to search for a single number:

Type: Shift Esc N

For example, the following code searches macro A for the string stored in macro B. When XS finds the first occurrence of the string, it saves the matching string in macro D; saves the information preceding the string in macro C; and saves the information following the string in macro E.

```
xs A,B,C,D,E
```

If macro A contains a filename and macro B contains a period, the code would extract the extension from the filename. After the program is run, macro C would contain the filename without any extension, macro D would contain a period, and macro E would contain the file extension.

ATTACHING VALUES TO A MACRO AS A DEFAULT

Default SG enables you to attach a string of text or numbers to a macro.

The format for the SG default in SETTINGS.DFL or the printer file is:

```
df sg=n,string
```

n is any macro (A-Z, 0-9, or 000-1999).

string is the text or numbers you want to save to the macro.

For example, the following line in SETTINGS.DFL assigns the string, "Yes" to macro 999 as a default:

```
df sg=999,Yes
```

If you use the DF command, XyWrite accepts spaces in the *string* and honors upper- and lowercase letters. However, if you execute the DEFAULT command from the command line, you must use the /NV switch if *string* contains spaces and lowercase characters. Otherwise, XyWrite will recognize only the first word and stop at the first space you type in the string or convert all the letters to uppercase. For example:

```
d/nv sg=999,Text 2
```

NOTE: If you use DF SG to attach a lengthy string (over 80 characters) of text or numbers to a macro, you may experience problems. If you omit the DF in SETTINGS.DFL or the printer file, there is no size limitation.

LOADING A PROGRAM TO A KEY

Although you can load your XPL programs to ordinary macro keys, additional programming macro keys are available, which let you save your ordinary macros for on-the-fly use.

You assign a macro to a key with a two-character instruction representing a function call. Ordinary macros use an "at" symbol (@) followed by a letter or a number. Additional programming macros use an ampersand (&) followed by a letter or number. You can map them to any available keys in the keyboard file or create a whole new table for them.

To use an additional programming macro, first attach the two-character instruction to a key. For example, the following notation in a keyboard file assigns additional programming macro E to key 18 (E on an IBM keyboard):

```
18=&E
```

Then, load a program into memory and associate it with that function call. For example, you might write a program to help you print envelopes on your laser printer. To load that program and associate it with a key, the command might be:

Type: F5 `ldpm envelope.pgm,&e` Enter

Result: Whenever you press the key defined as &E, XyWrite executes the program.

You can load programs in your startup file or while you are working. You can also develop several sets of programs to be loaded at appropriate times. Each time you load a program to a key, XyWrite overwrites the previous one

DISPLAYING A MACRO ON THE STATUS LINE

You can use the PR (Prompt) command in a program to display the contents of a macro on the status line. To do this you specify the number of the macro, preceded by an @ character in the *message* argument of the PR command. This must be the last item in the message; XyWrite ignores everything after the number of the macro.

For example, the following code displays the contents of macro 99 on the status line:

```
«PR@99»
```

As another example, the following code includes the content of macro 98 as part of a message displayed on the status line:

```
«PRThe file is @98»
```

You can also use the VA command to display the contents of ordinary macros A-Z and 0-9. For this application, the VA command has the format:

```
va/nv @#
```

is the number of the macro you want to display.

For example, to display the contents of macro D:

Type: F5 `va/nv @D` Enter

Result: XyWrite displays the contents of macro D on the status line.

SAVING A MACRO TO A FILE

You can save the contents of a macro to a new file on disk with SAVE %. SAVE % enables you to do this without opening a window. For this application, the format for the SAVE command is:

`save %#,`*filename*

is the number (or letter) of the macro, the contents of which you want to make into a file. # can be A-Z or 0-9 or 00-1999.

filename is the name of the file you want to create (including drive and path, if necessary). If you don't specify a filename, XyWrite uses #.SAV.

For example, the following code creates a file named MYFILE and includes the contents of macro b without opening a new window:

```
BC save %b,myfileXC
```

CLEARING MACROS

XyWrite has three commands that enable you to clear macros.

- CLRASG clears all macros. To clear macros A-Z, 0-9, &A-&Z, &0-&9 and 100-1999:

Type: F5 `clrasg` Enter

- CLRSGT clears all ordinary macros but leaves programs assigned to programming macros (100-1999) and additional programming macros (&A-&Z and &0-&9) in memory. To clear macros A-Z and 0-9:

Type: F5 `clrsgt` Enter

- CLRXSGT clears only the additional programming macros (10-999). To clear macros 10 through 999:

Type: F5 `clrxsgt` Enter

FLOW CONTROL COMMANDS

The commands described in this section control the "flow" of a program, such as looping, jumping to a label, branching in response to a particular condition, and exiting a subroutine or the entire program.

LABELING A POINT IN THE PROGRAM (LB)

You use the LB (Label) command to designate a point in the program that you can jump to with the GL (Go to Label) command. You can insert the LB command anywhere in a program.

The syntax for the LB command is:

`LB` *label*

label is any length string you want to use to identify a specific point in the program. You can also use the LB command to include comments in your program.

If the string is lengthy (more than 80 characters), you can execute the LB command without a label. XyWrite opens a command window so you can enter any length string you want. When you finish entering text, press either F11 or Alt F1 to close the window. (To abort the command window, press Esc.)

For example, the following code labels the point in the program *changes*:

```
«LBchanges»
```

GOING TO A SPECIFIC POINT IN THE PROGRAM (GL)

You use the GL (Go to Label) command to move to a specific point of the program designated by the LB command.

The syntax for the GL command is:

`GL` *label*

label is the name of the point in the program where you want execution to flow.

For example, the following code directs the program to go to the point labeled *program* to continue execution:

```
«GLprogram»
```

NOTE: Label names are case sensitive; therefore, «GLend» does *not* jump to «LBEND».

USING IF STATEMENTS

IF statements enable you to test a condition and change the flow of the program depending on the results of that comparison. This conditional branching is particularly useful for creating a loop of instructions that you want executed repeatedly.

An IF statement evaluates an expression and determines whether the expression is TRUE or FALSE.

The format of the IF statement is important. IF statements contain two commands, IF and EI (End If), and take the following form:

```
IF expression trueaction EI
```

expression is a set of symbols that can be evaluated to have a particular value — either true or false. Often expressions contain relational operators or the Error command. You must enclose the expression to be evaluated within the IF command brackets («IF*expression*»).

trueaction is one or more commands that are to be executed if the expression is TRUE.

Unless the *trueaction* contains a GL command or an Exit command, execution continues immediately after the EI, *whether the expression is TRUE or FALSE*.

For example, the following code compares the value of macro 01 with the value of macro 02:

```
«IF«PV01»==«PV02»»«GLA»«EI»«GLB»
```

If the expression in the example is TRUE (the value of macro 01 is the same as the value of macro 02), the program goes to the label A to continue execution; if the expression is FALSE, the program goes to the label B to continue.

Every IF statement is paired with the next EI command. When an IF statement is false, execution jumps to the next EI command it finds. You may directly include (nest) one IF statement within another, taking care that the IF and EI commands are balanced. As an XPL alternative to nesting IF statements, you can use the GL (Go to Label) command in an IF statement to jump to another IF statement, and then use another GL command to return to the first IF statement. For example:

```
«IF«PV01»==«PV02»»«GLA»«LBB»«EI»

«LBA»«IF«PV03»==«PV04»»«GLB»«EI»
```

If the first expression in the example is TRUE, the program goes to the label A and compares the value of macro 03 with the value of text macro 04. If that expression is TRUE, the program returns to the label B in the first IF statement. If neither expression is true, execution continues after the second EI command.

EXITING A PROGRAM

There are two commands you can use to exit a program:

- EX (Exit and Continue)
- EX1 (Exit and Stop)

Usually you use the EX command at the end of a main program to stop the program. If EX is used in a subroutine, it exits the subroutine but continues with the main program at the point where the subroutine ends.

EX1 stops the program altogether, whether execution is in the main program or in a subroutine.

For example, the following code stops the program whether it appears within the main program or within a subroutine:

```
«EX1»
```

COMMANDS THAT RETURN VALUES

The commands described in this section insert values into your program from one of three sources: from the keyboard, from XyWrite, or from another XPL program. You can use any of these commands in an expression (such as an IF statement).

READING KEYBOARD INPUT

The RC (Read Character) command pauses the program and waits for you to press a key. When you do, the program continues with the next step.

Usually, the RC command is used in conjunction with the SX command to take a keystroke entry from the keyboard and save it to a macro. For example, the following code pauses the program, waits to receive a keystroke, and saves it to macro 01:

```
«SX01,«RC»»
```

In this case, if you pressed 1, macro 01 contains "1" and the program continues.

(If the SV command had been used in the example instead of the SX command, XyWrite would save the literal string "«RC»" to macro 01.)

TESTING FOR ERRORS

The ER (Error) command takes on a logical value of either TRUE or FALSE. XyWrite returns a value of TRUE if there was an error in the *previously executed command*; otherwise, it returns a value of FALSE.

You can only use the ER command with XyWrite immediate commands (for example: CALL, PRINT, STORE, etc.); embedded commands and function calls have no effect on ER.

The ER command is often used as part of an expression within an IF statement. If TRUE (there is an error), the next command is executed; if FALSE (there is *no* error), the program continues after the EI command.

For example, the following code uses the ER command after executing the SEARCH command. If the search returns an error message (such as "Cannot find item"), ER has the value of TRUE.

```
BC se /XyWrite/XC «IF«ER»»«EX»«EI»«GLSTART»
```

The example executes a search for the word "XyWrite." If it is found (there is no error message, i.e. ER is FALSE), the program goes to the label START (because of the GL command immediately following the EI). If the search returns the error message "Cannot find item", then ER is TRUE, and the program continues with the next command, EX, which ends the program.

You can also use the logical operator @NOT to test for no errors:

```
«IF@NOT(«ER»)»«GLSTART»«EI»«EX»
```

Refer to "Logical Operators" for more information.

CHECKING FOR FILENAMES

The EXIST command checks to see if a filename already exists in the directory you specify. If the file is not found, EXIST returns a value of TRUE. The format is:

```
exist d:filename
```

filename is the name of the file that you want to determine exists.

For example, the following code tests whether the file named in macro 01 already exists.

```
BX exist «PV01»Q2 «IF«ER»»«GL-Create»«EI»
```

If the file does not exist, XyWrite returns a value of TRUE, therefore, ER is TRUE, and the program goes to the label -*Create*.

READING CURRENT CURSOR POSITION

The CP (Character Position) command takes on the value of the current cursor location. The value returned is the number of characters from the beginning of the file.

Usually, the CP command is used in conjunction with the SX command to save the cursor position to a macro. For example, the following code saves the current cursor position to macro 01:

```
«SX01,«CP»»
```

Once you have saved the cursor position and moved to another place in the file, you can use the JMP (Jump) command to return to that position.

The syntax for the JMP command is:

`jmp` *n*

n is the number of characters from the start of the file. Each "←" counts as two characters: carriage return/line feed. The characters within embedded commands (that appear in expanded view) also count (for example `«RM7IN»` counts as 7 characters).

For example, to position the cursor on the 9885th character of the file:

Type: F5`jmp 9885`Enter

As another example, the following code returns to the position indicated in macro 01:

BC `jmp «PV01»` **XC**

READING CURRENT COLUMN POSITION

The CL (Column Location of Cursor) command takes on the value of the current column position of the cursor.

Usually, you use the CL command with the SX command to save the column location to a macro.

For example, the following code saves the cursor's current column location to macro 02:

`«SX02,«CL»»`

(In graphic view, CL displays the pixel position of the cursor rather than the column location.)

PASSING INFORMATION FROM THE COMMAND LINE

You can pass information to a program when executing it by including the information as an argument to the RUN command. The syntax for the RUN command is:

`run` *programfile,string*

programfile is the name of the file that contains the program.

string is any information you want to pass to the program.

When you include information as an argument to the RUN command, XyWrite automatically stores the information in macro 00. You can access it by including «IS00» or «PV00» in the program, depending on the type of information and how you want to use it.

For example, the following command line passes the string "10" to macro 00 for use in the program, SAMPLE.PGM:

```
run sample.pgm,10
```

(If you have loaded the program to a key, you can place just the string on the command line, and then press the key to execute the program. XyWrite passes the string on the command line to macro 00.)

When comparing macro 00 to a macro that contains a literal string, you must use «IS00». For example, the following code saves the value "data" to macro 01 and compares it to macro 00:

```
«SV01,data»«IF«IS00»==«IS01»»«GLSTART»«EI»
```

If the string "data" is passed to macro 00, macros 00 and 01 are equal and the program goes to the label *START*. If it is some other string, the program continues after the EI.

When comparing macro 00 to a macro that contains a numeric value, the information passed to the program must also be numeric; and therefore, can be accessed with «PV00». For example, the following code saves the value "10" to macro 01 and compares it to your input:

```
«SX01,10»«IF«PV00»==«PV01»»«GLSTART»«EI»
```

If the number 10 is passed to macro 00, macros 00 and 01 are equal and the program goes to the label START. If it is some other number, the program continues after the EI. (The string you enter must be numeric, because the program compares it to a macro that has a numeric value. If you enter a non-numeric string, the program stops and XyWrite generates an error.)

Refer to "Macros" for more information on the IS (Insert Macro) and the PV (Put Value) commands.

PASSING INFORMATION TO STARTUP.INT

From the DOS prompt you can pass information to the STARTUP.INT program through the EDITOR command. In this context, EDITOR has the format:

`editor` *d:\path\filename,d:startup,string*

d:\path is the drive:\path\ where the file is stored.

filename is the name of the file you want displayed once XyWrite is running.

startup is the name of an alternate initialization file. If you omit this argument, XyWrite runs STARTUP.INT.

string is any information you want to pass to STARTUP.INT.

For example, to run XyWrite and pass the string 1234 into STARTUP.INT through macro 00, at the DOS prompt (for example C>):

Type: `editor ,,1234` 

NOTE: You can include a program in STARTUP.INT that runs when you start XyWrite. Since STARTUP.INT is itself a program, your program is a subroutine within it. If you use EX1 in the subroutine, execution stops after the subroutine without returning to STARTUP.INT.

READING THE CURRENT VALUE OF A VARIABLE

You use the VA (Value of Variable) command to return the value of any XyWrite default setting (e.g., left margin value) or any environment setting (e.g., current window number). Since the VA command returns a value, you must use the SX (Save Expression) command to store the value to a macro.

The VA command has the format:

`va` *nm*

nm is the variable. Settings for variables other than defaults are preceded by a dollar sign ($). Appendix C, *VA Command Settings*, lists all the variables you can use with the VA command.

For example, the following code saves to macro 01 the value of the current tab settings:

```
«SX01,«VATS»»
```

As another example, the following code saves to macro 02 the value of the current window number:

```
«SX02,«VA$WN»»
```

Two VA settings are especially helpful to pass information to the program during execution:

- **VA $KC** (Key Code) displays the key code number of the last key pressed.

- **VA $SC** (Scan Code) displays the value of the last key pressed, plus the value of any current shifted state (Caps Lock = 256, Shift = 512, Alt = 1024, Ctrl = 2048, NumLock = 32768). For example, if Caps (256) and X (45) are pressed, $SC displays 301 (256+45).

NOTE: You may receive different values with VA $SC, depending on either the number of tables or the order the shifting keys are defined in your keyboard file. For example:

```
CTRL=29,99
ALT=56,98
SHIFT=42,54
CAPS=58,T:C
```

The last shifting key defined has a value of 256 (CAPS in the example); the next to last has a value of 512 (SHIFT in the example), etc.

If a variable has more than one argument, you can get the values separately.

For example `«VAIP1»` gets first line indent; `«VAIP2»` gets subsequent line indents.

In XyWrite, you can use VA settings in your program without first storing them to a macro. Two examples:

```
«IF«VA$WS»==0»

«IF«VA$AOP»=="C:\XY4"»
```

Getting the Size of a Macro

You can use the VA setting ¦*n* to display the number of characters in a macro (¦ = XyWrite character 124). It works similar to the string function @SIZ, but much faster.

For example, the following code returns the number of characters in macro 100:

```
va|100
```

Result: XyWrite returns the number of characters contained in macro 100 as if you had used the string function `@siz(«IS100»)`.

NOTE: If the macro does not exist, XyWrite returns a value of 65535.

Getting the Contents of a Macro

You can use the VA command to display the contents of ordinary macros A-Z and 0-9 and programming macros 10-1999. For this application, the VA command has the format:

```
va @#
```

is the number of the macro you want to display.

(XyWrite displays only the first 76 characters of the macro.)

For example, to display the contents of macro 108:

Type: F5 `va @108` Enter

Result: XyWrite displays the first 76 characters of macro 108 in the text.

You can use the /NV switch with the VA command to display the contents of the macro on the status line. For example, to display the contents of macro 109:

Type: F5 `va/nv @109` Enter

Result: XyWrite displays the contents of macro 109 on the status line.

Getting One Term from a Macro

Also, you can specify that XyWrite only display a certain portion of the contents of a macro by specifying the number of the argument. For example, if you put the TS (Tab Setting) command (.5IN,1.5IN) in macro 110, to display the second argument:

Type: F5 `va/nv @110,2` Enter

Result: XyWrite displays 1.5IN (the second term) on the status line.

NOTE: When using VA/NV with formatting commands, you must move the cursor back into your document and then press F9 to get a correct result. Otherwise, VA/NV will display the value of the XyWrite command line, rather than the value at the current cursor location.

MISCELLANEOUS XPL COMMANDS

Four miscellaneous XPL commands enable you to:

- Switch windows to an open file (GOFILE)
- Produce an audible signal (BEEP)
- Display a message from within a program (DEFAULT/NV MG)
- Replace a string without opening a file (RPLFIL or APFIL)

SWITCHING WINDOWS TO AN OPEN FILE

You can use the GOFILE command to switch to the window containing a specific file.

The format for GOFILE is:

```
gofile filename
```

filename is the name of the file to which you want to move.

For example, the following code goes to the window that contains the file named in macro 99:

```
BC gofile «PV99»XC
```

THE BEEP COMMAND

You can use the BEEP command to produce an audible signal at a point in your program. For example:

```
BX (beep)
```

DISPLAYING A MESSAGE FROM WITHIN A PROGRAM

You can use default MG (Message) to display a message on the status line that is overwritten only by error messages and other system messages. The default message returns when the error or system message goes away or the next time the program refreshes the screen (**FF**) or blanks the command line (**BC**).

The format for the MG default in SETTINGS.DFL is:

```
df mg=message
```

message is any message you want displayed on the status line as a default.

For example:

```
df mg=Have a Nice Day!
```

If you use the DF command, XyWrite accepts spaces in the *message* and honors upper- and lowercase letters. However, if you execute the DEFAULT command from the command line, you must use the /NV switch if *message* contains spaces or lowercase characters. Otherwise, XyWrite will recognize only the first word and stop at the first space you type in the message or convert all the letters to uppercase. For example:

```
d/nv mg=Have a Nice Day!
```

To remove the message, leave *message* blank. For example:

```
df mg=
```
or
```
d mg=
```

REPLACING A STRING WITHOUT OPENING A FILE

In XyWrite, two commands enable you to replace a string in a file without opening the file in a window. The difference between RPLFIL (Replace File) and APFIL (Append File) is that if APFIL cannot find the string you specify, it appends it to the end of the file, whereas RPLFIL does not.

NOTE: RPLFIL and APFIL were implemented to allow menus to modify certain XyWrite files, such as SETTINGS.DFL. They are not general purpose commands but are included here for completeness.

The formats of the two commands are:
`RPLFIL/=` *filename,nm=string*
`APFIL/=` *filename,nm=string*

= is a delimiter.

filename is the name of the file in which you want to replace the string.

nm is unique text that you want XyWrite to search for that immediately precedes the second delimiter in *filename*.

string is the text you want the value for *nm* replaced with.

XyWrite searches *filename* for a specific line of text (*nm*=) starting with a carriage return and replaces anything after the second = with *string*.

For example, the following replaces the existing MD BO value in SETTINGS.DFL with 23:

```
rplfil/= settings.dfl,md bo=23
```

Result: XyWrite searches the file SETTINGS.DFL for the text "←md bo=" and replaces the text after the = to the end of the line with "23".

NOTE: The delimiter can be any character, typically "=" or "<". You cannot use a comma as a delimiter. If a comma appears in the line of text you are searching for (*nm*), use a XyWrite character 6 (♠) as the first delimiter. In this case, XyWrite replaces everything after the second comma with *string*.

For example, the following replaces the contents of macro 1001 with the string, "NO":

```
rplfil/♠ settings.dfl,sg=1001,NO
```

Result: XyWrite searches the file SETTINGS.DFL for the text "SG=1001," and replaces the existing value from the comma up to the carriage return with "NO".

RELATIONAL OPERATORS

You can use the following relational operators in an IF statement to compare either two numeric expressions (with PV) or two string expressions (with IS):

Operator	Meaning
<	Less Than
>	Greater Than
<=	Less Than or Equal To (same as =<)
>=	Greater Than or Equal To (same as =>)
<>	Not Equal
==	Equal

When comparing numeric expressions, the relational operators function as normal arithmetic operators. For example, the following code evaluates the contents of two macros to determine if the number stored in macro 01 is less than the number stored in macro 02:

```
«IF«PV01»<«PV02»»«GLA»«EI»
```

If the number in macro 01 is less than the number in macro 02, the program goes to the label *A*. If it is not, the program continues after the EI.

You can also use <, >, = =, and <> to compare two strings. In this case, the operators function on the basic alphabetic sort rule (ASCII order). XyWrite compares each pair of letters (one from each string) until it finds a pair that does not match, and then it stops comparing. A character is "less than" another character if its XyWrite character number is lower.

For example, the following code evaluates the contents of two macros to determine if the string stored in macro 01 is less than the string stored in macro 02:

```
«SV01,Jones»«SV02,Smith»«IF«IS01»<«IS02»»«GLA»«EI»
```

In an alphabetic sort, the string in macro 01 (Jones) would be placed before the string in macro 02 (Smith). Since macro 01 is less than macro 02, the IF statement is TRUE, and the program goes to the label *A*.

If the string in macro 01 were "Walter", it would be sorted after "Smith." Then the string in macro 01 would be greater than the string in macro 02; and the IF statement would be FALSE, which would cause the program to proceed to the command immediately after the EI.

If the string in macro 01 were "jones", it would be placed after the string in macro 02, because the lowercase "j" has a higher XyWrite character value than the uppercase "S".

NOTE: Any customized sort table you have loaded into memory *does not* affect the comparison of two strings; XyWrite uses only the basic ASCII order sort rule.

ARITHMETIC OPERATORS

You use the following operators to perform arithmetic on numeric values:

Operator	Meaning
*	Multiplication
/	Division
+	Addition
-	Subtraction

XyWrite uses a standard mathematical hierarchy during calculations (multiplication, division, addition, and subtraction). It works from left to right, performing all the multiplication in the expression first, all the division second, and so on. Any operations in parentheses are performed first, using the standard hierarchy.

For example, the following code adds 10 to the number stored in macro 02, multiplies the total by the number stored in macro 01, and saves the result in macro 99:

```
«SX01,2»«SX02,5»«SX99,«PV01»*(10+«PV02»)»
```

Parentheses are required in this expression to establish the order in which XyWrite is to do the calculation. With the parentheses, the result is 30 (2 x 15); without the parentheses, the result is 25 (2 x 10 + 5).

LOGICAL OPERATORS

You use logical operators in IF statements to perform boolean (logical) operations on numeric or string expressions. Boolean operations return a value of either TRUE or FALSE.

Operator	Meaning
&	And
!	Or
@XOR	Exclusive Or
@NOT	Not

USING AND

AND (&) tests two or more conditions, *all* of which must be true in order for the expression to be TRUE. For example, the following IF statement compares the contents of two pairs of macros. If macro 01 equals 02 and macro 03 equals 04, the program goes to the label *A*; otherwise, the program exits.

```
«IF(«IS1»==«IS2»)&(«IS3»==«IS4»)»«GLA»«EI»«EX»
```

USING OR

OR (!) tests two or more conditions, *one or more* of which must be true in order for the expression to be TRUE. For example, the following IF statement compares the contents of two pairs of macros. If either or both comparisons result in a match, the program goes to the label *B*; otherwise, the program exits.

```
«IF(«IS5»==«IS6»)!(«IS7»==«IS8»)»«GLB»«EI»«EX»
```

USING EXCLUSIVE OR

Exclusive OR (@XOR) tests two or more conditions, *only one* of which can be true in order for the expression to be TRUE. For example, the following IF statement compares the contents of two pairs of macros. If only one pair of macros contains a match, the program goes to the label *C*. If neither or both pairs of macros contains a match, the program exits.

```
«IF(«IS9»==«IS10»)@XOR(«IS11»==«IS12»)»«GLC»«EI»«EX»
```

USING NOT

Not (@NOT) tests a condition, *which must be false* in order for the expression to be TRUE. @NOT functions the same as the relational operator < >. You must use parentheses around the expression being tested with @NOT. For example, the following IF statement compares the contents of two macros. If macros 15 and 20 are not equal, the program goes to the label *NEXT*.

```
«IF@NOT(«PV15»==«PV20»)»«GLNEXT»«EI»
```

@NOT is also very useful to test for no errors. For example, the following code tests for errors; and if there are none, the program goes to the label START:

```
«IF@NOT(«ER»)»«GLSTART»«EI»«EX»
```

STRING FUNCTIONS

You use string functions to operate on string expressions and return a value or another string.

Operator	Meaning
@UPR	Uppercase
@SIZ	Return size of string
@CNV	Convert function call to its mnemonic
@NUM	Strip punctuation and letters
@INT	Eliminate fractions after decimal point
@ABS	Return unsigned numeric value

CONVERTING LETTERS TO UPPERCASE

@UPR is used with IS (Insert Macro) to convert the letters of the string stored in the macro to uppercase. Parentheses are required around the IS command. You must use the SX (Save Expression) command to save the uppercase string.

For example, the following code converts the characters stored in macro 10 to uppercase and saves the converted string back to macro 10:

```
«SX10,@UPR(«IS10»)»
```

RETURNING THE NUMBER OF CHARACTERS IN A STRING

@SIZ is used with IS (Insert Macro) to return a value equal to the number of characters in a string. Parentheses are required around the IS command. You must use SX (Save Expression) to save the result.

For example, the following code saves the string "hello" in macro 04. @SIZ saves the number 5 in macro 05 because "hello" contains five characters.

```
«SV04,hello»«SX05,@SIZ(«IS04»)»
```

NOTE: Although XyWrite function calls are unique, single-unit representations, they return a value of 3 when evaluated by the @SIZ string function.

As an alternative, you can use the VA setting ¦*n* to display the number of characters in a macro. It works similar to the string function @SIZ, but much faster.

For example, the following code returns the number of characters in macro 04:

```
«SX05,«VA¦04»»
```

Result: XyWrite returns the number of characters contained in macro 04 as if you had used the string function `@SIZ(«IS04»)`.

NOTE: If the macro does not exist, XyWrite returns a value of 65535.

CONVERTING A FUNCTION CALL TO ITS TWO-LETTER MNEMONIC

Function calls that are read by the RC (Read Character) command are stored in hexadecimal form in XyWrite. @CNV can convert the stored form to the XyWrite two-letter function call mnemonic. @CNV is used with IS (Insert Macro) and parentheses are required around the IS command.

For example, the following code reads a keystroke into macro 01. If the keystroke is a function call, it is converted to two-letters, and stored back to macro 01.

```
«SX01,«RC»»«SX01,@CNV(«IS01»)»
```

If you press F5, XyWrite stores a three-byte function **BC** in macro 01. @CNV converts the stored form into the string "BC" and saves two text letters back to macro 01. You can then use the two letters in macro 01 as you use any regular text.

NOTE: You can use @CNV only with function calls. If you try to process a regular keystroke, @CNV will terminate your program. Before using @CNV, it is a good idea to test the keystroke with @SIZ. (@SIZ returns a value of 3 for a function call and a value of 1 for a regular keystroke.)

STRIPPING PUNCTUATION AND LETTERS FROM STRINGS

@NUM strips all punctuation and letters, leaving only digits and any existing decimal point. For example, the following code strips any letters and punctuation from the contents of macro 02 and stores the remaining digits in macro 11:

```
«SX11,@NUM(«VA@02»)»
```

If the macro does not contain a number, the program may exit.

ELIMINATING FRACTIONS FROM NUMERIC STRINGS

@INT eliminates fractions after a decimal point (without rounding) and keeps only the integer portion. For example, the following code divides the content of macro 01 by 2, eliminates any fractions after the decimal point, and saves the result in macro 02.

```
«SX02,@INT(«PV01»/2)»
```

If the macro does not contain a number, the program may exit.

STRING OPERATORS

You use string operators to manipulate literal strings. You can link (concatenate) strings together or you can locate a string within another string.

CONCATENATING STRINGS

You use the string operator + to concatenate two or more separate strings. To concatenate you use + with either the IS (Insert Macro) command inside the expression or literal strings enclosed in double quotes. Although XyWrite considers all strings to be linked as literals, you use SX (Save Expression) to save the finished concatenation; because XyWrite must interpret the expression in order to join the two strings.

For example, the following code concatenates the contents of macro 01 with the contents of macro 02 and saves the resulting string to macro 99:

```
«SX99,«IS01»+«IS02»»
```

If "My name is " (note the space after "is") was the string in macro 01, and "John" was in macro 02, macro 99 would contain "My name is John" after concatenation. If macro 01 contained the string "10" and macro 02 contained the string "20", macro 99 would contain the *string* "1020", *not* the number "30", after concatenation.

As another example, the following code concatenates the contents of macro 99 and the literal string ".XYZ" and saves the resulting string to macro 99:

```
«SX99,«IS99»+".XYZ"»
```

In XyWrite, you can also concatenate a string with a VA setting without first reading the value into a macro.

For example, the following code concatenates the names of the current drive and path and the literal string "\" and saves the resulting string to macro 01:

```
«SX01,«VA$PA»+"\"»
```

LOCATING A STRING WITHIN ANOTHER STRING

You use the string operator "ε" [238] to determine whether a particular string exists within another string and if it does, the position within the string where it is located. To enter character ε [238], press Ctrl Alt 238.

NOTE: In Graphic view under XyWrite IV (in Code Pages 437 and 850), character 238 often appears as a "¯" macron or overscore; in Draft and Expanded views, character 238 always displays as an "ε" epsilon. In all views under Nota Bene for Windows (in Code Page 1252), character 238 is displayed as "î" i-circumflex.

If the first string is found to be part of the second string, XyWrite returns a numeric value equal to the number of characters that *precede* the occurrence of the first string within the second.

For example, if the first string begins as the fourth character of the second string, XyWrite returns a value of 3. If the first string begins as the first character of the second, XyWrite returns the value of 0. If the first string is not found within the second string, XyWrite returns a value of -1.

You use the string operator "ε" [238] with the IS (Insert Macro) command to test for a specific value.

For example, the following code saves the string *abc* to macro 01 and saves the string *defgh* to macro 91. The IF statement tests whether the string operator returns a value of -1, meaning that the characters in macro 01 are *not* present anywhere in macro 91.

```
«SV01,abc»«SV91,defgh»«IF«IS01»ε«IS91»==-1»«GLNO»«EI»
```

Since the comparison of macros 01 and 91 returns a value of -1, the IF statement is true; and the program goes to the label *NO*.

As another example, the following code saves the string *abc* to macro 01 and saves the string *xyzabcd* to macro 92. The IF statement tests whether the string operator returns a value greater than or equal to 0, meaning that the characters in macro 01 *are* present somewhere in the string saved in macro 92.

```
«SV01,abc»«SV92,xyzabcd»«IF«IS01»ε«IS92»>=0»«GLYES»«EI»
```

The comparison of macros 01 and 92 returns a value of 3, because three characters precede *abc* in the string *xyzabcd*. Therefore, the IF statement is true; and the program goes to the label *YES*.

You use the string operator "≡" [240] to determine whether a particular string contains another string. The expression returns either TRUE or FALSE. You use the string operator "≡" [240] with the IS (Insert Macro) command to test for a specific value. To enter character ≡ [240], press Ctrl Alt 240.

NOTE: In Graphic view under XyWrite IV (in Code Pages 437 and 850), character 240 "≡" often appears as an "—" m-dash; in Draft and Expanded views, character 240 always displays as an "≡" equivalence or identity symbol. In all views under Nota Bene for Windows (in Code Page 1252), character 240 is displayed as an "ð" eth.

If the first string contains the second string, XyWrite returns TRUE and executes a conditional statement.

For example, the following code saves the string bc to macro 01 and saves the string abcd to macro 91.

```
«SV01,bc»«SV91,abcd»«IF«IS91»≡«IS01»»«GLYes»«EI»
```

Since macro 91 contains macro 01, the IF statement is TRUE; and the program goes to the label *Yes*.

DEBUGGING PROCEDURES

Testing consists of running all (or a portion) of your program and observing the results. This section describes using the XPL debugger and provides some general debugging tips.

USING THE XPL DEBUGGER

Default DB enables you to pause execution at certain points in your XPL program, so you can verify that the program is doing what you want. The format of the DB command is d db=x,y or df db=x,y. y is optional.

d db=0 turns debugging off [default].

d db=1 stops the program on IF statements.

d db=2 stops the program on labels.

d db=4 stops the program on JM (jump to Menu) commands.

d db=8 stops the program on carriage returns.

d db=16 stops the program on commands that don't execute properly.

You can add two or more values to stop the program at more than one type of entry:

d db=3 stops the program on both labels and IF statements.

d db=5 stops the program on IF statements and JM commands.

d db=6 stops the program on labels and JM commands.

d db=7 stops the program on labels, IF statements, and JM commands.

y (optional) is one of the following:

1 Ignore carriage returns when running the program.

2 Ignore tabs when running the program.

3 Ignore both.

d db=23,3 stops the program on all events except carriage returns and tabs.

Turn debugging on before you run your program by setting default DB to one of the values from 1 to 31.

When the XPL debugger stops the program after an IF statement, it displays the result of the evaluation followed by the expression. For example:

```
TRUE «IS109»=="tom"
```

When the XPL debugger stops the program after a label, it displays the label. For example:

```
«LBLoop»
```

You can continue debugging by pressing any key except @.

Pressing @ enables you to view the contents of a macro. When the program pauses (without going to the command line):

Type: @#Enter

is the number of the macro you want to display.

XyWrite does not display the characters as you type them; but when you press Enter, it displays the contents of the macro on the status line. For example, to display the contents of macro 500:

Type: @500Enter

GENERAL DEBUGGING TIPS

Here are some techniques to help you when you are debugging your program:

- Back up any files your program alters so you can rerun the program on the same file.
- Test each branch of your code to make sure it is working properly.
- Test boundary conditions and special cases. For example, if the program executes a command eight times, make sure the eighth and ninth cases provide the results you want.
- Don't add the ES (Error Suppression) command or the **DX** (Display Canceled) function call to your program until everything else is running smoothly.
- While you are debugging, use ordinary macros (A-Z and 0-9) in your program to save information. You can quickly check the contents of ordinary macros after the program runs either by using the F2 key or by displaying a macro directory (Ctrl F2). Also, the content of an ordinary macro is relatively permanent, because it remains intact until either you overwrite it or you QUIT XyWrite. After you finish debugging, you can change the ordinary macros to programming macros.
- You may find it easier to debug your program when it is displayed in expanded view.

• If XyWrite displays an error when you try to run your program, often it is due to an error in syntax — you used one or more XPL commands improperly. For example, here are four common errors:

"SX command requires a number." You used the SX (Save Expression) command instead of the SV (Save String) command. In general, you use SX when you want XyWrite to *evaluate* the expression and you use SV when you want XyWrite to save the literal string. You may also receive this message if the data being interpreted is not the right type of data; for example, using @NUM or @INT with letters instead of numbers.

"Go to label command requires a Label command." You have either misspelled a label name or used the wrong case. The LB (Label) command must define the same label that is referred to by the GL (Go to Label) command. For example, «GLend» does *not* jump to «LBEND».

"Mismatched logical or numeric operands." You tried to compare a string to a number. In an IF statement, you can compare strings to strings or numbers to numbers, but not strings to numbers.

"Function requires ID and expression." You have an error in syntax either with the SX (Save Expression) command or with the SV (Save String) command. The syntax for the SX command is:

`SX` #,*exp*

is the number or letter of the macro you want to save the value to

exp is either a number, a numeric expression, or a XyWrite variable you want evaluated.

The syntax for the SV command is:

`SV` #,*string*

is the number or letter of the macro you want to save the literal string to

string is a literal string.

PART 2 • COMMON APPLICATIONS

Part 2 presents some examples of commonly used XPL programming techniques:

- Using IF statements
- Monitoring keyboard input
- Displaying a selective directory

The examples provided for the various applications combine several of the concepts presented in Part 1. You can include many of the examples as parts of your own programs.

USING IF STATEMENTS

There are two common ways to use IF statements:

- To compare the contents of two or more macros
- To detect if a particular condition happens or fails to happen during program execution

COMPARING MACROS

IF statements establish the conditions to be compared; provide instructions to be executed in the event the condition is true; and provide instructions to be executed if the condition is false.

For example, the following IF statement compares macro 88 to macro 99:

```
«IF«PV88»==«PV99»»«GLNEXT»«EI»
```

If they are the same, the program branches to the label *NEXT*. If they are not equal, the program continues with the instructions immediately after the EI (End If) statement.

The instructions on what to do if the condition is true can also be a series of commands, as long as they end with the EI statement. If these instructions do not include a GL (Go to Label) command, execution continues immediately after the EI *whether the expression is TRUE or FALSE*.

DETECTING AN ERROR DURING EXECUTION

The most common application for monitoring a program during execution involves the use of the ER (Error) command within an IF statement.

The ER command checks for the occurrence of an error only in the previous command (not function calls). If XyWrite generates an error during program execution, ER returns a value of TRUE; if no error occurs, ER returns a value of FALSE.

For example, the following situations cause ER to return a value of TRUE:

- The SEARCH or CHANGE command does not find a match.
- The CALL command does not find the specified file.
- The filename specified with the NEW command already exists in the current directory.

When a command sets ER to TRUE, ER remains in that state only until the next command is executed. Therefore, include the ER command in an IF statement immediately after the command you want to test.

For example, the following program changes all occurrences of a double space to a single space:

```
«LBLOOP»BC ch /  / /XC «IF«ER»»«EX»«EI»«GLLOOP»
```

Since XyWrite does not generate an error while the program continues to find double spaces, ER has a value of FALSE. As in all IF statements, a value of FALSE causes the program to jump to the EI statement and continue execution with the next command. The program continues to execute in a loop until it cannot find any more double spaces.

When this occurs, XyWrite generates the error "Cannot find item" and ER has a value of TRUE. Then, the command immediately following the expression (EX) is executed and the program stops.

Refer to "Forcing an Error to Stop a Loop" in Part 3 for another example of this application.

MONITORING KEYBOARD INPUT

Occasionally you may need to write a program that pauses during execution to allow you to type in some information. The program waits for a keystroke and resumes when it receives input.

There are several ways you can do this. Two common ways are:

- To use a predetermined terminating character
- To use a counting method

USING A TERMINATING CHARACTER

You can define a character (for example, +) that you type to signal the program that you have finished entering information.

For example, if you are preparing a document with many headings, you may want to write a program that sets the UF (Use TypeFace), SZ (SiZe), and character mode for the heading, lets you input the heading, and then changes back to the proper typeface, size, and mode for the text. Since the headings are *variable* lengths, this is an ideal situation for a terminating character.

To use a terminating character, you first designate the character in a macro, then set up a routine to compare that macro to what is input, and finally direct the program accordingly.

For example, the code would look something like this in expanded view:

```
«SV01,+»BC uf helveticaXC BC sz 14ptXC M2 FF
«LBLOOP»«SX02,«RC»»«IF«IS01»==«IS02»»BC uf timesXC BC sz 12ptXC M1 FF
«EX»«EI»«PV02»«GLLOOP»
```

The code functions as follows:

`«SV01,+»` establishes + as the terminating character by saving it to macro 01.

BC `uf helvetica` places the UF command on the command line to change the typeface to helvetica.

XC executes the UF command.

BC `sz 14` places the SZ command on the command line to change the type to 14 point.

XC executes the SZ command.

M2 changes the character mode to Bold for the heading format.

FF refreshes the screen so that XyWrite can display the two command markers.

«LBLOOP» labels this point in the program *LOOP*.

«SX02,«RC»» waits for a character to be input from the keyboard (RC); and when the character is received, saves it to macro 02.

«IF«IS01»==«IS02»» compares the contents of macros 01 and 02 to determine if + was input.

BC uf times**XC BC** sz 12pt**XC M1 FF** «EX» are the instructions to be executed when the IF statement is TRUE (the content of macro 01 does equal the content of macro 02; e.g., the character just input was +). In that case, the program outputs the commands to change back to the text typeface, character mode, and size and stops (EX).

«EI»«PV02»«GLLOOP» are the instructions to be executed when the IF statement is FALSE (the content of macro 01 does *not* equal the content of macro 02). In that case, the program continues *after* the EI command, places the input character on the screen (PV02), and goes back to the label *LOOP* to receive another character.

If you want to use a carriage return as a terminating character, remember XyWrite carriage returns are actually stored internally in a set as a carriage return and line feed. The string function ε [238] comes in handy in this case to check whether the input character contains a single carriage return character. For example:

```
«SV01,←
»«SX02,«RC»»«IF«IS02»ε«IS01»==0»
```

(Make sure program mode is turned off before pressing Enter.)

The code functions as follows

«SV01,←
» saves the carriage return to macro 01.

«SX02,«RC»» saves the character just input to macro 02.

Refer to "Using a Terminator to End Keyboard Input" in Part 3 for another example of this application.

USING TWO DIFFERENT COUNTING METHODS

If you know that you always want to pause for an exact number of keystrokes, you can set up a loop to run that number of times. You do this by incrementing the value of a macro each time the program loops.

For example, if you are preparing a document that contains headings that are *always* eight characters long, you can set up a loop to execute eight times and then exit.

The following program is similar to the previous example except that here macro 01 is established as a counter and macro 02 contains the final value against which the counter is compared:

```
«SX01,1»«SX02,8»BC uf helveticaXC BC sz 14ptXC M2 FF «LBLOOP»«SX03,«RC»»«PV03»FF
«IF«PV01»==«PV02»»BC uf timesXC BC sz 12ptXC M1 FF
«EX»«EI»«SX01,«PV01»+1»«GLLOOP»
```

The code functions as follows:

`«SX01,1»` establishes a counter in macro 01 and sets the initial value for it to one.

`«SX02,8»` sets the maximum number for the counter to eight and saves it to macro 02.

BC `uf helvetica`**XC BC** `sz 14pt`**XC M2 FF** functions as described in the previous example and makes changes for the heading format.

`«LBLOOP»` labels this point in the program *LOOP*.

`«SX03,«RC»»«PV03»`**FF** waits for a character to be input from the keyboard (RC); and when the character is received, saves it to macro 03 and places it on the screen (PV03 **FF**).

`«IF«PV01»==«PV02»»` compares the contents of macros 01 and 02.

BC uf times**XC BC** sz 12pt**XC M1 FF** «EX» are the instructions to be executed when the IF statement is TRUE (the content of macro 01 equals the content of macro 02; i.e., the counter has reached the maximum number of characters as previously defined). The instructions function as those in the previous example to restore the text format.

«EI»«SX01,«PV01»+1»«GLLOOP» are the instructions to be executed when the IF statement is FALSE (the content of macro 01 does *not* equal the content of macro 02). In that case, the program continues *after* the EI command, increments the counter by one, saves the incremented counter to macro 01, and goes back to the label *LOOP* to receive another character.

NOTE: The method described in this example can be adapted to count in many circumstances.

As another example, if you often copy several consecutive files in a directory to a disk, you can write a program that lets you do that with one command. When you run the program, you display a directory, put the cursor on the first file to be copied, and specify the number of files you want copied as an argument to the RUN command. XyWrite saves the number you specify in macro 00. By using «PV00» in your program to access the number, you can copy a different number of files every time. (Refer to "Commands that Return Values" in Part 1 for more information.)

The code would look something like this in expanded view:

```
«SX01,1»«LBLOOP»BC copy a:GT XC
«IF«PV01»==«PV00»»«EX»«EI»«SX01,«PV01»+1»«GLLOOP»
```

The code functions as follows.

«SX01,1» establishes a counter in macro 01 and sets the initial value for it to one.

«LBLOOP» labels this point of the program *LOOP*.

BC copy a: enters the COPY command on the command line.

GT positions the cursor back in the directory.

XC executes the COPY command.

«IF«PV01»==«PV00»»«EX» compares the contents of macros 01 and 00 and gives the instruction to EXIT when the IF statement is TRUE (the content of macro 01 equals the content of macro 00; i.e., the counter has reached the maximum number of loops you defined with the RUN command).

«EI»«SX01,«PV01»+1»«GLLOOP» are the instructions to be executed when the IF statement is FALSE (the content of macro 01 does *not* equal the content of macro 00). In that case, the program continues *after* the EI command, increments the counter by one, saves the incremented counter to macro 01, and loops back to the label *LOOP* to execute the COPY command again.

To run this program, display a directory, put the cursor on the first file to be copied, and type the number of files you want copied to the A drive as part of the RUN command. For example, the following runs a program called COPY.PGM and passes the argument *10* into the program:

Type: F5 `run copy.pgm,10` Enter

This command line saves the value of 10 to macro 00. Macro 01 increments each time the program loops; and after the tenth file is copied (macro 01 equals 10), the program stops.

The second, Count Up method of counting accomplishes the same task more economically. The code would look like this in expanded view:

```
«CULOOP,00»BC copy a:GT XC «LBLOOP»«EX»
```

The code functions as follows.

The command line ("run copy.pgm,10") saves the value of 10 to macro 00.

A loop is established between the Count Up command «CU*TerminatingLabelname,macroID*» («CULOOP,00») and the actual «LB*TerminatingLabel*», here called «LBLOOP». All commands within the loop are executed as many times as the numeric value stored in the macro identified by the macroID (here, macro 00) of the Count Up command. In this example, the loop would be executed 10 times.

DISPLAYING A SELECTIVE DIRECTORY

The Document Information feature automatically generates summary information about every file you create. You can generate a selective directory based on the various summary items, such as the creation date, the login of the author, or keywords associated with the file.

There are two basic steps to preparing a selective directory:

- Specify the selection criteria as a default
- Display the directory

SPECIFY THE SELECTION CRITERIA AS A DEFAULT

You can use the XL default in an XPL program to generate a selective directory of document information. The form of the XL default is:

```
d xl=expression
```

expression contains the criteria you want XyWrite to use when selecting files for the directory. As part of the expression, you use the following field mnemonics to indicate document information items:

AU (Author) is the logon of the person who created the document.

LG (Logon) is the logon of the person who last modified the document.

CD (Creation Date) is the day, month, and year when the file was created. See Note on the next page.

MD (Modification Date) is the date of the last revision. See Note on the next page.

CT (Creation Time) is the hour, minute, and second when the file was created.

MT (Modification Time) is the time of the last revision.

CM (Comment) is any 44-character comment. This comment can be edited in each file at any time.

RV (Revision Number) is the number of times the file has been edited.

PJ (Project Number) is the number you entered in the Project Number field of the Document Info dialog box.

RP (Retention Period) is the number of days you entered in the Document Retention field of the Document Info dialog box.

KY (Keyword) is the word (or words) you entered in the Keywords field of the Document Info dialog box.

You use VA ^ to access the contents of a document information item you specify.

For example, the following prepares XyWrite to display information on files that have the word "is" somewhere in the comment field:

```
BX d xl="is"ε«VA^cm»<>-1Q2
```

To enter character [238], press Ctrl Alt 238.

NOTE: You must use the modifier @DAT with any of the date items. XyWrite converts the date to a format it can use to compare dates in document information. The format for the modifier is:

```
@dat("date")
```

date is a specific date you want XyWrite to use as a criteria for selecting files. Enter *date* in the format mm-dd-yy.

For example, the following code in an XPL program prepares XyWrite to display information on all files created on July 25,1992:

```
BX d xl=«VA^cd»==@dat("7-25-92")Q2
```

DISPLAY THE DIRECTORY

To display a selective directory of document information, use the /SL switch with the DIR command. For example, to display document information for all files in the NEW directory on the D drive that meet the criteria you specified with the XL default:

Type: F5 `dir/sl d:\new` Enter

PART 3 • PROGRAM EXAMPLES

Part 3 contains eight complete XPL programs to illustrate the following concepts:

Branching

Passing information from the command line

Using cursor position and column location to draw a horizontal line

Locating a specific string within another string

Forcing an error to stop a loop

Evaluating keyboard input and branching

Defining keyboard input and joining strings

Using terminators to end keyboard input to a program

All the programs and any related input files are included on the disk that accompanies this document. The instructions in this part of the document assume that you have inserted the XPL Program Examples disk into your A drive.

Some of these program examples perform useful functions and you can use them as is or as part of your own programs; other programs are included simply as illustrations.

The programs presented here were written as examples of one or more specific concepts — there are always many ways to write a program to accomplish the same task.

After a brief overview of the concept, the discussion of each program includes:

- An overview of what the program does plus step-by-step instructions you can follow for a demonstration
- A brief description of how the program functions
- A flowchart to illustrate the direction the program takes during execution [OMITTED]
- The program code in expanded view followed by an in-depth description of how each piece of the code functions
- A list of any of the program conditions and limitations

(Although each program on the disk begins with a Note that lists all the macros it uses, the list is not included in the description of the code.)

BRANCHING

When some programs encounter an error during execution, they change their direction and branch to another point in the program. Usually, the branch is part of an IF statement.

Sometimes you cannot branch on an error, because XyWrite does not generate one to signal the situation; for example, no error is generated when the program reaches the bottom of a file. One way to approach this problem is to have the program test for a condition. For example, you could have the program check the value of the $FE (File End) variable each time it moves down the list. When the value of $FE is 1, the cursor is at the end of the file and the program branches out of a loop and exits.

NUMBER.PGM DEMONSTRATION

The program NUMBER.PGM numbers a list of items automatically to illustrate the concept of branching. It also provides an example of using the VA (Value of Variable) command to determine if the cursor is at the end of the file. NUMBER.PGM uses TELELIST as an input file.

For a demonstration of NUMBER.PGM:

1. Display the input file. For example:

Type: F5`ca d:\path\telelist`Enter

2. Run NUMBER.PGM.

Type: F5`run d:\path\number.pgm`Enter

NUMBER.PGM DESCRIPTION

In order to number each item in TELELIST automatically, NUMBER.PGM places the Counter command (C1 on the command line, moves to the beginning of the first line item, executes the command, moves to the start of the next line item, executes the command again, and so on.

Since the number of items in the list may vary, NUMBER.PGM must be able to detect when it has reached the end of the file. Therefore, the program includes an IF statement that exits when the variable $FE (File End) indicates the end of the file.

NUMBER.PGM FLOWCHART [OMITTED]

NUMBER.PGM CODE AND EXPLANATION

NUMBER.PGM looks like this in expanded view:

```
TF «LBnumber»BC c1XC --FF EL CD «IF«VA$FE»==1»BC «PRDone.»«EX»«EI»«GLnumber»
```

The code functions as follows:

TF puts the cursor in the text area at the top of the file.

`«LBnumber»` labels this point in the program *number*.

BC `c1` places the Counter command on the command line.

XC executes the Counter command.

`--` inserts two dashes after the number.

FF refreshes the display.

EL moves the cursor express left to the beginning of the line.

CD moves the cursor down to the next line.

`«IF«VA$FE»==1»`**BC** `«PRDone.»«EX»` uses the variable $FE (File End) to determine if the cursor is at the end of the file. If the value of $FE is 1, the program has reached the end of the list; and it displays the message "Done" and exits. Otherwise, the program continues with the instructions following the end of the IF statement (EI).

`«EI»«GLnumber»` indicates the end of the IF statement (End If) and loops the program to the label *number* to continue execution.

NUMBER.PGM CONDITIONS AND LIMITATIONS

NUMBER.PGM assumes that the list is already displayed on the screen, that you are in insert mode, that each item is on a line by itself, that there are no blank lines between items, and that you want to number every line in the file.

PASSING INFORMATION FROM THE COMMAND LINE

Some programs work interactively with information you specify as an argument of the RUN command when you execute the program. Since the information may be different each time you use the program, it cannot be hard coded into the program.

MERGE.PGM DEMONSTRATION

The program MERGE.PGM takes a value you specify and searches for that value in an input file. MERGE.PGM can be run on a large mailing list to extract those records that are needed for a particular mailing and put them into a separate file. MERGE.PGM uses DATA3 as a data file.

For a demonstration of MERGE.PGM:

1. Make sure Window 9 is available.

2. Go to another window and display the input file.

Type: F5 `ca d:\path\data3` Enter

3. Run MERGE.PGM as follows:

Type: F5 `run d:\path\merge.pgm,01890` Enter

Result: The program finds all the records in the data file that contain the string 01890 and copies them to a new file in Window 9.

MERGE.PGM DESCRIPTION

MERGE.PGM takes the string passed in from the RUN command and saves it to macro 00. It searches the data file for records containing the string and copies those records into a new file, DATA.XYZ in Window 9.

The program has several branches to handle error conditions as well as to terminate the program.

MERGE.PGM FLOWCHART [OMITTED]

MERGE.PGM CODE AND EXPLANATION

MERGE.PGM looks like this in expanded view:

```
BX es 1Q2 #9 BX new data. xyzQ2 «IF«ER»»BX es 0Q2 BC «PR Please store and
execute the program again.»«EX»«EI»AS TF «LBcopy»BC search /«PV00»/XC «IF«ER»»BX
es 0Q2 «PR No more fields meet that criteria.»«EX»«EI»BC searchb /◙/XC
«IF«ER»»TF XD DF BC search /◙/XC DF AS CP ♪AS XD «GLcopy»«EI»CR DF BC search
/◙/XC DF AS CP ♪AS XD «GLcopy»
```

The code functions as follows:

BX `es 1`**Q2** suppresses any errors XyWrite generates. (See Note #1 below.)

#9 moves the cursor to window 9.

BX `new data.xyz`**Q2** creates a new file, DATA.XYZ in window 9.

`«IF«ER»»`**BX** `es 0`**Q2 BC** `«PR Please store and execute the program again.»«EX»` anticipates an error in creating a new file. If there is an error, the program disables error suppression, clears the command line, displays instructions, and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI). (Notice that error suppression is disabled *before* the instructions, in order to maintain the display on the status line.)

`«EI»`**AS TF** continues by moving the cursor back to the alternate screen and to the top of the data file.

`«LBcopy»` labels this point in the program *copy*.

BC `search /«PV00»/`**XC** searches for the argument you entered with the RUN command.

`«IF«ER»»`**BX** `es 0`**Q2** `«PR No more fields meet that criteria.»«EX»` anticipates an error in finding the argument in the data file. If there is an error, the program disables error suppression, displays a message, and exits. Otherwise, execution resumes with the instruction following the end of the IF statement (EI).

`«EI»`**BC** `searchb /◙/`**XC** continues by searching backward for the default record separator (a single carriage return/line feed combination).

«IF«ER»»**TF XD DF BC** search /◙/**XC DF AS CP** ♪**AS XD** «GLcopy» anticipates an error in finding the default record separator. If there is an error, the program assumes it has found the first record in the file and proceeds as follows:

- Goes to the top of the file (**TF**)
- Deselects any previously selected text (**XD**)
- Begins selecting a block of text (**DF**)
- Searches for the separator that signifies the end of the first record
- Ends the selected block (**DF**)
- Moves the cursor to window 9 (**AS**)
- Copies the selected block (**CP**) followed by a carriage return
- Moves the cursor back to the alternate screen (**AS**)
- Deselects the selected block (**XD**)
- Loops execution back to the label *copy* and continues

Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»**CR DF BC** search /◙/**XC DF AS CP** ♪**AS XD** «GLcopy» continues by moving to the next record, selecting it, and copying it as described in the previous set of instructions. Again, the program loops to the label *copy* and continues execution.

NOTE #1: The ES (Error Suppression) command is useful to prevent XyWrite from displaying a confusing assortment of error messages during program execution. XyWrite automatically disables error suppression when a program ends. However, it is good practice to disable error suppression (ES 0) at any exit points in the program. For example, MERGE.PGM contains two EX commands and error suppression is disabled prior to each of them to allow XyWrite to display error messages again.

NOTE #2: A carriage return in a program functions as a regular carriage return — it executes a command if the cursor is on the command line; it enters a hard return if the cursor is in the text. If you press Enter while program mode is on, XyWrite enters a XyWrite character 13 (♪) into your program. To enter a carriage return in a search within a program, make sure program mode is on, and press Ctrl Enter. In this case, XyWrite enters a XyWrite character 10 (◙) into your program. Alternatively, use the wildcard ← within a program:

BC search /←/**XC** .

MERGE.PGM CONDITIONS AND LIMITATIONS

MERGE.PGM assumes that you have not modified the default mail merge field and record separators, that the data file is displayed on the screen when you start the program, and that a file called DATA.XYZ does not already exist.

USING CURSOR POSITION AND COLUMN LOCATION TO DRAW A LINE

Some programs use information supplied by XyWrite during execution. This information varies, depending on current conditions, such as cursor position or column location; therefore, it cannot be hard coded into the program.

LINE.PGM DEMONSTRATION

The program LINE.PGM draws either a single or a double horizontal line by connecting two points in your file that you indicated with asterisks. (You can press Esc to terminate the program.)

For a demonstration of LINE.PGM:

1. Display any text file on the screen.

2. Using Space to move between them, mark the two ends of the line with an asterisk (*). For example:

```
*                                                    *
```

3. Run the program.

Type: F5`run d:\path\line.pgm`Enter

The program displays the message "For single line press 1; for double lines, press 2."

4. Press 1 to draw a single line (—) or press 2 to draw a double line (=).

Result: The program connects the two points marked by the asterisks with either a single or a double line and then deletes the asterisks.

LINE.PGM DESCRIPTION

LINE.PGM is organized into three sections. The first section sets up the line character for either a single line or double lines and saves the appropriate character to a macro key. The second section uses CP and CL to save the position of the two ends marked with asterisks. The third section draws the horizontal line. The values determined by CP and CL are used in IF statements throughout the program to compare the current cursor location with the end location to determine the length of the line that is needed.

[LINE.PGM FLOWCHART OMITTED]

3-11 and 3-12

LINE.PGM CODE AND EXPLANATION

The code for LINE.PGM looks like this in expanded view:

Defines Line Character:

```
«LBhome»BC For single line, press 1; for double lines, press
2«SX10,«RC»»«IF«VA$KC»==1»BC «PR Program stopped.»«EX»
«EI»«IF«PV10»==1»«SV04,-»«GLprogram»
«EI»«IF«PV10»==2»«SV04,=»«GLprogram»
«EI»«GLhome»
```

Marks two points:

```
«LBprogram»CI TF BC search /*/XC «IF«ER»»«GLerror»«EI»CL
«SX01,«CL»»«SX07,«CP»»CR BC search /*/XC «IF«ER»»«GLerror»«EI»CL
«SX02,«CL»»«GLline»
```

Draws the horizontal line:

```
«LBline»«SX06,0»BC jmp «PV07»XC BC GT «SX05,«PV02»-«PV01»-
2»«PV04»«IF«PV05»<0»«GLout»«EI»«LBloop»«PV04»«IF«PV06»==«PV05»»«GLout»«EI»
«SX06,«PV06»+1»«GLloop»«LBout»«PV04»«PR Done.»«EX»
```

Displays error message:

```
«LBerror»BC «PR The line ends were not marked correctly. Program stopped.»«EX»
```

The code functions as follows:

The first section prompts you to choose either single or double lines for the rule.

`«SX10,«RC»»` saves your response to macro 10.

`«IF«VA$KC»==1»BC «PR Program stopped.»«EX»` evaluates your response. If you pressed Esc (Key Code 1), the program displays the message "Program stopped." on the status line and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»«IF«PV10»==1»«SV04,-»«GLprogram»` evaluates your response. If you pressed 1, the program saves XyWrite character 196, the single line horizontal character (—), to macro 04; otherwise, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»«IF«PV10»==2»«SV04,=»«GLprogram»` evaluates your response. If you pressed 2, the program saves XyWrite character 205, the double line horizontal character (=), to macro 04. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»«GLhome»` loops execution back to the label *home* if you did not press 1, 2, or Esc.

The second section of the program marks the two ends of the line.

`«LBprogram»` labels this point of the program *program*.

`CI TF` puts the cursor into overstrike mode and moves it to the top of the file.

`BC` `search /*/` **`XC`** searches for the first asterisk, which should be the left end of the line.

`«IF«ER»»«GLerror»` anticipates that the file does not contain any asterisks. If there is an error, the program goes to the label *error*, displays instructions, and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»`**`CL`** `«SX01,«CL»»«SX07,«CP»»`**`CR`** continues by moving the cursor left one character (placing it directly on the asterisk), saving the column location and the cursor position to macro 01 and macro 07, respectively, and moving the cursor right one character so the second search will execute properly.

`BC` `search /*/` **`XC`** searches for the second asterisk, which should be the right end of the line.

`«IF«ER»»«GLerror»` anticipates that the file does not contain the second asterisk. If there is an error, the program goes to the label *error*, displays instructions, and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»**CL** «SX02,«CL»» continues by moving the cursor left one character (placing it directly on the second asterisk) and saving the column location to macro 02.

«GLline» directs the program to go to the label *line* to continue execution.

The third section of the program draws the horizontal line.

«LBline» labels this point of the program *line*.

«SX06,0» sets macro 06 to zero.

BC jmp «PV07»**XC BC GT** moves the cursor to the left end of the line, clears the command line (**BC**), and places the cursor back into the text area (**GT**).

«SX05,«PV02»-«PV01»-2»«PV04» calculates the distance between the two points, minus two to compensate for the asterisks; saves the result to macro 05; and outputs the first horizontal character at the current cursor location.

«IF«PV05»<0»«GLout» determines whether the two points are next to each other (i.e., the distance between them is less then zero). If the right end is next to the left end, the program goes to the label *out* and outputs the last horizontal character and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«LBloop»«PV04» labels this point in the program *loop*, and outputs a horizontal line character.

«IF«PV06»==«PV05»»«GLout» acts as a counter by comparing the number of horizontal line characters output (PV06) with the number of line characters required (PV05). If they are equal, the program goes to the label *out* and continues execution. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«SX06,«PV06»+1»«GLloop» continues by incrementing the counter by one, and loops execution back to the label *loop* to continue.

«LBout»«PV04»«PR Done.»«EX» labels this point in the program *out*, outputs the last horizontal line character and the message "Done", and exits.

«LBerror»**BC** «PR The line ends were not marked correctly. Program stopped.»«EX» labels this point in the program *error*, outputs an error message, and exits.

LINE.PGM CONDITIONS AND LIMITATIONS

LINE.PGM assumes you marked the two points of the line with asterisks before running the program and used spaces (rather than tabs) between the left point and the right point. It assumes that the first asterisk is the leftmost X position and the second asterisk is the rightmost.

LINE.PGM cannot be used in graphic view and can only use a monospace font to draw a line.

LINE.PGM leaves your keyboard in overstrike mode.

LOCATING ONE STRING WITHIN ANOTHER STRING

Many programs evaluate information you enter from the keyboard. Programs often use the logical operator @UPR to force all input into uppercase and therefore limit the number of options that the program must check. Some programs compare two groups of characters or strings to determine if they are the same. Sometimes, however, it is more important to determine if the string is included somewhere in another string. For some applications, it is useful not only to locate a string within another string, but also to extract it.

DRIVE.PGM DEMONSTRATION

The program DRIVE.PGM stores the current file to any valid drive you specify. (You can press Esc to terminate the program.)

For a demonstration of DRIVE.PGM:

1. Display any text file on the screen.

2. Run DRIVE.PGM.

Type: F5`run d:\path\drive.pgm`Enter

The program displays the message "Which drive to store on? (A, B, C, Esc to exit)."

3. Press A to store the file on the A drive, press B for the B drive, press C for the C drive, or press Esc to exit the program.

Result: The program stores the displayed document under its own name on the disk you specified.

DRIVE.PGM DESCRIPTION

DRIVE.PGM asks you to enter the drive letter where it is to be stored. The program checks that the drive letter you specified is included in the list of valid drives within the program. The program changes your response to uppercase rather than contain two lists of drives (one upper- and one lowercase) to anticipate your response. If the drive is valid, the program stores the file; otherwise, it loops back and asks you to enter the drive letter again.

DRIVE.PGM uses the XS (Extract String) command to separate the original default drive from the source path in order to restore the default before exiting. DRIVE.PGM also uses the variable $ER (Error) to check for an error. Since $ER cannot be cleared, DRIVE.PGM forces an error so that the value of $ER is known and the program can accurately evaluate the presence of other errors.

[DRIVE.PGM FLOWCHART OMITTED]

DRIVE.PGM CODE AND EXPLANATION

DRIVE.PGM looks like this in expanded view:

```
«SV01,»«SX02,«VA$FI»»«SV03,»«SX04,«VA$P\»»«SV05,:»«XS04,05,06,07,08»«LBloop»BC
Which drive to store on? (A, B, C, Esc to exit):«SX01,«RC»»«IF«VA$KC»==1»BC «PR
Program stopped.»«EX» «EI»«PV01»«SX01,@UPR(«IS01»)»«IF«IS01»ε"ABC"==-1»«PR A, B
or C only.»BX p 1Q2 BD «GLloop»«EI»BX es 1Q2 BC XC BX «PV01»:Q2
«IF«VA$ER»==195»BC «SX02,"Drive "+«IS01»+": is not ready."»
«PR@02»«EX»«EI»«SX03,«VA$P\»»BC store/nv «PV01»:XC «IF«VA$ER»<>11»«SX02,"Cannot
store to Drive "+«IS01»+":. Check drive and try again."»«PR@02»«EX»«EI»BX
«PV06»:Q2 «SX02,«IS02»+" stored to Drive " +«IS01»+":"»BC «PR@02»«EX»
```

The code functions as follows:

`«SV01,»` establishes macro 01 in preparation for holding the value of the key pressed.

`«SX02,«VA$FI»»` saves the current filename in macro 02.

`«SV03,»` establishes macro 03 in preparation for holding the path of the target drive.

`«SX04,«VA$P\»»` saves the current path to macro 04.

`«SV05,:»` saves a colon to macro 05.

`«XS04,05,06,07,08»` searches the source path name in macro 04 for a colon, stores the drive (the letter that precedes the colon) in macro 06, stores the colon in macro 07, and stores the rest of the path in macro 08.

`«LBloop»` labels this point of the program *loop*.

BC `Which drive to store on? (A, B, C, Esc to exit):` displays a prompt asking you to enter the letter that corresponds to the drive on which to store the current file.

`«SX01,«RC»»` saves your response to macro 01.

`«IF«VA$KC»==1»`**BC** `«PR Program stopped.»«EX»` evaluates your response. If you pressed Esc (Key Code 1), the program displays the message "Program stopped." on the status line and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«PV01» continues by putting your response on the command line.

«SX01,@UPR(«IS01»)» converts your response to uppercase.

«IF«IS01»ε"ABC"==-1»«PR A, B or C only.» compares your response (IS01) with a list of valid drives ("ABC"). If the string operator returns a value of -1, your response is not contained in the list, and the program displays the message, "A, B or C only."

BX p 1**Q2 BD** «GLloop» pauses execution for 1 second before deleting the character the user typed (**BD**) and going to the label *loop* to continue.

«EI»**BX** es 1**Q2** continues by suppressing any errors XyWrite generates.

BC XC forces the error, "There is no command on command line" (Error #11). Since the value of $ER cannot be cleared, this ensures that the value of $ER is 11, which enables the program to accurately evaluate the presence of other errors. (Since errors are suppressed, you will not see the message.)

BX «PV01»**:Q2** makes the drive you specified the default drive.

«IF«VA$ER»==195»**BC** «SX02,"Drive "+«IS01»+": is not ready."»«PR@02»«EX» anticipates the error "Drive not ready" (Error #195). If XyWrite generates error 195, the program concatenates the user's response with the text ": is not ready", saves the resulting string to macro 02, displays the string on the status line, and then exits.

«EI»«SX03,«VA$P\»» continues by saving the target path to macro 03.

BC store/nv «PV01»**:XC** executes the STORE command to the drive you specified. The /NV switch prevents XyWrite from displaying a verification message if the file already exists on the target drive.

«IF«VA$ER»<>11»«SX02,"Cannot store to Drive "+«IS01»+":. Check drive and try again."»«PR@02»«EX» anticipates an error saving to disk. If there is any error message other than 'There is no command on command line" (Error #11), XyWrite concatenates the user's response with the text "Cannot store to Drive...", saves the resulting string to macro 02, displays the string on the status line, and then exits.

«EI»**BX** «PV06»**:Q2** continues by making the source drive the default drive.

«SX02,«IS02»+" Stored to Drive "+«IS01»+":"»**BC** «PR@02»«EX» concatenates the message containing the filename, "Stored to Drive " and the target path; saves the resulting string to macro 02; blanks the command line; displays the string on the status line; and then exits.

NOTE: Before you can use either IS (Insert String) or PV (Put Variable) in a program, you must have previously established the macro with SV (Save String) or SX (Save Expression). SV and SX not only save strings and results, they also identify the macros as valid elements within the program.

DRIVE.PGM CONDITIONS AND LIMITATIONS

The drive letters specified in macro 01 (A, B, and C) must be valid, read/write drives on your system.

FORCING AN ERROR TO STOP A LOOP

In XPL, you can use the ER (Error) command in an IF statement («IF«ER»»...) to evaluate the result of a XyWrite command. The condition remains false until a command fails to execute properly (for any reason) and XyWrite generates an error; then the condition is true. You can use the ER command only with a XyWrite command that you normally enter on the command line. You can also use the logical operator @NOT to test for no errors.

You can freeze the display by using the **DX** function call. Since XyWrite does not have to update the screen as the program proceeds, **DX** speeds program execution. XyWrite automatically unfreezes the display when the program ends. However, it is good practice to turn on the display at any exit points in the program.

SPACE.PGM DEMONSTRATION

The program SPACE.PGM searches for all occurrences of two spaces and changes them to single spaces.

For a demonstration of SPACE.PGM:

1. Display any text file on the screen.

2. Run the program. For example:

Type: F5`run d:\path\space.pgm`Enter

Result: The program displays the message, "Working", and then "Done" when all multiple spaces in the file have been changed to single spaces.

SPACE.PGM DESCRIPTION

SPACE.PGM searches the open text file for all occurrences of two spaces and changes them to single spaces. To allow for multiple spaces, SPACE.PGM loops until XyWrite generates the error message, "Cannot find item." To do this, SPACE.PGM uses the logical operator @NOT to test for no errors. When an error occurs, execution resumes outside the loop and the program ends.

[SPACE.PGM FLOWCHART OMITTED]

FORCING AN ERROR TO STOP A LOOP

SPACE.PGM looks like this in expanded view:

```
DX BC es 1XC TF «LBloop»«PR Working . . .»BC ch/  / /XC
«IF@NOT(«ER»)»«GLloop»«EI»BC es 0XC DO BC «PR Done.»«EX»
```

The code functions as follows:

DX freezes the display.

BC `es 1`**XC** suppresses any error messages XyWrite generates.

TF moves the cursor to the top of the file.

`«LBloop»` labels this point in the program *loop*.

`«PR Working...»` displays a message on the status line.

BC `ch /  / /`**XC** changes all occurrences of two spaces to one.

`«IF@NOT(«ER»)»«GLloop»` tests that there are no errors. If there are no errors, the program goes to label *loop* to search for other occurrences of two spaces in a row. If there are errors, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»`**BC** `es 0`**XC DO BC** `«PR Done.»«EX»` continues by disabling error suppression, turning on the display, and displaying on the status line the message "Done."

SPACE.PGM CONDITIONS AND LIMITATIONS

SPACE.PGM assumes that a file is displayed on the screen.

EVALUATING KEYBOARD INPUT AND BRANCHING

You can write a program that lets you select from a list of activities and enter one or more characters to indicate your choice. Based on your input, execution branches to the point of the program that contains the correct code to accomplish the task.

CLEAN.PGM DEMONSTRATION

The program CLEAN.PGM deletes temporary files, backup files, or both from the current drive.

CAUTION: CLEAN.PGM is included to illustrate a concept. You should be very careful not to use CLEAN.PGM to delete .TMP files from your default drive. (The default drive is the one specified by the DR default setting, not the current drive.) Specifically, always make sure you never use XyWrite to delete the current XyWrite overflow file on your hard disk. The current overflow file has a creation date and time *after* the date and time of the current editing session.

NOTE: You can press Esc to stop CLEAN.PGM before Step 3. Once you have chosen the types of files to be deleted, you must press Ctrl[Break] to stop CLEAN.PGM.

For a demonstration of CLEAN.PGM:

1. Go to an empty window.

2. Run the program.

Type: F5`run d:\path\clean.pgm`Enter

The program displays the message "Please choose: Delete .(B)AK, .(T)MP, or (A)LL" on the command line and "Press B, T, or A (Esc to quit)" on the status line.

3. Press B to delete all the files on the drive with the extension .BAK, T to delete files with the extension .TMP, or A to delete files with either the .BAK or the .TMP extension.

Result: The program displays a list of all the files on the drive with the extension(s) you specified, places the DELETE/NV command on the command line, and deletes files (one at a time) until it reaches the end of the list.

CLEAN.PGM DESCRIPTION

CLEAN.PGM prompts you to enter a letter that corresponds to the types of files to be deleted. The program evaluates the response to make sure it is a valid choice. If not, it loops back and prompts you to choose again. Otherwise, CLEAN.PGM branches to a label corresponding to the chosen action and proceeds. CLEAN.PGM displays a list of files and deletes each one in a loop until there are no more files and XyWrite generates an error. The error branches execution to a different point of the program. The program ends by pausing briefly, displaying a message, and clearing the command line.

[CLEAN.PGM FLOWCHART OMITTED]

CLEAN.PGM CODE AND EXPLANATION

CLEAN.PGM looks like this in expanded view:

```
«LBstart»BC Please choose: DELETE .(B)AK, .(T)MP, or (A)LL«PRPress B, T, or A
(Esc to quit)»«SX01,@UPR(«RC»)»«IF«VA$KC»==1»BC «PR Program stopped.»BX p 1Q2
«EX»«EI»;*;←
;*; ←
«IF«IS01»=="B"»«GLbak»«EI»«IF«IS01»=="T"»«GLtmp»«EI»;*;
«IF«IS01»=="A"»«GLboth»«EI»BX beepQ2 «GLstart»←
←«LBboth»BC BX find *.bakQ2 «IF«ER»»«GLtmp»«EI»TF BC delete/nvGT «LBrpt»XC
«IF«ER»»«GLtmp»«EI»«GLrpt»←
←
«LBtmp»BC BX find *.tmpQ2 «IF«ER»»«GLend»«EI»TF BC delete/nvGT «LBrpt2»XC
«IF«ER»»«GLend»«EI»«GLrpt2»←
←
«LBbak»BC BX find *.bakQ2 «IF«ER»»«GLend»«EI»TF BC delete/nvGT «LBrpt3»XC
«IF«ER»»«GLend»«EI»«GLrpt3»←
←
«LBend»BX abortQ2 BC «PR Clean-up completed.»BX p 2Q2 «EX»
```

The code functions as follows:

`«LBstart»` labels this point of the program *start*.

`BC Please choose: DELETE .(B)AK, .(T)MP, or (A)LL«PRPress B, T, or A (Esc to quit)»` displays on the command line instructions for choosing the type of file(s) to be deleted.

`«SX01,@UPR(«RC»)»` saves your response to macro 01 and converts it to uppercase.

`«IF«VA$KC»==1»BC «PR Program stopped.»BX p 1Q2 «EX»` evaluates your response. If you pressed Esc (Key Code 1), the program displays the message "Program stopped" on the status line, pauses for one second, and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»;*;`
`;*;` indicates the end of the IF statement (End If). The comment lines ;*; are included to make the program more readable by separating this section of code from the next group of IF statements.

«IF«IS01»=="B"»«GLbak»«EI»
«IF«IS01»=="T"»«GLtmp»«EI»
«IF«IS01»=="A"»«GLboth» evaluates your response and branches to the appropriate part of the program. For example, if you pressed B, execution branches to the label bak. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»**BX** beep**Q2** «GLstart»← beeps and loops back to the label *start* because your response did not match any of the valid responses. A comment is not needed before the carriage return at the end of this section of code because the code ends with a GL command. Therefore, execution skips over the carriage return.

«LBboth» labels this point in the program *both*.

BC BX find *.bak**Q2** blanks the command line and finds all the files on the current drive with the extension .BAK.

«IF«ER»»«GLtmp» anticipates the absence of .BAK files on the current drive and directs the program to go to the label *tmp*. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»**TF BC** delete/nv**GT** continues execution by moving the cursor to the top of the file, entering the DELETE command with the /NV (No Verify) switch on the command line, and moving the cursor to the first filename in the text area.

«LBrpt»**XC** labels this point in the program *rpt*, which executes the DELETE/NV command.

«IF«ER»»«GLtmp» anticipates XyWrite returning the message "File not found" because all .BAK files have been deleted and directs the program to go to the label *tmp*. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«GLrpt» continues by looping back to the label *rpt* and executing the DELETE command. When XyWrite generates an error because there are no longer any files with the .BAK extension, execution branches to the label *tmp*.

The program uses two similar routines that correspond to the two other choices (TMP and BAK).»

«LBend» labels this point of the program *end*.

BX abort**Q2** clears the screen.

BC blanks the command line.

«PR Clean-up completed.»**BX** p 2**Q2** «EX» displays the message "Clean-up completed", pauses the program for two seconds, and exits.

CLEAN.PGM CONDITIONS AND LIMITATIONS

CLEAN.PGM must be run in an empty window. The program assumes that your directories do not include a header.

DEFINING KEYBOARD INPUT AND JOINING STRINGS

You can write a program that monitors the exact number of characters to be input and branches when that number is reached. In order to do this, the program must read characters as they are input, concatenate or link them together, and determine when the maximum number of characters has been input.

INPUT.PGM DEMONSTRATION

The program INPUT.PGM creates a new file, using 8 of the characters you input as a filename. After the eighth character you type, INPUT.PGM automatically inserts the filename extension .DOC. (You can press Esc to terminate the program.)

For a demonstration of INPUT.PGM:

1. Go to an empty window.

2. Run the program.

Type: F5`run d:\path\input.pgm`Enter

The program displays the message "Enter filename (or Esc to exit)."

3. Type eight characters.

Result: The program inserts a period and the extension .DOC after the eighth character and creates a new file, using the new filename.

INPUT.PGM DESCRIPTION

INPUT.PGM saves each character you input to macro 02 and compares the contents of macro 02 to an internal counter in macro 01. When you have typed eight characters, the program saves the entire contents of the command line to macro 03. INPUT PGM uses the XS (Extract String) command to remove any trailing spaces from the contents of the command line, leaving only the filename you typed. The program then concatenates the contents of macro 03 to the literal string .DOC. INPUT.PGM uses the SX (Save Expression) command to concatenate your input because XyWrite must interpret the expression in order to join the two strings. However, INPUT.PGM uses the IS (Insert Macro) command within the expression instead of the PV (Put Variable) command, which is only used in expressions for numbers or XyWrite default values.

INPUT.PGM also checks the environment in which the program runs to determine if there is a window available for the file or if there is a file in the current window.

[Omitted INPUT.PGM FLOWCHART]

INPUT.PGM CODE AND EXPLANATION

INPUT.PGM looks like this in expanded view:

```
«SX01,0»«SV02,»«SV03,»«SX04,«VAMG»»«SV05, »«IF«VA$FS»==512»«PR No window
available for new file.»«EX»«EI»«IF«VA$WS»==1»BX window nQ2 «EI»BX d/nv mg=
Enter filename (or Esc to exit)Q2 BC FF «LBchar»«SX02,«RC»»«IF«VA$KC»==1»
BX d/nv mg=«PV04»Q2 «PR Program stopped.»BX p 1Q2 BC «EX»«EI»
«PV02»«SX01,«PV01»+1»«IF«PV01»<>8»«GLchar»«EI»«SX03,«VA$CM»»
«IF" "ε«IS03»>0»«XS03,05,06,07,08»«SX03,«IS06»»«EI»«SX03,«IS03»+".DOC"»
BC new «PV03»XC BC BX d/nv mg=«PV04»Q2 FF «PR Done.»«EX»
```

The code functions as follows:

«SX01,0» sets the initial value for the internal counter in macro 01 to zero.

«SV02,» establishes macro 02 in preparation for holding the value of the key pressed.

«SV03,» establishes macro 03 in preparation for holding the new filename.

«SX04,«VAMG»» stores the value of any default message to macro 04. This is done so that the default can be restored at the end of the program.

«SV05, » saves a space to macro 05.

«IF«VA$FS»==512»«PR No window available for new file.»«EX» determines if all windows have text files in them. If the value of $FS (File Status) is 512, no windows are available; and the program displays a message stating that fact and exits.

«EI»«IF«VA$WS»==1»BX window nQ2 uses the variable $WS to determine whether there is a file open in the current window. If the value of $WS is 1, a file is open in the current window. In that case, the program opens the next available window. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»**BX** d/nv mg= Enter filename (or Esc to exit)**Q2** displays a message that prompts you to start entering characters. The default MG is used to ensure that the message displayed on the status line can only be overwritten by error messages and other system messages.

BC FF clears the screen.

«LBchar» labels this point in the program *char*.

«SX02,«RC»» saves your response to macro 02.

«IF«VA$KC»==1»**BX** d/nv mg=«PV04»**Q2** «PR Program stopped.»**BX** p 1**Q2 BC** «EX» evaluates your response. If you pressed Esc (Key Code 1), the program displays the message "Program stopped." on the status line, restores any original default message, pauses execution for one second, clears the command line, and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«PV02» continues by displaying your response on the command line, so you can see what you typed.

«SX01,«PV01»+1» increments the counter by one.

«IF«PV01»<>8»«GLchar» compares the value of the counter with the number 8. If you have not typed 8 characters, the program loops to the label *char* to process the next keystroke. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«SX03,«VA$CM»» saves the entire contents of the command line to macro 03.

«IF" "ε«IS03»>0»«XS03,05,06,07,08» eliminates the trailing spaces that $CM included with the contents of the command line. If a space is included in macro 03 anywhere but as the first character, the program searches the new filename in macro 03 for a space, stores the filename (the string that precedes the space) in macro 06, stores the space in macro 07, and stores the remainder of the trailing spaces from the command line in macro 08.

«SX03,«IS06»» saves the new filename in macro 03.

«EI»«SX03,«IS03»+".DOC"» continues by concatenating the contents of macro 03 (the new filename) with the literal string .DOC.

BC new «PV03»**XC** creates a new file, using the concatenated string as a filename.

BC BX d/nv mg=«PV04»**Q2** clears the command line and restores any original default message.

FF «PR Done.»«EX» clears the screen and displays a message on the status line.

INPUT.PGM CONDITIONS AND LIMITATIONS

INPUT.PGM does not make any allowances for typing mistakes.

USING A TERMINATOR TO END KEYBOARD INPUT

You can write a program that allows unlimited input from the keyboard. When you press the key that you previously defined as a signal or terminating character, the program stops accepting input and moves on to the next step.

TERMINAT.PGM DEMONSTRATION

The program TERMINAT.PGM creates a new file, using characters you enter as a filename. (You can press Esc to terminate the program, *before* you press +.)

For a demonstration of TERMINAT.PGM:

1. Go to an empty window.

2. Run the program.

Type: F5`run d:\path\terminat.pgm`Enter

The program displays the message "Enter filename, press '+' when ready (Esc to quit)" on the status line.

3. Type a filename and press +.

Result: The program creates a new file using the characters you entered as a filename.

TERMINAT.PGM DESCRIPTION

TERMINAT.PGM defines the + character as a terminating character and compares each character you enter against it. As in the program INPUT.PGM, TERMINAT.PGM saves the entire contents of the command line. However, it uses an alternate method to do this, which avoids the necessity of deleting trailing spaces. When the program encounters the terminating character, it stops accepting input and checks to see if the file already exists. If it does not, TERMINAT.PGM creates a file using the concatenated string as a filename. The program also does some environment checking.

[Omitted TERMINAT.PGM FLOWCHART]

TERMINAT.PGM CODE AND EXPLANATION

TERMINAT.PGM looks like this in expanded view:

```
«SV01,»«SX02,«VAMG»»«SU03,«SX04,«IS00»»»«IF«VA$FS»==512»«PR No window available
for newfile.»«EX»«EI»«IF«VA$WS»==1»BX window nQ2 «EI»BX d/nv mg=Enter filename,
press "+" when ready (Esc to quit).Q2 BC «LBchar»«SX01,«RC»»«IF«VA$KC»==1»BX
d/nv mg=«PV02»Q2 «PR Program stopped.»BX p 1Q2 BC
«EX»«EI»«IF«IS01»=="+"»«GLout»«EI»«PV01»«GLchar»
«LBout»«GT03»BX d/nv mg=«PV02»Q2 BC BX exist «PV04»Q2 «IF«ER»»BX new «PV04»Q2
«EX»«EI»«SX04,"File ("+«IS04»+") already exists."»«PR @04»«EX»
```

The code functions as follows:

`«SV01,»` establishes macro 01 in preparation for holding the value of the key pressed.

`«SX02,«VAMG»»` stores the value of any default message to macro 02. This is done so that the default can be restored at the end of the program.

`«SU03,«SX04,«IS00»»»` creates a subroutine in macro 03 that gets the contents of the command line. The subroutine saves to macro 04 the contents of macro 00, which is the contents of the command line. (Whenever you run a subroutine, XyWrite reinitializes macro 00 with the contents of the command line.)

`«IF«VA$FS»==512»«PR No window available for new file.»«EX»` determines if all windows have text files in them. If the value of $FS (File Status) is 512, no windows are available; and the program displays a message stating that fact and exits.

`«EI»«IF«VA$WS»==1»BX window nQ2` uses the variable $WS to determine whether there is a file open in the current window. If the value of $WS is 1, a file is open in the current window. In that case, the program opens the next available window. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»**BX** d/nv mg=Enter filename, press "+" when ready (Esc to quit).**Q2** displays a message that prompts you to start entering characters. The default MG is used to ensure that the message displayed on the status line can only be overwritten by error messages and other system messages.

BC clears the command line.

«LBchar» labels this point in the program *char*.

«SX01,«RC»» saves your response to macro 01.

«IF«VA$KC»==1»**BX** d/nv mg=«PV02»**Q2** «PR Program stopped.»**BX** p 1**Q2** **BC** «EX» evaluates your response. If you pressed Esc (Key Code 1), the program displays the message "Program stopped." on the status line, restores any original default message, pauses execution for one second, clears the command line, and exits. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«IF«IS01»=="+"»«GLout» compares the character you typed (saved in macro 01) to the literal string "+"; if they are equal, the program goes to the label *out* to continue execution. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

«EI»«PV01»«GLchar» continues by displaying your response on the screen, so you can see what you typed and loops to the label *char* to process the next keystroke.

«LBout» labels this point in the program *out*.

«GT03» runs the subroutine that saves the contents of the command line.

BX d/nv mg=«PV02»**Q2** restores any original default message.

BC **BX** exist «PV04»**Q2** checks to see if the filename you specified already exists in the current directory. If the file is not found, EXIST returns a value of TRUE.

«IF«ER»»**BX** new «PV04»**Q2** «EX» creates a new file with the name you specified if the EXIST command returns a value of TRUE. Otherwise, execution resumes with the instructions following the end of the IF statement (EI).

`«EI»«SX04,"File ("+«IS04»+") already exists."»«PR@04»«EX»` concatenates the literal text "File (," the filename, and the literal text ") already exists;" saves the resulting string to macro 04; displays the string on the status line; and then exits.

PART 4 • TIPS

Part 4 contains fourteen useful tips and hints to help you write effective, efficient XPL programs. In summary, they are:

- Plan ahead
- Put first things first
- Use the correct macro commands
- Use IF statements correctly
- Use efficient commands to complete the task
- Avoid unnecessary screen displays if possible
- Make your program files easy to read
- List all the macros the program uses
- Use unique label names
- Control the program's environment
- Decide whether to beep or not to beep
- Avoid unnecessary mode commands in your file
- Make your program easy to access
- Get help with complex utilities

TIP #1: PLAN AHEAD

Program execution begins with the very first character of the program file and usually continues, character by character, or command by command (see Note below). Therefore, it is important that you plan what each command will be, what its environment should be, and what impact it will have during execution. Consider all possibilities and try to anticipate problems. If the command is capable of causing an error, you should plan an alternate course of action for that event.

XPL programs can be as simple or as complex as you want to make them, but even the simplest set of instructions can benefit from preplanning.

NOTE: There are only five cases where execution does not continue with the next character or command: GL, IF, EX, EX1, and ;*;.

TIP #2: PUT FIRST THINGS FIRST

When you are planning your program, try to visualize how you would perform the various steps from the keyboard.

For example, when you execute an SE (Search) or a CH (Change) command, you usually start at the top of the file. Therefore, before any commands such as these, you should include the **TF** (Top of File) function call:

```
TF BC se /«PV00»/XC
```

Remember that many commands (such as ERASE, PRINT, ABORT, and SAVE) require you to verify what you want to do. In order to override the verification message, you can use the /NV switch with the following commands:

DELETE or ERASE
PRINT or TYPE
PRINTF or TYPEF
ABORT
SAVE
STORE
COPY
SAVEDEF or SAVESEL

Refer to Chapter 2 of the *XyWrite Command Reference Guide* for more information on the syntax for these commands.

For example, when saving to a file that already exists, you could use SAVE/NV or SA/NV.

As another example, when printing a directory or a selected block, you could use the PRINT/NV command.

TIP #3: USE THE CORRECT MACRO COMMANDS

The basic point to remember is that you must save strings of characters with the SV (Save String) command. To compare two strings in an expression, you must use the IS (Insert String) command. For most other cases, you use the SX (Save Expression) and the PV (Put Value) commands:

Use SX to save the RC (Read Character) command to a macro.

Use PV to output the contents of any macro in a program to the command line.

Use PV or GT to output the contents of any macro in a program to the text area.

Use PV to compare two numbers in an IF statement.

Use SV to save function calls to macros.

Use IS to compare two function calls in an IF statement.

Use IS to compare two letters (or two strings) in an IF statement.

Use IS to compare the contents of a macro to a literal string or a macro to another macro.

Use IS when concatenating the contents of two macros, but SX to save the finished concatenation.

Use IS with ε [238], ≡ [240], @UPR, @SIZ, and @CNV *regardless* of the contents of the macro.

TIP #4: USE IF STATEMENTS CORRECTLY

IF statements are very powerful programming tools when used properly. Here are some tips:

- If you are using the Equal relational operator in an expression, make sure you use *two* equal signs (==), not just one.

- Many operators require parentheses around the expressions they affect (for example, @UPR, @SIZ, and @NOT). If you omit the required parentheses, the operator will not perform properly.

TIP #5: USE EFFICIENT COMMANDS TO COMPLETE THE TASK

Use a command or function call that accomplishes exactly the task at hand. For example, use the function calls **GT** (Go to Text Area) and **GH** (Go to Header) to move the cursor to a specific location rather than the toggle **CC** (Change Cursor Header/Text). This saves the trouble of determining the current location of the cursor and provides more predictable results.

TIP #6: AVOID UNNECESSARY SCREEN DISPLAYS IF POSSIBLE

Here are some ways to avoid the messages XyWrite displays during program execution:

- Stop error messages from flashing while a program is running by turning on ES (Error Suppression):

```
BC es 1XC
```

XyWrite automatically disables error suppression when the program ends.

- Use the CH (Change) command rather than the CHANGE command. With CH, the changes happen faster and a rapidly changing, confusing screen display is avoided.

- Freeze the screen display altogether, using the **DX** (Display Canceled) function call. This has two benefits: you do not see the program working, and program speed is increased. (When you issue the **DX** function call, it affects all windows.)

XyWrite automatically unfreezes the display when a program ends. However, it is good practice to force the display to refresh at all exit points using the function calls **DO** (Display On) and **FF** (Force Fill).

TIP #7: MAKE YOUR PROGRAM FILES EASY TO READ

There are several things you can do to increase the readability of your program files:

- Type a hyphen between the label commands GL (Go to Label) or LB (Label) and the name of the label to make it easier to read in expanded view:

```
«GL-Loop»
```

Also, typing the label names with initial capital letters, as shown in the previous example, improves clarity. (However, remember that label names are case sensitive; therefore, «GLend» does *not* jump to «LBEND» or «LBEnd».)

• Complex programs are easier to read in expanded view if you follow every GL (Go to Label) command with a carriage return. For example:

```
«LB-A»BC saveXC «GL-B»
«LB-B»BC p This is a commentXC «EX»
```

Program execution jumps from «GL-B» to «LB-B», skipping the carriage return that immediately follows «GL-B».

• If you are certain the cursor will be on the command line, use Enter instead of **XC** . For example:

```
BC save←
```

Reminder: you cannot use a carriage return with the **BX** (Blind Execute) function call.

• Add comments liberally to your programs. You can use the string ;*; to indicate that the text that follows up to the next carriage return is a comment. As alternatives, you can place your remarks as part of an LB command, after a GL (Go to Label) command, or after an EX (Exit) command. However, many comments within the program may slow a complex program slightly. Try to include the majority of your comments at the end of the program.

• If you add a brief description of the program (including the program name) after a label at the top of the program file, you can use a long directory display (DIRL) to quickly find the program file you want.

For example, you can begin your program with a GL command that jumps to an LB command further down in the file. That leaves you space between the GL and the LB commands to add comments about the program's purpose.

If the program is one that you load onto a macro key, you can put a short (less than 36 characters), descriptive comment immediately after the GL command. The comment will be visible when you display the macro directory (Ctrl F2).

TIP #8: LIST ALL THE MACROS THE PROGRAM USES

A large program with many macros can be difficult for you to interpret, especially several weeks after you write it. It is a good idea to include a comment in your program that lists each macro the program uses and its purpose. Refer to any of the Example programs appended to this document for an example of this technique.

TIP #9: USE UNIQUE LABEL NAMES

A common error in large programs is naming two points in the program with the same label. If you have two LB (Label) commands with the same name, the GL (Go to Label) command moves only to the first occurrence. You can have as many identical GL commands as you want without any conflicts, but avoid using the same LB command more than once.

TIP #10: CONTROL THE PROGRAM'S ENVIRONMENT

A good XPL program is written so that it can be executed without trouble no matter what the conditions. Controlling the XyWrite environment is a four-part process. First, you must determine what conditions your program requires to execute properly. Second, you must save the current environment. Third, you must change the current environment to match these requirements. Fourth, you must provide a mechanism for reestablishing the original environment at the conclusion of the program.

For example, if your program requires that windows 1 and 2 be empty, you should check the status of those windows before getting into the program. You can save in programming macros the filenames of any displayed files, store them, and then bring them back into the same windows at the end of the program.

The primary tool for evaluating the current XyWrite environment is the VA (Value) command. You can use it to check the current status of practically any XyWrite default or condition, such as margin settings, current window number, available memory, etc. You can also place the current value of any condition into a macro. For example, the following places the current file and path name into programming macro 200 and the current window number into 201:

```
«SX200,«VA$FP»»«SX201,«VA$WN»»
```

Now that you know the filenames and locations, you can store the files to disk and call them back before exiting the program:

```
BC window «PV201»XC BC ca «PV200»XC
```

You might also want to check current status so that you can rearrange things to help your program. For example, many XyWrite users have their systems configured to automatically display a Help frame whenever they make an error. Since many programs depend on making errors to know where to branch, this can cause problems: the Help frame will appear and demand certain keystrokes to clear it and the program will try to execute underneath it. You can include the following in your program to anticipate this situation:

```
«SX300,«VA$EH»»BC default eh=0XC
```

This places the current status of Error Help in macro 300, and then turns Error Help off. At the end of the program, you can include the following command to return Error Help to its original state:

```
BC default eh=«PV300»XC
```

NW (New Window) and EP (Error Prompt) are two other system defaults that are useful to check. If NW is set to 1 or 2, your program may end up in windows it knows nothing about. If EP is enabled and you try to delete a file within the program, the program will stop because you have no opportunity to confirm the Delete command. You can get around the Error Prompt problem by using the NV (No Verify) switch with the Delete command in your programs. (Tip #2 describes other commands you can use the /NV switch with.)

TIP #11: DECIDE WHETHER TO BEEP OR NOT TO BEEP

You can disable the computer beep so that it never sounds.

To turn the beep off, use the ES (Error Suppression) command as follows:

Type: F5 es 1 Enter

XyWrite will not cause the computer to emit a beep in most situations, until you restore the beep.

Normally, a value of "1" signals that a XyWrite setting is ON; therefore, in this case, it may be helpful to think of the ES command as Error Silence and a value of "1" means that error silence is enabled.

NOTE: The ES (Error Suppression) command also prevents XyWrite from displaying error messages during program execution. XyWrite automatically disables error suppression when a program ends.

To turn the beep back on:

Type: F5 es 0 Enter

TIP #12: AVOID UNNECESSARY MODE COMMANDS IN YOUR FILE

Unintentionally including mode commands in your program file makes the program difficult to read, and, in some cases can affect the program's performance. This can happen inadvertently when you copy a part of the program to another place in the program file. Examine your program in expanded view to make sure there are no unnecessary mode commands.

TIP #13: MAKE YOUR PROGRAM EASY TO ACCESS

Once you have created your program, you run it to automatically execute the commands (and text) as if you have typed them from the keyboard. There are several ways to do this:

- Execute the RUN command from the keyboard
- Assign a simple program to a key in the keyboard
- Assign the program to a macro key

The techniques for accessing programs are described throughout this document.

TIP #14: GET HELP WITH COMPLEX UTILITIES

If you or your company could use a special utility to make your work easier, XYQUEST Technical Services might be able to help you.

Technical Service representatives are available to tailor-make XPL utilities and Help to your specifications. For example, if you need a way to assist entering data into standardized forms, a customized HELP file might be just the thing. Or perhaps you want a program to help convert XyWrite formatting commands into your typesetter's commands.

If you are interested in this service, please write to the Technical Projects Manager, who can provide you with a brief description of the terms and conditions of program development:

Technical Projects Manager
XYQUEST, Inc.
44 Manning Road
Billerica, MA 01821

APPENDIX A • XPL COMMANDS AND OPERATORS REFERENCE CHART

• General XPL Commands
- Append File (**APFIL**/= *d:filename,nm=string*)
- Beep (**BEEP**)
- Call Program File (**CA** *d:programfile*)
- Go to a File (**GOFILE** *d:filename*)
- Load Program (**LDPM** *d:programfile,#*)
- New Program File (**NE** *d:newfile*)
- Pause (**P** *t comment*)
- Put Function Call (**PFUNC** *fc*)
- Replace File (**RPLFIL**/= *d:filename,nm=string*)
- Run Program File (**RUN** *d:programfile,n*)
- Search for Function Call (**SE/FN** */text ♂mn text/*)

• Macro Commands
- Clear All Macros A-Z, 0-9, &A-&Z, &0-&9, and 100-1999 (**CLRASG**)
- Clear Macros A-Z and 0-9 (**CLRSGT**)
- Clear Programming Macros 10-999 (**CLRXSGT**)
- Extract part of a Macro (**XS** *#1,#2,#3,#4,#5*)
- Get Macro (**GT** #)
- Insert Macro (**IS** #)
- Put Variable (**PV** #)
- Save Expression (**SX** #,*numericexp*)
- Save String (**SV** #,*stringexp*)
- Save Subroutine (**SU** #,*string*)

• Flow Control Commands
- Exit and Continue (**EX**)
- Exit and Stop (**EX1**)
- Go to Label (**GL** *label*)
- If and End If (**IF** *exp trueaction* **EI**)
- Label (**LB** *label*)

• Commands that Return Values
- Character Position (**CP**)
- Column Location of Cursor (**CL**)
- Error (**ER**)
- Exist (**EXIST** *d:\path\filename*)
- Read Character (**RC**)
- Value of Variable (**VA** *nm*)

• Arithmetic Operators
- Addition (**+**)
- Division (**/**)
- Multiplication (*****)
- Subtraction (**-**)

- Relational Operators
  - Equal (==)
  - Greater Than (>)
  - Greater than or Equal (>=)
  - Less Than (<)
  - Less than or Equal (<=)
  - Not Equal (<>)

- Logical Operators
  - And (**&**)
  - Exclusive Or (**@XOR**)
  - Not (**@NOT**)
  - OR (**!**)

- String Function
  - Convert function call to its mnemonic (**@CNV**)
  - Return size of string (**@SIZ**)
  - Uppercase (**@UPR**)
  - Strip punctuation and letters (**@NUM**)
  - Eliminate fractions after decimal point (**@INT**)

- String Operators
  - Concatenation (**+**)
  - String exists within a string (**ε** [238])
  - String contains a string (**≡** [240])

- Function Calls Especially Useful in Programming
  - Blank Command Line (**BC** *command*)
  - Blind Execute (**BX** *(command)* or **BX** *command***Q2**)
  - Escape (**ES**)
  - See Appendix B for a complete list

- Defaults Especially Useful in Programming
  - Macro (**D/NV SG**=*n,string*)
  - Message (**D/NV MG**=*message*)
  - XPL Debugger (**D DB**=*n*)

APPENDIX B • XYWRITE FUNCTION CALLS (Updated)

For extended information, see TABLE FUNCTIONS<Helpkey> in XyWWWeb.U2.

AC Auto Check
AD Append Define to Text Macro
AK Accelerator key
AR Auto Replacement
AS Alternate Screen
AZ Auto-Replace On/Off Toggle
B4 Previous to Last Dialog Box
BC Blank Command Line
BD Backspace Delete
BF Bottom File
BK Break
BL Jump to left edge of guillemet command [NBWin]
BR Jump to right edge of current (balanced) guillemet command [XyWin,NBWin]
BS Bottom of Screen
BT Window Border Toggle On/Off
BX Blind Execute
C0 Counter Commands C0 to C9
CB Cycle Backwards
CC Change Cursor Header/Text
CD Cursor Down
CF Column Function
CH Clear Header
CI Clear Insert Mode
CL Cursor Left
CM Change View between Draft and Expanded (Codes)
CN Clear Num Lock
CO Comma
CP Copy Selected Block
CR Cursor Right
CS Clear Scroll Lock
CU Cursor Up
DB Selection Beginning
DC Select Column
DD Delete Selected Block
DE Selection End
DF Select Block
DL Select Line
DM Select Modify
DN Delete Selected Text, No Save to Delete Stack
DO Display On
DP Select Paragraph
DS Select Sentence
DT Display Total
DW Select Word
DX Display Off
DZ Selection End
EA Edit Second Argument
EC End Column (Table)
ED Entire Row Select (Table)
EE Erase Entry (Table)
EF Edit Footer, Header or Footnote
EL Express Left
EN Edit Next File
ER Express Right
ES Escape
ET Elapsed Time
EX Exit XyWrite
FC Flush Center

FD Find Difference
FF Force Fill (Screen Refresh)
FL Flush Left
FM Find Match
FR Flush Right
FS Fix Spelling
FX Transpose Text
GH Go to Header
GT Go to Text Area
H@ Help Screens
HC Home Column
HF Help Frame
HG Hide Graphic
HL Help
HM Home
IB IBM Mode
IR Insert Replacement
IT Insert Tab
JC Jump Over Command
JH Jump to Help
JM jump to Menu
KD Close Dialog box
KF Keyboard Flip
LB Line Begin
LD Linear Down
LE Line End
LL Linear Left
LR Linear Right
LT Login Toggle
LU Linear Up
M0 Mode 0, Reset
M1 Normal View
M2 Bold Mode
M3 Underline Mode
M4 Reverse Mode
M5 Bold Underline Mode
M6 Bold Reverse Mode
M7 Superscript Mode
M8 Subscript Mode
M9 Italic Mode
MC Mark Column
MD Move Down
MF MakeFunc. Make 3-byte codes. Format: "MF {byte3h}{byte2h}",
e.g. "**MF** 1F81"=**BC** (255-129-31) [NBWin]
MI Momentary Insert
MK No Marker Mode
MN Menu
MS Mouse
MT Math Function
MU Move Up
MV Move Selected Block
MW[xx] Microsoft Windows functions: [XyWin, NBWin]
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
hi Display Help Index
mn Minimize XyWrite screen
mv Display 4-headed arrow to move XyWrite screen

mw Move window
mx Maximize XyWrite screen
pa Paste text from Windows Clipboard
pl Paste link
pr Display information about Windows printer driver
ps Paste special
qu Quit
rm Restore current text window to maximum size
rs Restore XyWrite screen to previous non-max|min size
rw Restore current text window to previous non-max|min size
sf Repaint the screen
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 current text window

MX Prevailing Mode (same as M0, but does not get inserted in programs)
MZ Bold Italic Mode
NB Non-Breakable Block
NC Next Character
NF Next Formatted Page
NI Not IBM-Sensitive
NK Num Lock Toggle
NL Next Line
NM No Markers
NN Generic Wild Card
NO No Operation
NP Next Paragraph
NR Next Ruler
NS Next Sentence
NT Next Tab
NU No Undelete
NW Next Word
NX Next Window
OL[n] Outline Level [0-9] [NBWin]
OP Old Prompt
PC Previous Character
PD Page Down
PF Previous Formatted Page
PL Previous Line
PP Previous Paragraph
PR Print Screen
PS Previous Sentence
PT Previous Tab
PU Page Up
PW Previous Word
Q1 Spelling: ignore once [XyWrite]
Q2 Execute Too; Spelling: suspend [XyWrite]
Q3 Spelling: add word temporarily [XyWrite]
Q4 Spelling: add pair temporarily [XyWrite]
Q5 Spelling: add word permanently to personal dictionary [XyWrite]
Q6 Spelling: add pair permanently to personal dictionary [XyWrite]
Q7 Spelling: escape from spell-checking [XyWrite]
Q8 Spelling: replace questionable word with the highlighted word [XyWrite]
QH Quick Help
QP Quick Print
R0 XyWrite character 0
R1 XyWrite character 1
R2 XyWrite character 2
R3 XyWrite character 3
R4 XyWrite character 4
R5 XyWrite character 5
R6 XyWrite character 6

R7 XyWrite character 7
R8 XyWrite character 8
R9 XyWrite character 9
RB Rubout Word Before Current
RC Rubout Character
RD Rubout Selected Block
RE Rubout to End of Line
RK Record Keystrokes Mode
RL Rubout Line
RO Redlining On/Off
RP Rubout Paragraph
RS Rubout Sentence
RU Revert to national language translation (cf. func US)
RV Review
RW Rubout Word
RX Execute Recorded Keystrokes
S1 Acute Accent
S2 Grave Accent
S3 Umlaut
S4 Circumflex
S5 O Accent
S6 Tilde
S7 Underline
S+ Stack Up
S- Stack Down
SA Save
SC Spelling Auto-Check
SD Macro Directory
SF Store Files
SG Get Macro
SH Show Help
SI Set Insert
SK Show Macro Key
SL Save Files
SM Sum
SN Set Numeric Lock
SO Spell One Word
SP Show Page-Line Number
SS Set Scroll Lock
ST Show Triangle Help
SU Subtract Value
SV Save Selection
SW Show Window Menu
SY Synonyms
TE Table Entry
TF Top of File
TG Toggle to/from Expanded
TI Toggle Insert
TL Table Column Left
TM Table Move
TN Toggle Numeric Lock
TO Toggle Overstrike
TP Toggle Page Normal/Expanded
TR Table Column Right
TS Toggle Scroll Lock
TT Toggle TypeThru
TW Toggle Word
UD Undelete
UN Paste copy from clipboard
UP Unpad Spaces
US # Language to use in evaluation of variables (0=English)
VH Verify Help
WA Wild Alphanumeric
WC Wild Carriage Return

WG Draft View Without Page-Line Nos.
WH Emulate WordPerfect's implementation of the Home key
WL Wild Letter
WN Wild Number
WS Wild Separator
WW Wild String
WX Wild Any Character
WZ Graphic View
XC Execute
XD Cancel Selection
XH Clear Help
XM Express Middle
XN Transpose text (1=character; 2=word; 3=sentence; 4=paragraph)
XP Expanded View
XS Toggle the display of markers affected by scoping
XX Define floating accent (must also be entered in AC Table in Default file):
XX,≈ Cedilla accent pair
XX,/ Stroke/slash accent
XX," Double acute accents
XX,ε Macron accent
XX, ̨ [331] Ogonek accent
XX, ̌ [329] Caron accent pair
XX, ̆ [335] Breve accent
YD Cancel Selection (don't close command window)
ZT Zero Time
@A Macro Key (**@A** to **@Z**)
@0 Macro Key (**@0** to **@9**)
&A Macro Key (**&A** to **&Z**)
&0 Macro Key (**&0** to **&9**)
#1 Window No. 1 (**#1** to **#9**)
$A Call Help (**$A** to **$Z**)
$0 Call Help (**$0** to **$9**)

APPENDIX C • VA COMMAND SETTINGS (Updated)

Compendium of XyWrite VAriables RJH 5/6/98 Latest 11/4/05
For extended information, see TABLE VARIABLES<Helpkey> in XyWWWeb.U2.

You can obtain formatting information or the value of DeFaults by entering the appropriate VA command on the command line. For example, to see the "value" (name) of the current typeface:
VA/NV UF<cr>
The value of «VAUF» (VAriable UseFont), 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          Contents of S/G 100...
VA @1999         ...Contents of S/G 1999
VA @*1           Current «CP» CharPos [*0001]
VA @*5           Last CoPied|MoVed Text [*0005]
VA @*11          Recorded Keystrokes (func RK) [*000B]
VA @*26          First (or most recent) running program or frame [*001A]
VA @*27          Second running program or frame [*001B]
VA @*29          ? Next running program or frame [*001D] (increments)
VA @*31          ? [*001F]
VA @*213         Second LDPM program not assigned to a key [*00D5] (decrements)
VA @*214         First LDPM program not assigned to a key [*00D6]
VA @*248         &0 LDPM program [*00F8] (increments to...
VA @*257         ...&9 LDPM program [*0101]
VA @*265         &A LDPM program [*0109] (increments to...
VA @*290         ...&Z LDPM program [*0122]
VA @*13841       Current running program frame [*3611]
VA @*13843       KBD layout [*3613]
VA @*13844       First MeNu file selection [*3614]
VA @*13845       Second MeNu file selection [*3615] (increments)
VA @*13853       ? [*361D]
VA [it           Additive MoDe On (e.g."IT") at current «CP»
VA \902          Error Message 902
VA \@01          Error Message from S/G, e.g. «SX01,«VA$ER»»«PR\@01»
VA !01           Flag Value (initialization type) for S/G: 0=$tring (SV|SX),
                   2=SUb, 4=SX value, 16=expression evaluates FALSE,
                   24=expression evaluates TRUE, 255=nonexistent
VA §01           S/G Contains Only Numbers 0|1=Yes (§=XyWrite character 21)
VA |01           Size of S/G
VA *ul           MoDe number of MoDe mnemonic, e.g. «MDUL»
VA _bc           (First) Location of keyboard assignment, e.g.  ü
VA £13           Dollars
VA <m1#u1        Unit of Measure Conversion (<measure#unit), e.g. VA <PT#2IN
                   =PoinTs/2 inches
VA <m1]va        VAriable expressed in Measure(<measure]variable) ("]"=system
                   default), e.g.
VA <PT]SZ        PoinT SiZe, 1st and 2nd elements
VA <m1va         VAriable expressed in Measure(<measurevariable) (without
                   "]"=current value), e.g.
VA <PTSZ1        PoinT SiZe, 1st element only or VA <INIP (whereas standard
                   «VAIP» is expressed in Deci-Inches [=«VA<DIIP»])
VA command#      Value of Command Element, where "command"=embedded command or
                   default, "#"=element within command, e.g. «VAIP2»
```

VA (*cmdname,var* Value of Nested Command: "cmdname"=BOlabel|FAlabel|FM12or3|IGfilename|SSstylename,"var"=variable|element|keyword to solicit, e.g. «VA(SSCompendium,UF»

VA ⌂*dfbit* Bit status within default setting, where "df"=default name, "bit"=value to check; 0|1=On, e.g. «VA⌂HD4» (⌂=XyWrite character 127)

VA {*var* GC variable value

VA =*filename,string*= «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 Ignore End-of-File Marker 0|1=ignore EOF byte (Ascii-26)

VA $1A Ignore End-of-File Marker

VA 3D Three-Dimensional Effects: appearance of dialog boxes 0=2D|1=3D (XyWin, NBWin)

VA $3D Three-Dimensional Effects (XyWin, NBWin)

VA $AC Auto-Correct 0|1=On

VA AH Allow Hyphenation

VA $AH Allow Hyphenation

VA AL Automatic Leading

VA $AM*MoDe* Available MoDes (BI|BO|IT) in current type family (0|1=available):

VA $AM*bi* Bold Italic

VA $AM*bo* BOld

VA $AM*it* ITalic

VA $AN 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 AutOsave Path

VA AOT AutOsave Timer (min[,max default=min+5] in minutes)

VA AP Auto-Pause

VA $AR Auto-Replace 0|1=On

VA $AT ATtribute value returned by last ATTRIB command (0=R/W|1=RO)

VA AZ 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 Counter Numbering Style

VA BC ??

VA $BD BaD Words

VA BF Bottom Footnote

VA BG BackGround color

VA BI Beep Inhibit: 0|1=Display any format without error beep

VA $BI Beep Inhibit

VA BK BacKup Files 0|1=keep BAcKups

VA BL BLank Lines; NB: Base Line

VA BN ButtoN Description: Face,Width,Height (where Face is 0=Picture, 1=Text, 2=Both) (Windows); ?? in Xy4

VA $BN ButtoN Description (Windows); ?? in Xy4

VA $BQ*bo* Border Query 0|1=definition is present

VA BS Backspace Control

VA $BS Backspace Control

VA BT Bottom Margin

VA $BT Black and White Trace: value of BW command

VA BW Black and White (for CGA monitors) 0|1
VA $BW Monitor Type: Black and White 0|1=Color
VA BX Window Border Colors
VA $BX Window Border Colors
VA BZ Select Displayed Button Set (Windows)
VA C# NBWin: CodePage default(?)
VA $CA Cartridges currently loaded
VA CB Correction Beep: Xy4 values are frequency,duration; NBWin 0|1=Off
VA $CB Correction Beep
VA CF Change Footnote Separator: 0-use series 1 separator (even if no series 1 notes); 1-start with separator for first set of notes used
VA $CF Change Footnote Separator
VA CH ??
VA $CH ??
VA CK Spelling Checker: 0|1=ignore words that contain a number
VA $CK Spelling Checker
VA $CL Command Line last issued (40 chars max)
VA $CM Current content of CoMmand Line (80 chars max)
VA $CN Cartridge INstalled
VA CO COlumns
VA $CO COlumns
VA $CP Operating System Code Page (cf. LAnguage DeFault)
VA CR Cursor Type
VA $CR Carriage Return character(s)
VA CT ??
VA $CT Column Style: name of style used for current column (if specify «VA$CT5», returns style used in 5th column)
VA CV Change Verification Prompt 0|1=Confirm CHange commands
VA $CV Change Verification Prompt
VA CW Value of Margin Units MU (set in PRN)
VA $CW Value of Margin Units MU (set in PRN)
VA $CX Cursor Column Position
VA $CY Cursor Row Position
VA $CZ DDE Conversation Number: value of highest conversation currently active
VA D1 Delete Stack: # of entries,min.chars considered deletable unit
VA $D1 Delete Stack
VA $DA*d.Mmmm.yyy* Embed in text; displays date in specified «VA$DA*format*»
VA DB Debug a Program
VA $DB Debug a Program: Stop on 0|1=«IF|2=«LB|4= éú|8=<cr> |16=«ER»,Ignore 0|1=<cr>|2= 09|3=both
VA $DC Define Column currently selected 0|1=yes
VA DD Display Selected Blocks
VA $DD Display Selected Blocks
VA DE Define "Soft" End-of-Line Characters; VADE1=actual|VADE2=visual
VA $DE Define Ended 0|1=yes
VA $DF DeFine Status 0|1=currently selected
VA $DG DialoG File location
VA DH Discretionary "Soft" Hyphen
VA $DH Discretionary "Soft" Hyphen
VA DI Long DIrectory Display x,y,z (x=filesize divisor; y=lines of text displayed; z=0|1 (remove <cr>s)
VA $DI DIrectory Type in current window
VA $DK NBWin: Date of "K"reation, current file: octal date+time
VA $DL DeFauLt File Location
VA $DN Define ENd «CP»
VA $DO DOcument Information attached to file 0|1=yes
VA DP Decimal Point (USA="." Europe=",")
VA $DP Directory Path if «VA$WS»=2; otherwise=«VA$FP»
VA DR DRive:\Path\ for Temporary Files
VA $DR File at cursor position in currently-displayed DiRectory

```
VA $DS          Define Start «CP»
VA DT           Display Current Type (0|1|2|4|8|9|10|12|17|18|20)
VA $DT          Display Type
VA DU           Display Units
VA $DU          Display Units
VA $DV          Current DriVeletter
VA DY           Dye: activate color printer control codes
VA DZ           Date Format of DA and TODAY commands
VA $DZ          Date Format
VA EB           Error Beep
VA $EB          Error Beep
VA EC           ??
VA $ED          XyWrite Editor location
VA EE           Element End Margin Offset
VA EF           Special EFfects (immediate command)
VA EG           IBM EGA Control: 0=25 lines, 1=43 lines B&W, 2=43 lines Color
VA $EG          EGA Control
VA EH           Error Help
VA $EH          Error Help
VA EJ           Eject Last Page: 0-leave last page in printer/1-eject last
                  page
VA $EJ          Eject Last Page
VA EL           Extra Leading on current line, in INches*PoinTs/INch
VA $EL          ELement ID: internal ID of the last element clicked on
VA EP           Error Prompt: DEL, TY DeFined-block, TY dir, ABort, func SA,
                  del deltas, screen:printer font mismatch [Added in NBWin:
                  replace with CORRECT command in batch spell, Search/Replace]
VA $EP          Error Prompt
VA $ER          Last ERror Number
VA ES           Enable Scoping (apply format commands from «CP»=0, current
                  paragraph=1, replace previous=2)
VA $ES          Enable Scoping
VA ET           Element Top
VA $ET          Elapsed Time since  éWissued, else current time in format
                  hh:mm:sec.hundredths
VA EU           EUropean Punctuation
VA $EU          EUropean Punctuation
VA $EX          Current File .EXtension
VA $FA          Frame Attribute: internal ID of last frame clicked on
VA $FB          File Begin: cursor at TOF 0|1=yes
VA FC           Current value of FL|Flush Center|FR
VA $FC          Font Count: how many fonts available in current printer file
VA FD           Form Depth
VA $FE          File End: cursor at EOF 0|1=yes
VA FF           Form Feed
VA FG           ForeGround color
VA FH           Format Bar Height (Windows); ?? in Xy4
VA $FH          Format Bar Height (Windows); ?? in Xy4
VA $FI[/F]      Current FIlename [/F=long filename NBWin]
VA FL           Current value of Flush Left|FC|FR
VA $FM          Forms Mode: is current file a form? 0|1=yes
VA $FP          Drive:\Path\Current Filename
VA FQ           Format Bar Queue: items to be displayed on format bar
                  (Windows)
VA $FQ          Format Bar Queue: items to be displayed on format bar
                  (Windows)
VA FR           Current value of FL|FC|Flush Right
VA $FR          Last FRamename called
VA $FS          File Status: 0=no files open; non-zero=at least 1 file open
VA FT           Footnote Transition
VA FU           Footnote Unit
VA $FU          Footnote Unit
VA $FW          Full Screen Window: status of window (0=not FS|1=FS)
VA FX           Field Separator in data files
```

VA $FX Fixed Pitch=0|Proportional=1 (current font)
VA $FY Font FamilY for current font 1|2|3|4|5
VA FZ File Date format in DIRectories
VA $FZ Field Separator
VA GA Graphic Adapter
VA $GA Graphic Adapter
VA GB Global Library file location (Windows)
VA $GC GCI Status
VA GG Location of U5 file (General Counsel)
VA $GM*format* Mode Status in format, e.g.
VA $GM1 Printing a file
VA $GM2 Waiting for a + to continue printing
VA $GM8 Insert mode On
VA $GM16 Waiting for printing
VA $GM32 Printing suspended
VA $GM64 Message is displayed
VA $GM128 Spell or search has highlighted a string
VA $GM256 Automatic Uppercase On
VA $GM2048 Format bar being built
VA $GM4096 Sets internal flag for redisplaying a page
VA $GM8192 Running program
VA $GM16384 Executing BX function
VA GP Graphics Drive:\Path\
VA $GP Graphics Drive:\Path\
VA GU GUtter (current value)
VA GV Graphics Variable (set in PRN file)
VA $GV Graphics Variable
VA GW NBWin: ??
VA GX ??
VA $GX ??
VA HB Header Wildcard color (3rd element of L1 command)
VA $HB Header Wildcard color
VA HD Xy4: Fixed drives; NBWin: unknown numeric value
VA $HD Xy4: Fixed drives (C,D,E,F,G,H,...)
VA HI Header info
VA $HI Header info
VA HL Help Construction
VA $HL Help File location
VA HM ??
VA HN Header Normal display mode (2nd element of L1 command)
VA $HN Header Normal display mode
VA $HP Hewlett-Packard Printer File loaded 0|1=Yes
VA HR Header Reverse display mode (for CM) (1st element of L1 command)
VA $HR Header Reverse display mode
VA HT Header title (DocInfo)
VA $HT Header title
VA HV Hyphenation Values
VA $HV Hyphenation Values
VA HY Hyphenation 1=On (responds only to embedded «HY1», not to "d hy=1")
VA $HY Hyphenation dictionary file
VA $IG Import Graphics: number of IG commands in current file
VA II Italic Information font=0|attribute=1
VA $II Image Information: returns information about an image (compression, color, depth, width and height)
VA IM Image Mode Printing
VA $IM Image Mode Printing
VA $IN Cursor Inside Define 0|1=yes
VA IO Document Information: 0|1=On|2=Enter comments
VA $IO Document Information
VA IP Indent Paragraph; 3rd param (Xy4) is right indent
VA IT Insert Cursor Type
VA $IT Insert Cursor Type

VA IU	Information MenU 0|1=store file after clearing DocInfo dialog box
VA $IU	Information MenU
VA JB	Send PC Codes at Job Begin
VA $JB*PCCode#*	Job Begin PC code# set by a JB command, e.g. «VA$JB34» returns 0|1=specified
VA $JC	Number of Journal entries in file (Windows)
VA JL	Justify UnderLine Characters
VA $JL	Justify UnderLine Characters
VA JR	Journaling: 0=update existing journals|1=maintain journals but do not save|3=create journals for new files (maintain & save)
VA $JS	Size of Journal in bytes (Windows)
VA JT	Justification Type
VA JU	JUstification|NJ
VA JZ	Job End 0|1|2|3
VA $JZ*PCCode#*	Job Element 0|1=active setting for PC Code range, e.g. «VA$JZ120»
VA $KB	KeyBoard File location
VA KC	Key Click
VA $KC	Key Code
VA KP	(In NB-DOS: Special KayPro laptop=1)
VA $KP	(as above)
VA $KR	Keystrokes Recorded: number of keystrokes saved
VA KS	Keyboard [cursor] Speed
VA $KS	Keyboard [cursor] Speed
VA L0	Menu bar color control
VA $L0	Menu bar color control
VA L1	Command line color control
VA $L1	Command line color control
VA L2	Status line color control
VA $L2	Status line color control
VA L3	Ruler line color control
VA $L3	Ruler line color control
VA L4	Pull-down menus color control
VA $L4	Pull-down menus color control
VA LA	LAnguage: Xy4 current Code Page value 437|850; NBWin language name (e.g. «LAEnglish»)
VA LC	Line End Character
VA $LC	Line End Character
VA LF	(May be NB only & disabled in XyWrite; performs like VAWF)
VA $LF	(as above)
VA LG	Logic state (GC)
VA $LG	Logged-On User
VA LH	Low-High super/subscript control for Speedos
VA $LH	Low-High super/subscript control for Speedos
VA LL	Line Leading
VA LM	Left Margin
VA $LN	Current Line Number in P-L mode
VA $LO	Previous logical condition at same level as current level (GCC)
VA LQ	Letter Quality 0-9
VA LR	Left-to-Right Mode (for Hebrew etc) L2R=1;R2L=0
VA LS	Line Space
VA $LT	Logon Notes Toggle
VA LX	Main LeXicon path (LEXAM)
VA $LX	Main LeXicon path (LEXAM)
VA $LV	Current level of display (GCC)
VA LZ	Format Redlining Date
VA $LZ	Format Redlining Date
VA MA	# of chars to Find MAtch
VA $MA	# of chars to Find MAtch
VA MB	Message Box display location (0=status line, 1=message box) (XyWin, NBWin)

```
VA $MB          Message Box
VA MC           Minimum Size to Add Command to Stack: threshold minimum size
                  (XyWin, NBWin)
VA $MC          Minimum Size to Add Command to Stack (Windows)
VA MD           Current Character MoDe
VA ME           MEnu editing for deltas 0=command window|1=dialog box (XyWin,
                  NBWin)
VA $ME          Available MEmory
VA -M-          Main Dictionary memory
VA -e-          Expanded Dictionary memory
VA $M+          Memory Used by XyWrite
VA $M+0         All XyWrite memory
VA $M+1         All Code memory
VA $M+2         All Overlays memory
VA $M+3         Root memory (cseg)
VA $M+4         Editor code data memory
VA $M+5         Data memory
VA $M+6         Save/Gets program memory
VA -P1          Load program memory
VA -M1          Load file memory
VA -P2          Math/Program program memory
VA -M2          Math/Program file memory
VA -P3          Spell program memory
VA -M3          Spell file memory
VA -P4          Help program memory
VA -M4          Help file memory
VA -P5          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          Directory program memory
VA -M9          Directory file memory
VA -pa          Load printer program memory
VA -ma          Load printer file memory
VA -pb          Search program memory
VA -mb          Search file memory
VA -pc          Redline memory
VA -pd          Box drawing memory
VA -pe          Call/Save memory
VA -pf          Counters memory
VA -pg          Memory manager memory
VA -ph          Command table memory
VA -pi          Menus memory
VA -pj          Error messages memory
VA -pk          WYSIWYG memory
VA -pl          Styles memory
VA -pm          Soft fonts memory
VA -pn          Image memory
VA -po          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          RFT:DCA Export memory
VA MF           Mode for Forms
VA $MF          Mode for Forms
VA MG           Current MessaGe
```

VA $MG Current MessaGe
VA $MN MeNu File location
VA $MO File MOdified 0|1=yes
VA MR Metric Ruler
VA $MR Metric Ruler
VA MS Microspace Factor
VA $MS*md* Mode Status:
VA $MS1 Document Information dialog box displayed on STore|SAve
VA $MS2 Scroll Lock on
VA $MS4 No file open
VA $MS8 [Not used]
VA $MS16 Text selection started and NOT ended
VA $MS32 Selected text is on screen AND ended
VA $MS64 File open for read only
VA $MS128 Command window open
VA $MS256 Redlining on
VA $MS512 Directory displayed
VA $MS1024 Current file has never been saved
VA $MS2048 Current file has been edited since last saved
VA $MS4096 Insertion point in file (text=0|header=1)
VA $MS8192 Column selected
VA $MS16384 Current file includes Document Info (0=Yes|1=No)
VA $MS32768 REVIEW.TMP file (created by PRINTS|TYS) displayed
VA MT Military Time 0|1=Use MT
VA $MT Military Time
VA MU Margin Unit
VA $MU Margin Unit
VA MW Maximize Windows
VA $MW Mouse Window number (current location of mouse) (Xy4)
VA MX RAM committed to DICT.SPL
VA $MX Mouse X pixel|row position (Xy4)
VA MY MagnifY Dialog Boxes: specify SZ and Windows font (Windows)
VA $MY Mouse Y pixel|row position (Xy4)
VA $MZ*md* 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 command in a command window. Cleared means we are entering a command 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 Non-Printable Area
VA NB ??
VA NC Normal Carriage Return
VA $NC ??
VA ND Network Drives
VA $ND Network Drives
VA NE No Errors from Printer: 0|1=ignore spurious errors
VA $NE No Errors from Printer
VA NI No Index: suppress printing of indices
VA NJ No Justification|JU
VA NL Network Login Path
VA $NL Network Login Path

VA NM	No Modification Mode: 0|1=text marked with «NM1»...«NM0» commands can't be changed
VA NP	No Pause
VA $NR	No Ruler (Xy4)
VA $NU	UNused Printer Memory
VA NW	New Window
VA $NW	New Window: 0=no auto windows; 1=auto, no ABort if CAll in DIRectory display; 3=auto, ABort if CAll in DIRectory display
VA O1	Options: Error correction between screen<>printer fonts
VA $O1	Options
VA OB	Overstrike Beep 0|1=On
VA $OB	Overstrike Beep
VA OC	OCtagon Control: define shape of radio buttons (Windows)
VA $OC	OCtagon Control
VA OD	Offset Display 0-7
VA $OD	Offset Display
VA OE	Open Editor on LAN 0|1=Keep Editor open after reading shared code (faster performance)
VA $OE	Open Editor on LAN
VA OF	OFfset
VA OH	??
VA $OH	??
VA OL	OutLine Fonts Drive:\Path\
VA $OL	OutLine Fonts Drive:\Path\
VA OM	Old Masticon (PCLEX=0; Microlytics=31)
VA $OM	Old Masticon (PCLEX=0; Microlytics=31)
VA $OO	Command Override On 0|1=On (Xy4)
VA OP	Number of lines for OrPhans
VA OR	ORientation
VA OS	One-Sided Printing (can also embed «OS0»|«OS1»)
VA $OV	Overflow File Drive (if overflow file exists)
VA $P?*PCCode#*	P? returns 1 if specified PC code exists in PC table, e.g. «VA$P?120»
VA P.	Truncated Path
VA $P.	Truncated Current Drive:\Path Name (14 chars max)
VA $P\	Current Path, add Backslash "\" in Root Directories
VA $PA	Current Drive:\Path
VA $PB	Bottom Depth defines command controlling text length (0=Page Length PL|1=Bottom Margin BT)
VA $PC*PCCode#*	PC returns 1 if specified PC code exists in PC table, e.g. «VA$PC34»
VA $PC	PC Code returns 1 if specified PC code exists in PC table
VA PD	Pad Spaces
VA $PD	Pad Spaces
VA $PE	Page Elements: number of page elements in current file
VA $PF	Current Printer File (SETP) selection: $PF1=port#, $PF2=filename, $PF3=description
VA $PF7	Current SETP setting (number only)
VA PG	??
VA $PG	Current Page Number in P-L mode
VA PK	Page Break Color
VA $PK	Page Break Color
VA PM	Printer Memory (set in PRN file)
VA $PM	Printer Memory
VA PL	Page Length
VA $PR	Printer File location
VA PT	Numeric Print Type
VA PW	Page Width
VA PX	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 R2	Mouse Double Click

```
VA $R2          Mouse Double Click
VA $RA          Read Attribute displays absolute number of current character
                  MoDe
VA RB           Reverse Buttons
VA $RB          Reverse Buttons
VA $RC          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           Redline Data
VA $RD          Redline Data
VA $RE          0|1=Read-Only File
VA RG           ??
VA RI           Mouse Repetition Interval
VA $RI          Mouse Repetition Interval
VA $RK          Record Keystrokes 0|1=On
VA RL           Ruler Markers
VA $RL          Redlining 0|1=On
VA RM           Right Margin
VA RN           Round Off Line Numbers to nearest .5 in Page-Line display
                  0|1=On
VA $RN          Round Off Line Numbers
VA $RR          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           Relative Tabs
VA RX           Ratio for X Direction
VA $RX          Ratio for X Direction
VA RY           Ratio for Y Direction
VA $RY          Ratio for Y Direction
VA RZ           Record SiZe (max chars for data files)
VA $RZ          Record Separator in data file
VA SB           ??
VA $SB          Scalable Fonts available 0|1=Yes
VA SC           Superscript Footnote Numbers
VA $SC          Scan Code of last key pressed
VA SD           Separator Depth (spacing between text & note separator): 0-put
                  out current line spacing before separator [use only if you
                  have modified printer driver]; 1-use fixed (single) line
                  spacing before separator
VA $SD          Sort Directory
VA $SE          SEarch $tring last sought
VA $SF          Soft Font List File
VA SG           Assign Save/Get,$tring
VA $SG          Macro LDSGT File location
VA SH           ??
VA SI           Screen Resolution
VA $SI          Screen Resolution
VA SK           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          Sort Key
VA SL           Screen Length
VA $SL          Screen Length (Xy4); Item in list box has been selected 0|1=1
                  item (Windows)
VA SM           Show Menus
VA $SM          Show Menus: menu currently displayed 0|1=Yes
VA SN           Snaking Columns
VA SO           Sort Order Setting
VA $SO          Sort Order Setting
VA SP           ??
VA $SP          Personal Dictionary location
VA SQ           Sequential Page#: 0=respect SP command; 1=override SP and use
                  actual pages in text
VA $SQ          Sequential Page#
```

VA $SS*style#*	Style name defined in document, e.g. «VA$SS2» returns 2nd style
VA $#S	Current Style name
VA ST	Show Tabs (0|1|2|3)
VA $ST	STARTUP.INT location
VA ?ST	NBWin: Command STack
VA SW	Screen Width
VA $SW	Screen Width
VA SY	SYmbol Set
VA $SY	SYmbol Set for current font
VA SZ	SiZe: point size of font in use
VA TB	TaB Control 0=expand as spaces|1=output directly
VA $TB	TaB Character
VA $TE#	Type Effect: status of different bits set with the EF printer file setting 0|1=On
VA TF	Ignore Top Margin
VA $TF	Ignore Top Margin
VA $TM	Current TiMe
VA TO	(Time function of some sort)
VA TP	Top Margin
VA $TP	Current Cursor Position [NBWin]
VA TR	Tab Settings measured in points (72/")
VA $TR	TRiangle Mnemonic: display command embedded in delta
VA TS	Tab Settings
VA TW	?? [XyWrite]
VA TW	Text Width [NotaBene (=«VARM»)]
VA $TW	Text Window command window 0=text|1=window|2=header, footer, frame, footnote
VA TX	Triangle Suppress: 0=show|1=display contiguous deltas as one
VA $TX	Cursor Location: 0=Header|1=Text
VA $U1	U1 File
VA $U2	U2 File
VA $U3	U3 File
VA $U4	U4 File
VA $U5	U5 File (Windows)
VA $U6	U6 File (Windows)
VA $U7	U7 File (Windows)
VA $U8	U8 File (Windows)
VA $U9	U9 File (Windows)
VA UA	User Access: treatment of DeFined text 0=Xy4-DOS|1=Windows (XyWin, NBWin)
VA $UA	User Access
VA UB	??
VA]UD1	Use Main Dictionary (value of current 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	Medical
VA $UD2	Legal
VA $UD4	Computer terms
VA $UD8	Special
VA $UD16	Any Supplemental
VA UF	Use TypeFace
VA UH	Units Horizontal (default measure, e.g. IN)
VA $UH	Units Horizontal
VA UI	User Interface (Windows):
VA UI1	User Interface Command line is visible
VA UI2	User Interface 1=Status line is visible
VA UI3	User Interface 1=Button bar showing
VA UI4	User Interface 1=Format bar showing
VA UI5	User Interface 1=Ruler bar showing
VA UI6	User Interface 1=Menu bar showing
VA UI7	User Interface 1=Horizontal scroll bar showing
VA UI8	User Interface 1=Vertical scroll bar showing

```
VA UI9           User Interface 1=CMline Position
VA UI10          User Interface 1=PRompt line Position
VA UI11          User Interface 1=Button position
VA $UI           XWUIF.UIF file location (Windows)
VA UL            UnderLine Setting
VA UM            Unhide MoDe Markers
VA $UM           Unhide MoDe Markers
VA UN            UNtitled File
VA $UN           UNtitled File
Unformatted Copy Clipboard    NB.INI [Defaults]: 0|1=Text only
Unformatted Paste Clipboard   NB.INI [Defaults]: 0|1=Text only
VA $UO           Use Outline Fonts 0|1=supported
VA UP            ??
VA UR            Use Rodent 0|1=Yes
VA $UR           Use Rodent
VA $US           US-English switch: 0=native|1=English
VA UV            Units Vertical (default measure, e.g. LI)
VA $UV           Units Vertical
VA V3            ??
VA $V3           ??
VA $VE           Version Number
VA VF            Variable Forms (0|1=use multiple lines)
VA $VF           Variable Forms
VA $VI           Type of GC variable: 0=words|1=numbers|2=dates|3 =calculations
VA $VM           Vendor-Independent Messaging: 0|1=VIM-compatible electronic
                   mail system installed
VA VU            Vertical Units: min vertical movements in 1/6",screen
                   lines,decimal places in vertical formats
VA $VU           Vertical Units
VA WA            Error Message Wait Time
VA $WA           Window Availability
VA WB            Window Border Characters
VA $WB           Window Border Characters
VA $WC           Word Count from SPELL|WC|WCB command; number of CHanges made by
                   CH|CI command
VA WD            Number of lines for WiDows
VA $WE           Where-Is-Error: displays location of error "Left margin/indent
                   is greater than right margin"
VA WF            Wrap-to-Fit (0|1=keep within current borders)
VA $WF           Status of UWF command (printing mode and fonts in effect)
                   0|1|2 (XyWin, NBWin)
VA $WH           WHere: location of insertion point (after BE/wh command)
VA ?WI           NBWin: WIndow list (<VA$WN>s and d:\path\filenames)
VA $WI           WIndow Parameters: dimensions (left,top,width,height) of the
                   text window
VA WL            ??
VA $WL           ??
VA $WM           ??
VA WN            Window Handling Style: auto-renumber=0; fixed numbers=1
                   (NBWin)
VA $WN           Window Number
VA WO            Word Overstrike all except:
                   <cr>=0|...+space+tab=1|...+separators=2
VA $WO           Windows Open
VA ?WP           NBWin: Current Windows default Printername
VA $WP           Returns default printer driver, or (with args) corresponds to
                   WPROF command (Windows): count,file,appname,keyword (no args
                   returns default printer device)
?WS              windows sets
VA WS            Whole-Space Justification 0=partial (micro) spaces|1
VA $WS#          Window Status (0=no file|1=file|2=dir):
VA $WS1          Window 1 Status
VA $WS2          Window 2 Status
VA $WS3          Window 3 Status
```

```
VA $WS4          Window 4 Status
VA $WS5          Window 5 Status
VA $WS6          Window 6 Status
VA $WS7          Window 7 Status
VA $WS8          Window 8 Status
VA $WS9          Window 9 Status
VA $WV#          Windows Flag Values (Windows):
VA $WV1          Protected mode
VA $WV2          CPU 286
VA $WV4          CPU 386
VA $WV8          CPU 486
VA $WV16         Standard
VA $WV32         WIN286
VA $WV64         Enhanced
VA $WV128        WIN386
VA $WV256        CPU086
VA $WV512        CPU186
VA $WV1024       Large frame
VA $WV2048       Small frame
VA $WV4096       80x87
VA WW            Conversion Filters Path
VA $WW           Conversion Filters Path
VA WX            Windows EXception Characters: associate Speedos with TT fonts
VA $WX#          Windows Font Family: display Bitstream font used for character
                   substitution (XyWin, NBWin):
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            ??
VA $XA           ??
VA XC            Space Constant
VA $XC           Space Constant
VA XD            Read-Only Directories: 0|1=R/O
VA $XD           Read-Only Directories
VA XF            EXtract Fields from data file with SORTD
VA XI            Swap Italics (Windows)
VA XL            Selective Directory Listing
VA XM            Transpose Messages: Order on PRompt line
VA $XM           Transpose Messages
VA XN            ??
VA $XN           ??
VA XR            EXtract Records from data file with SORTD
VA XT            EXpand Triangles when cursor on delta=1
VA $XT           EXpand Triangles
VA $XW           TeXt Width
VA XY            SCRFONTS.BIN Location (default)
VA $XY           SCRFONTS.BIN Location (Xy4); Full Screen Window dimensions
                   (left,top,width,height) of application window (Windows)
VA $XZ           Status of Application Window: 0=window has been restored to
                   previous size|1=minimized|2=maximized
VA Y3            XyWrite 3+ compatibility mode=1
VA $Y3           XyWrite 3+ compatibility mode
VA YK            PRompt offset (Xy3+ only)
VA $YK           PRompt offset (Xy3+)
VA ZB            ??
VA $ZB           ??
VA ZC            Zero Capitalization 0|1=preserve case in spelling
VA $ZC           Zero Capitalization
VA ZL            LaZer printer 0|1|2|3 (set in PRN file)
VA $ZL           LaZer printer
VA ZM            ZooM percentage
VA $ZM           ZooM percentage
VA ZO            ??
```

```
VA $ZO          ??
VA ZS           Point Sizes for Scalable Fonts
VA $ZS          Point Sizes for Scalable Fonts
VA ZX           Cancel eXpanded memory=1|0=use Xmem
VA $ZX          Cancel eXpanded memory
VA $ZZ          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 23 March 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
```

Additive Mode On ([x)
Allow Hyphenation (AH)
Attribute Value ($AT)

Auto-Correct ($AC)
Auto-Pause (AP)
Auto-Replace ($AR)
Automatic Leading (AL)
Autosave Path (AOP)
Autosave Timer (AOT)
Available Modes ($AMmd)
Backspace (BS)
Backup Files (BK)
Bad Words ($BD)
Bit Status (□variablebit)
Black and White Monitor ($BW)
Black and White Trace ($BT)
Border Query ($BQbo)
Bottom Footnote (BF)
Bottom Margin (BT)
Cartridge Installed ($CN)
Cartridges ($CA)
Change Footnote Separator (CF)
Change Verification Prompt (CV)
Character Mode (MD)
Character Position (current) (@*1)
Code Page ($CP)
Columns ($CO)
Command Line ($CM)
Command Line Color Control (L1)
Contents of Macro (@#)
Contents of Summary Item (^x)
Conversion Filters Path (WW)
Correction Beep (CB)
Copied/Moved Text (last) (@*5)
Current Drive and Path ($PA)
Cursor Column Position ($CX)
Cursor Location ($TX)
Cursor Row Position ($CY)
Cursor Type (CR)
Date Format (DZ)
Debug a Program (DB)
Decimal Point (DP)
Default Drive ($DV)
Default Drive for Temporary Files (DR)
Default File Location ($DL)
Define Column ($DC)
Define End ($DN)
Define End-of-Line Character (DE)
Define Ended ($DE)
Define Start ($DS)
Define Status ($DF)
Directory Type ($DI)
Discretionary Hyphen (DH)
Display command brackets as {&reg} and {&macr} (AN) [NBWin]
Display Current Type (DT)
Display Selected Blocks (DD)
Display Type ($DT)
Display Units (DU)
Document Information ($DO)

Document Information: Summary Item (^mm)
  Author (^AU)
  Creation date (^CD)
  Comments (^CM)
  Creation time (^CT)
  Keywords (^KY)
  Last Revisor (^LG)
  Modified Date (^MD)
  Modified Time (^MT)
  Project Number (^PJ)
  Retention Period (^RP)
  Revision Number (^RV)
Dollars (£#)
Drive and Path ($PA)
Drive, Path, and Filename ($FP)
Drive, Path, Filename ($DP)
EGA Control (EC)
Eject Last Page (EJ)
Elapsed Time ($ET)
Error Beep (EB)
Error Help (EH)
Error Message (/nv \xxx)
Error Message Wait Time (WA)
Error Number ($ER)
Error Number from Macro (\@)
Error Prompt (EP)
European Punctuation (EU)
EXE Location ($ED)
Expand Triangles (XT)
Extract Fields (XF)
Extract Records (XR)
Field Separator ($FZ)
Field Separator (FX)
File Beginning ($FB)
File Date (FZ)
File End ($FE)
File Extension ($EX)
File in Directory ($DR)
File Modified ($MO)
File Status ($FS)
Filename ($FI)
Find Match (MA)
Fixed Pitch ($FX)
Flag Value for a Macro (!n)
Flush Center (FC)
Flush Left (FL)
Flush Right (FR)
Footnote Transition (FT)
Footnote Unit (FU)
Form Depth (FD)
Form Feed (FF)
Format Redlining Date (LZ)
Forms Mode ($FM)
GCI Status ($GC)
GC Variable Value ({variable)
Graphic Adapter (GA)

Graphic Path (GP)
Header Item (HI)
Header Title (HT)
Help Construction (HL)
Help File ($HL)
HP Printer File ($HP)
Hyphenation (HY)
Hyphenation Variables (HV)
Ignore End-of-File Marker (1A)
Ignore Top Margin (TF)
Image Mode Printing (IM)
Indent Paragraph (IP)
Information On/Off (IO)
Insert Cursor Type (IT)
Inside Define ($IN)
Justification (JU)
Justification Type (JT)
Justify Underline Characters (JL)
Key Click (KC)
Key Code ($KC)
Keyboard Assignment for Function Call or Text (_fc)
Keyboard File ($KB)
Keyboard Speed (KS)
LDPM Program &0 (@*248) to &9 (@*257)
LDPM Program &A (@*265) to &Z (@*290)
Left Margin (LM)
Line End Character (LC)
Line Leading (LL)
Line Number ($LN)
Line Space (LS)
Log On Notes Toggle ($LT)
Logged on User ($LG)
Long Directory Display (DI)
Macro (SG)
Macro Contains a Number (§)
Macro File ($SG)
Main Dictionary Path (LX)
Margin Unit (MU)
Maximize Windows (MW)
Memory Available ($ME)
Memory Used by XyWrite ($M+)
Menu Bar Color Control (L0)
Menu File ($MN)
Message (MG)
Metric Ruler (MR)
Microspace Factor (MS)
Military Time (MT)
Mode for Forms (MF)
Mode Number (*md)
Mode Status ($MSmd)
Mode Status ($MZmd)
Mouse Double Click (R2)
Mouse Repetition Interval (RI)
Mouse Window ($MW)
Mouse X Position ($MX)
Mouse Y Position ($MY)

Nested Command [(commandname,variable]
Network Drives (ND)
Network Login Path (NL)
New Window (NW)
No Errors from Printer (NE)
No Justification (NJ)
No Pause (NP)
No Ruler ($NR)
Non-Printable Area ($NA)
Normal Carriage Return (NC)
Not Used Printer Memory ($NU)
Numbering Style (AZ)
Offset (OF)
Outline Fonts Location(OL)
Overflow File Drive ($OV)
Overstrike Beep (OB)
Pad Spaces (PD)
Page Break Character (PX)
Page Break Color (PK)
Page Length (PL)
Page No. at Cursor ($PG)
Path No Backslash ($P\)
PC Code ($PC)
PC Code Exist ($P?)
PC Code Explanation ($PX#)
Personal Dictionary ($SP)
Point Sizes for Scalable Fonts (ZS)
Print Type (PT)
Printer File ($PR)
Printer File Selection ($PF)
Pull-down Menus Color Control (L4)
Ratio for X Direction (RX)
Ratio for Y Direction (RY)
Read Attribute ($RA)
Read Character ($RS)
Read-Only Directories (XD)
Read-Only File ($RE)
Recent Running Program (@*26)
Record Keystrokes ($RK)
Recorded Keystrokes (@*11)
Record Separator ($RZ)
Record Separator (RS)
Redline Data (RD)
Redlining ($RL)
Relative Tabs (RT)
Resume Code ($RC)
Return Error from DOS ($RR)
Return File Text from DOS (=filename,string=)
Reverse Buttons (RB)
Right Margin (RM)
Round Off Numbers (RN)
Ruler Line Color Control (L3)
Ruler Markers (RL)
SaveStyles($SSn)
Scan Code ($SC)
Screen Length (SL)

Screen Resolution (SI)
Screen Width (SW)
SCRFONTS.BIN Location (XY)
Search String ($SE)
Selective Listing (XL)
Sequential Page No. (SQ)
Show Menus ($SM)
Show Menus (SM)
Show Tabs (ST)
Size of Macro (¦*n*)
Snaking Columns (SN)
Soft Font List File ($SF)
Sort (SO)
Sort Directory ($SD)
Sort Key (SK)
Space Constant (XC)
Space Factor (XF)
Spelling Checker (CK)
Startup ($ST)
Status Line Color Control (L2)
Superscript Numbers (SC)
Tab Character (TB)
Tab Set (TS)
Text Window ($TW)
Three-Dimensional Effects (3D) (Windows)
Top Margin (TP)
Transpose Messages (XM)
Triangle Mnemonic ($TR)
Truncated Path Name ($P.)
Truncated Path Name (P.)
Type Effect ($TEn)
Underline Setting (UL)
Unhide Markers (UM)
Unit of Measure Conversion (<u1#u2)
Units Horizontal (UH)
Untitled File (UN)
Use Dictionary ($UDn)
Use Dictionary (UD)
Use Horizontal (UH)
Use Outline ($UO)
Use Pointer (UR)
Use Vertical (UV)
Variable Forms (VF)
Version Number ($VE)
Vertical Unit (VU)
Whole-Space Just. (WS)
Window Availability ($WA)
Window Border Characters (WB)
Window Border Colors (BX)
Window Number ($WN)
Window Status ($WS)
Windows Open ($WO)
Word Count ($WC)
Word Overstrike (WO)
Zero Capitalization (ZC)

APPENDIX D • FROM BASIC TO XPL

The purpose of this appendix is to help someone who knows the BASIC programming language pick up XyWrite Programming Language (XPL) quickly and easily. It compares the two languages by taking the major features of XPL and describing their equivalents in BASIC.

VARIABLES

A macro in XPL is equivalent to a variable in BASIC.

BASIC: Variables can be given names.
XPL: Macro names are restricted to the letters A-Z and numbers 0-9 and 000-1999 — for a total of 2036 macros.

UNASSIGNED VARIABLES

BASIC: If you use a variable before you assign a value to it, a numeric variable is given the value zero; a string variable is given the value null.
XPL: In XPL, a macro does not exist until you assign a value to it. Thus, using a macro before you assign a value to it in many cases will produce an error (for example, PV inside an IF statement or GT outside an IF statement).

EXECUTION

In BASIC, each statement takes one line; in XPL, statements are all strung together.

BASIC: You normally follow each statement with a carriage return, so that each statement is on its own line.
XPL: You string all statements together, one after the next (word-wrapping to the next line). In general, carriage returns are not allowed after statements. In fact, any character (including a carriage return) which appears outside double-angle brackets is inserted at the cursor location in the text or the header when the program is run. The only place you can put a carriage return without inserting it into the text is after «GL» (Go to Label), after «EX» (Exit), inside a label «LB», or at the end of a command (as a substitute for **`XC`** , to execute a command).

STRING VARIABLE

BASIC:	`A$="Yes"`
XPL:	`«SV01,Yes»`
Comment:	Assigns the string Yes to macro 01

NUMERIC VARIABLE

BASIC:	`A=20`
XPL:	`«SX01,20»`
Comment:	Assigns a value of 20 to macro 01.

EMPTYING A VARIABLE

BASIC:	`A$=""`
XPL:	`«SV01,»`
Comment:	Empties or initializes macro 01, so that its content is null.

NUMERIC EXPRESSION

BASIC:	`A=(X+Y)/2`
XPL:	`«SX01,(«PV02»+«PV03»)/2»`
Comment:	Averages the values in macros 02 snf 03, and assigns the result to macro 01.

STRING CONCATENATION

BASIC:	`A$=B$+C$`
XPL:	`«SX01,«IS02»+«IS03»»`
Comment:	Joins the string in macro 02 to the string in macro 03 and assigns the result to macro 01.

COMMENTS

BASIC:	`REM comment`
XPL:	`;*; or «LBcomment»`
Comment:	The comment can be any length string.

LABEL STATEMENT

BASIC:	`Test: statement`
XPL:	`«LBTest» statement`
Comment:	Defines a label named *Test*.

GO TO A LABEL

BASIC: `GOTO Test`
XPL: `«GLTest»`
Comment: Goes to the label named *Test*.

IF STATEMENT (GENERAL STATEMENT)

BASIC: `IF expr THEN truebranch ELSE falsebranch`
XPL: `«IFexpr» truebranch «EI» ...continues`
Comment: (See the examples below.) In XPL, the End If «EI» command is required, even if there is no falsebranch. Notice this other difference: In BASIC, when expr is TRUE, execution continues through the truebranch, hits ELSE and skips over the falsebranch. However, in XPL, you must explicitly put a Go to Label («GL») command before the «EI» to prevent instructions after the «EI» from being executed — the «GL» causes execution to jump to a label. As in BASIC, you may directly nest IF statements in XPL.

IF STATEMENT (COMPARING NUMBERS)

BASIC: `IF A=20 THEN GOTO A ELSE GOTO B`
XPL: `«IF«PV01»==20»«GLA»«EI»«GLB»`
Comment: Notice that in XPL, two equal signs (==) are used to indicate "is equal to."

IF STATEMENT (COMPARING STRINGS)

BASIC: `IF A$="Yes" THEN GOTO A ELSE GOTO B`
XPL: `«IF«IS01»==«IS02»»«GLA»«EI»«GLB»`
Comment: Notice we use «IS01» and «IS02» here when comparing strings, rather than the «PV01» used above for comparing numbers.

IF STATEMENT WITH AND

BASIC: `IF A>0 AND A<10 THEN GOTO TEST`
XPL: `«IF(«PV01»>0)&(«PV01»<10)»«GLTest»«EI»`
Comment: Similarly, use the exclamation point (!) for logical OR.

IF STATEMENT WITH NOT

BASIC: `IF NOT (A>B)`
XPL: `«IF@NOT(«PV01»><PV02»)»`
Comment: The XPL code reads: "If it's not true that macro 01 is greater than macro 02."

SUBROUTINE

BASIC: `TEST: X=X+1 RETURN`
XPL: `«SU01,«SX02,«PV02»+1»»`
Comment: Define subroutine 01 «SU01» as incrementing macro 02 by 1.

GO TO SUBROUTINE

BASIC: `GOSUBTEST`
XPL: `«GT01»`
Comment: «GT01» runs the subroutine. In XPL, the subroutine «SU01» must be defined ahead of the «GT01» statement.

DISPLAY DATA ON THE SCREEN

BASIC: `PRINT "x=";x`
XPL: `x=«PV02»`
Comment: In XPL, any text that is not inside double-angle brackets is automatically displayed on the screen at the cursor location.

PAUSE FOR KEYSTROKE

BASIC: `PRINT "OK to continue? (Y/N)" A$=INKEY$`
XPL: `«PROK to continue? (Y/N)»«SX01,«RC»»`
Comment: The Prompt command «PRcomment» displays the comment on the status line. Execution stops when it hits the Read Character «RC» command and does not continue until a key is pressed.

LENGTH OF STRING

BASIC: `B$=LEN(A$)`
XPL: `«SX02,@SIZ(«IS01»)»`
Comment: Find the number of characters in macro 01, and put the result in macro 02. If macro 01 has never been assigned a value (and thus, does not yet exist), XyWrite displays an error and the program terminates.

TEST FOR ERROR

BASIC: `ON ERROR GOTO A ELSE GOTO B`
XPL: `«IF«ER»»«GLA»«EL»«GLB»`
Comment: If an error occurs («ER» is true), then go to label A. Alternatively, you can test to see if no error has occurred: `«IF@NOT(«ER»)»«GLA»«EI»«GLB»`

RETURNING CHARACTERS

In XPL, the XS (Extract String) command is similar to the BASIC LEFT$, MID$, or RIGHT$ functions. Refer to the *IBM Basic Handbook* for information.

INDEX

XPL PROGRAMMING EXAMPLES

Included in the present XPL.ZIP file, from the original Guide:
CLEAN.PGM
DRIVE.PGM
INPUT.PGM
LINE.PGM
MERGE.PGM (uses DATA3)
NUMBER.PGM (uses TELELIST)
SPACE.PGM
TERMINAT.PGM

Added in the present XPL.ZIP file, for printing, non-expanded viewing, and NBWin:
CONVERT.PM
XY4.KBD

RJH LastRev.1/27/2006