WANG

2200

Programmer's Guide to Word Processing

2200 Programmer's Guide to Word Processing

1st Edition — August, 1982

700-6961

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL. (617) 459-5000, TWX 710-343-6769, Telex 94-7421

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

PREFACE

This manual describes programming tools available to purchasers of Wang 2200 Word Processing software. Before using the tools described in this manual the reader should become familiar with 2200 Word Processing by scanning the <u>2200 Word Processing Operator's Guide</u> (700-6937A). While becoming familiar with 2200 Word Processing in general, the reader should pay particular attention to the following: the structure of 2200 Word Processing documents, especially the use of page breaks and format lines, the Document Summary, the Print Document Menu, glossaries, and the Wang 2200 Word Processing menus.

Chapter 1 of this manual describes the functions that can be performed using the Document Access subroutines. The subroutines allow the programmer to access, modify, and create 2200 Word Processing documents from executing BASIC-2 programs. The final section of Chapter 1 is a reference guide for the subroutine user.

Chapter 2 describes the steps a programmer takes to create a menu entry on one of the 2200 Word Processing menus.

Chapter 3 describes the steps a programmer takes to access the 2200 Word Processing Software system from an application program.

CONTENTS

CONTENTS (continued)

FIGURES

TABLES

CHAPTER 1
DOCUMENT ACCESS SUBROUTINES

1.1 INTRODUCTION

The 2200 Word Processing Document Access Subroutines are BASIC-2 subroutines that allow the programmer to access, modify, and create 2200 Word Processing documents from executing BASIC-2 programs. With these subroutines a programmer can perform the following functions.

- Read specific information from documents
- Rewrite document text
- Modify existing documents (i.e. pages, formats, etc.)
- Create new documents
- Read document summary information from documents
- Modify document summary information
- Search a document for a specified text string
- Recall glossary text in a document

Each subroutine execution accomplishes one function in a chain of functions necessary to perform an application determined by the programmer. This chapter will demonstrate subroutine operation by creating some simplified sample programs. The sample programs are designed only to give the programmer a basic understanding of how the subroutines operate. The sophistication of the applications depends on the creativity and knowledge of the individual programmer.

The Document Access subroutines are marked subroutines (GOSUB') that allow the programmer to perform specific operations with 2200 Word Processing documents. Word processing document access is accomplished on a page oriented basis. In simplest terms, the subroutines allow the user to access documents by locating pages, adding pages, deleting pages, or replacing pages. To successfully use the Document Access subroutines, the programmer must be familiar with the structure of a 2200 Word Processing document, as described in the following section.

1.2 STRUCTURE OF A 2200 WORD PROCESSING DOCUMENT

2200 Word Processing documents contain administrative and text information. Each 2200 Word Processing document is saved on disk in a BASIC-2 file called a "volume". Each volume may contain one or many documents organized in one or many libraries. A detailed description of the kind of information contained in each document follows.

1.2.1 Administrative Information

Administrative information consists of Document Summary information, statistical information, and Print Document menu information. The Document Summary information is composed of alphanumeric and numeric fields that contain information useful in document file maintenance. The summary allows the user to enter title, author, and operator; and provides space for comments (such as a brief description of the document or keyword information to aid the user in identifying the document). Each time the document is accessed by means of word processing, the document summary screen appears with the previously entered information. The user may retain the summary information, or change specific fields.

Statistical information appears below the summary information on the screen. The system maintains statistical information, and updates it every time a document is accessed by means of word processing. The word processing user is unable to alter any of the information in these fields. However, the Document Access subroutines allow the programmer to manipulate these data fields, accessing and changing the information they contain.

Print menu information consists of default values presented by the Print Document menu when it is viewed. When a selected document is printed, the values supplied by the user in the Print Document menu become the default values.

1.2.2 Text Information

Pages of text information comprise the main body of the document. All 2200 Word Processing document pages consist of the following three parts.

- A format line that begins the page of text
- Text area that may include additional format lines
- A page break that ends each page of text, excluding the last page

When a word processing document is viewed on the screen, special graphic characters appear within the document pages. The graphic characters represent special keys pressed by the operator when the document was created. Every word processing document is stored on disk as a document file. The document file contains hex codes to represent both text characters and special graphic characters. The hex codes are stored in a continuous string of data, regardless of how they appear on screen, or when output to a printer. The following rules apply to a word processing document stored on disk.

Spaces created by pressing special keys (such as the TAB, RETURN, or INDENT key) are not stored in the disk form of the document. Only the hex code for the graphic character is stored. Spaces created in this manner appear as unoccupied space when the document is viewed on screen.

Spaces typed within the text of a document are stored as HEX(80) in the disk form of the document. Spaces created in this manner are displayed as a small dot when the document is viewed on screen.

All unused characters from the last text character to the end of the page are stored on disk as HEX(20)s.

The last text character of all document pages, except the last page, must be a page break character (HEX(8F)).

The first format line of a page begins with a First Format Line Indicator (HEX(8F)). All other format lines contained in a page of text begin with a HEX(86) which indicates a format line other than the first format line of a page. Each format line begins with a format line indicator and a vertical spacing indicator, followed by a user-defined sequence of spaces and tabs. A return character must end the format line.

A text page should not contain adjacent format lines.

Table 1-1 lists word processing graphic characters, their meaning, and their BASIC-2 hex codes. Figure 1-1 shows the word processing character set hex codes for the 2236DW terminal. The word processing character set, an alternate character set for the 2236DW terminal, is automatically activated when WANG 2200 Word Processing software is executed.

The word processing character set is not in use when a word processing document is viewed on a 2236DE terminal. In this case, the hex codes and their meaning remain as shown in Table 1-1, but a different graphic symbol is displayed.

The programmer can control the appearance of a word processing document with the hex codes shown in the following table. Refer to the <u>2200 Word Processing Operator's Guide</u> for a description of the word processing functions performed when the following hex codes are used in a document.

Table 1-1. Hex Codes for 2236DW Terminal Graphic Characters

Hex Code	Graphic	Meaning
HEX(5C)*	\	Required Space
HEX(5E)*	↑	Superscript
HEX(5F)*	↓	Subscript
HEX(7F)	↕	Merge
HEX(81)	◆	Center
HEX(82)	▶	Tab
HEX(83)	◀	Return
HEX(84)	→	Indent
HEX(85)	└	Dec Tab
HEX(8B)	■	Stop
HEX(8C)	!!	Note
HEX(8F)	⊥	Page Break or First Format Line Indicator
HEX(86)	|	Format Line Indicator for Format Lines within Text
HEX(FF)	↨	Don't Merge

* Starred HEX codes may be different in some non-English implementations of the system.

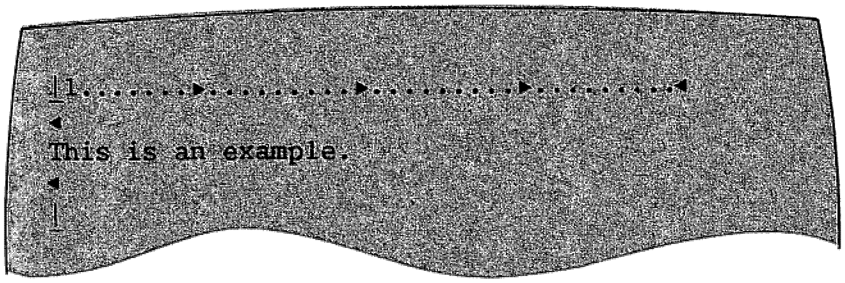
High-order HEX Digit

Low-order HEX Digit

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	â	Space	0	@	P	°	p	Space	â̲	_	0̲	@̲	P̲	°̲	p̲
1	ê	!	1	A	Q	a	q	♦	ê̲	!̲	1̲	A̲	Q̲	a̲	q̲
2	î	"	2	B	R	b	r	▶	î̲	"̲	2̲	B̲	R̲	b̲	r̲
3	ô	#	3	C	S	c	s	◀	ô̲	#̲	3̲	C̲	S̲	c̲	s̲
4	û	$	4	D	T	d	t	→	û̲	$̲	4̲	D̲	T̲	d̲	t̲
5	ä	%	5	E	U	e	u	└─	ä̲	%̲	5̲	E̲	U̲	e̲	u̲
6	ë	&	6	F	V	f	v	|	ë̲	&̲	6̲	F̲	V̲	f̲	v̲
7	ï	'	7	G	W	g	w	••	ï̲	'̲	7̲	G̲	W̲	g̲	w̲
8	ö	(	8	H	X	h	x	ʼ	ö̲	(̲	8̲	H̲	X̲	h̲	x̲
9	ü	)	9	I	Y	i	y	ʻ	ü̲	)̲	9̲	I̲	Y̲	i̲	y̲
A	à	*	:	J	Z	j	z	^	à̲	*̲	:̲	J̲	Z̲	j̲	z̲
B	è	+	;	K	[	k	§	■	è̲	+̲	;̲	K̲	[̲	k̲	§̲
C	ù	,	<	L	\	l	£	!!	ù̲	,̲	≤	L̲	\̲	l̲	£̲
D	Ä	-	=	M	]	m	é	↕	Ä̲	-̲	=̲	M̲	]̲	m̲	é̲
E	Ö	•	>	N	↑	n	ç	ß	Ö̲	•̲	≥	N̲	↑̲	n̲	ç̲
F	Ü	/	?	O	↓	o	↕	⊥	Ü̲	/̲	?̲	O̲	↓̲	o̲	↕̲

Figure 1-1. Word Processing Character Set for the 2236DW Terminal

Figure 1-2 is a sample page of a 2200 Word Processing document as it appears on the screen of a 2236DW terminal. The page consists of a format line that begins with a format line indicator, HEX(8F), and ends with a return, HEX(83). Text and special graphic characters follow the format line. The last character of the page is a page break, HEX(8F). Figure 1-3 is the same page as it appears when stored on disk.



```
‡1.......▶.........▶.........▶..........◀
◀
This is an example.
◀
‡
```

Figure 1-2. Appearance of a Document Page on the 2236DW Terminal Screen

```
Format Line Begins
       |
       ▼
HEX(8F318080808080808082808080808080808082808080808080808080828080808080808080
8083835468697380697380616E806578616D706C652E83838F)
  ▲                                                   ▲
  |                                                   |
Format Line Ends                               Page Break
```

Figure 1-3. Appearance of a Page (Figure 1-2) when Stored on Disk

1.3 DOCUMENT ACCESS SUBROUTINE FUNCTIONS

Table 1-2 lists the subroutine function performed by each of the Document Access subroutines, and the DEFFN' assignment for each subroutine.

Table 1-2. Document Access Subroutines

Subroutine Function	DEFFN' Assignment
Initialize Data	205
Open Document	206
Close Document	207
Create Document	208
Delete Document	209
Change Password	079
Go To Page	245
Read Page	247
Rewrite Page	248
Insert Page	249
Delete Page	251
Append Page	252
Search Text	253
Read Administrative Information	218
Write Administrative Information	219
Attach Glossary	229
Call Glossary	230
Detach Glossary	233

1.4 DESIGNING A DOCUMENT ACCESS APPLICATION

The Document Access subroutines provide the programmer with the tools necessary to manipulate a word processing document from an executing BASIC-2 program. The approach used to produce results can vary from programmer to programmer. However, the following discussion can serve as a guide for programmers designing an application.

An application program can access up to four open documents at a time. The user passes parameters to the subroutines that assign a slot number (1 to 4) to each open document. The subroutines distinguish between the open documents based on the user-assigned slot numbers, not on the word processing document ID.

Figure 1-4 shows the order in which the 2200 Document Access subroutines are used to successfully access documents.

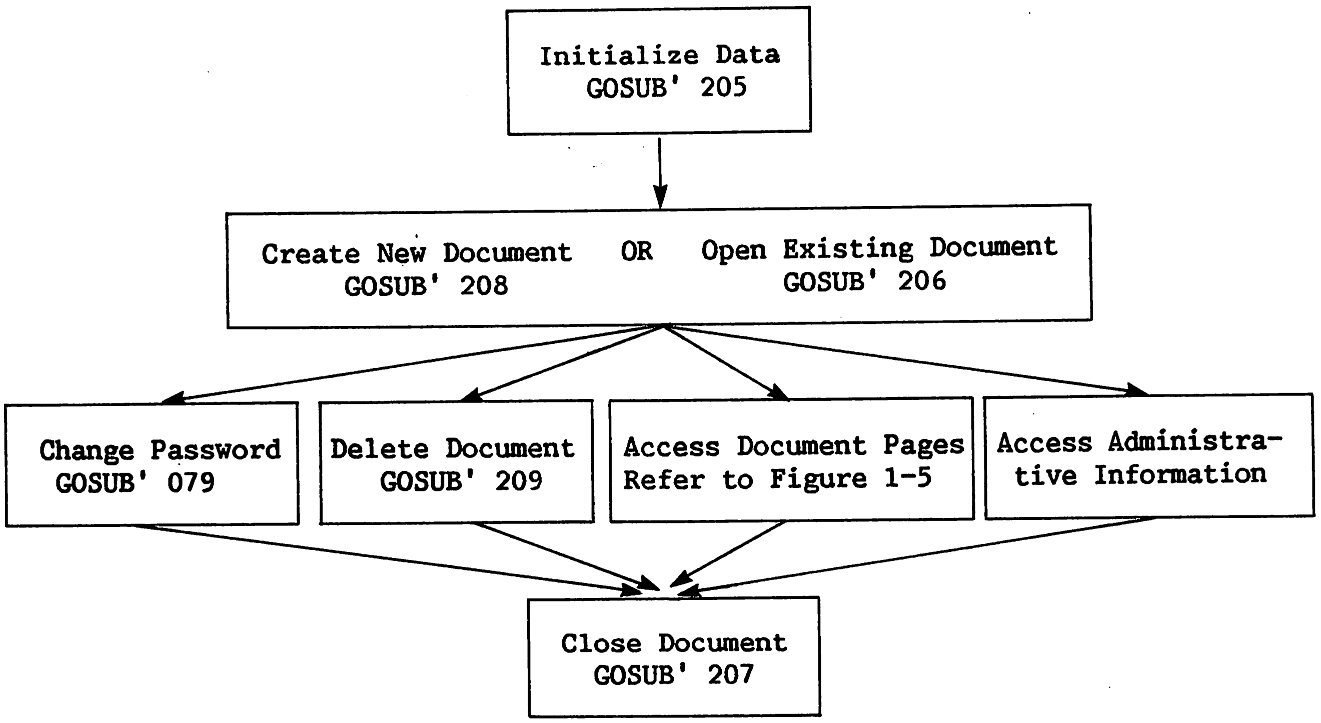


Figure 1-4. Subroutines Used to Successfully Access Documents

Figure 1-5 shows the Document Access subroutines used to access pages of a document after it is successfully opened, as shown in Figure 1-4.

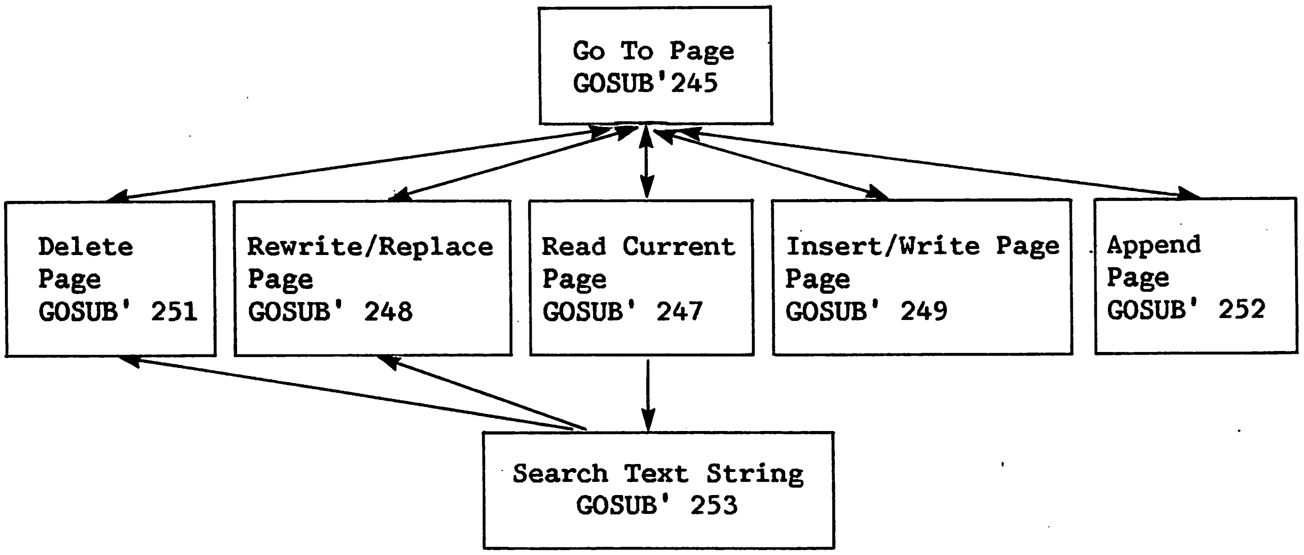


Figure 1-5. Page Accessing Subroutines

1.4.1 Identify the Application Goals

To design a document access application, the programmer first identifies the goals of the application. For example, goals for a document access application could be: to delete a document, to create a new document, to add text to a document, to get administrative information from a document.

After the application is designed, the application program is coded to achieve the design goals. Subroutine calls to access documents, and the order in which they are used, are shown in Figures 1-4 and 1-5. The following subsections list the subroutine calls needed to achieve example design goals.

Delete a Document

To delete a document the application program must contain the following subroutine calls, in order.

Call the Initialize Data subroutine with a GOSUB' 205.
Call the Open Existing Document subroutine with a GOSUB' 206.
Call the Delete Document subroutine with a GOSUB'209.

Create a Document with No Text

To create a document containing no text but containing administrative information, the application program must contain the following subroutine calls, in order.

Call the Initialize Data subroutine with a GOSUB' 205.
Call the Create New Document subroutine with a GOSUB' 208.
Call the Write Administrative Information subroutine with a GOSUB' 219.
Call the Close Document subroutine with a GOSUB' 207.

Get Administrative Information from an Existing Document

To retrieve administrative information from an existing document, the application program must contain the following subroutine calls, in order.

Call the Initialize Data subroutine with a GOSUB' 205.
Call the Open Existing Document subroutine with a GOSUB' 206.
Call the Read Administrative Information subroutine with a GOSUB'218.
Call the Close Document subroutine with a GOSUB' 207.

Create a Document with Text

To create a document containing text, the application program must contain the following subroutine calls, in order.

Call the Initialize Data subroutine with a GOSUB' 205.
Call the Create New Document subroutine with a GOSUB' 208.
Call the Go To Page subroutine with a GOSUB' 245.
Call the Insert New Page subroutine with a GOSUB' 249.
Call the Write Administrative Information subroutine with a GOSUB' 219.
Call the Close Document subroutine with a GOSUB' 207.

The subroutines use, as a parameter, the variable VO$() that holds the contents of a document page. The variable is, in effect, a user-created buffer area for the page contents. The buffer established in this manner must represent a valid 2200 Word Processing document page, as described in Section 1.2.

The following are three principal ways to create a page buffer to be placed in a word processing document through the Document Access subroutines. It is up to the programmer to decide what method, or combination of methods, is preferred.

1. Create a page buffer by establishing a variable that consists of concatenated literals, alpha variables, and hex codes.

2. Create a page buffer by modifying a page read from an existing document. For example, the page modification can consist of any, or all, of the existing page concatenated with data from a data file.

3. Create a page buffer that consists of recalled glossary text. The three Document Access subroutines used to place glossary text in the page buffer are described in the following subsection.

Glossary Text

To recall glossary text and place the text into the page buffer, the application program should contain the following subroutine calls, in order.

Call the Attach Glossary subroutine with a GOSUB' 229.
Call the Call Glossary subroutine with a GOSUB' 230.
Call the Detach Glossary subroutine with a GOSUB' 233.

Glossary text recalled through the use of the Document Access subroutines is first created by means of 2200 Word Processing. Refer to the 2200 Word Processing Operator's Guide (700-6937) for a description of glossaries and directions on how to create glossaries.

1.5 LOADING THE SUBROUTINES

In 2200 Word Processing Software, Wang Laboratories, Inc. includes modules that contain the Document Access subroutines and the BASIC-2 program "DAST.DAT". The DAST.DAT program is known as the Software Selection Tool.

An application can load the document access subroutines by two different methods: the subroutines can be loaded directly, or the subroutines can be loaded indirectly by using DAST.DAT, the Software Selection Tool. An advantage gained by the direct method can be reduced memory use and improved program performance. However, a disadvantage is that program maintenance is less convenient. Applications programs are not insulated from changes made to the Document Access subroutine modules, or module names. An advantage of the Software Selection Tool method is that the application only supplies the prime numbers of the subroutine calls; all the module names required for the calls are supplied through the Software Selection Tool. The remainder of this section describes both methods of overlaying the subroutines.

NOTE

Line numbers in the range 4000 to 5500 are reserved for the program overlays. The user's application program must not contain statements with line numbers in that range. The common variables required by the overlays are reserved variables and can not be used as variables in the application program.

1.5.1 Direct Method

To load the document access subroutines directly, the programmer must know the number of modules and the module names needed for each subroutine call. Appendix A contains a list of each subroutine function, assignment number, number of modules, and module names needed for each subroutine.

The 2200 Word Processing Software system uses the following variable assignment conventions. When directly loading the subroutine modules, the user application can follow these conventions or choose to use other variable assignments. In this manual, examples and directions follow the conventions.

I9(4) = number of modules required to perform the subroutine

I9$() = alpha variable specifying sequentially the names of the modules to load

To load the Document Access subroutines directly, the programmer must take the following steps. For the example disk access statements, it is assumed that all software is stored on the device in Slot 0.

Step 1

The application program must declare common variables required for the Document Access subroutine module overlays. The following COM statement must appear in the application program.

```
COM  A6$6, B1$1, R1$1, U0$3, U3$1, U4$4, V5$(160)1, U0(23), U5(10),
V6(9), V9(9), V5(9), A1(2,4), A3(4), A9(4), A7(4), B7(4), B2(4),
U1$(256)1, V1$(3)82, V4$1, U2$(64)1, U3(9), U0, V5, U6, U7, V9, U8, U4,
V0, V2
```

For user convenience, COM statements required for the Document Access subroutines are provided in the program "DAST.COM". When the programmer is keying in the application program, the LOAD command (LOAD DCT#0, "DAST.COM") can be issued to load the COM statements into memory. After the LOAD command is executed, the programmer can edit the COM statements and enter the remaining statements of the application program. The last COM statement provided in DAST.COM is not needed when using this method.

Step 2

To allow the user to reduce the amount of memory reserved for COMmon variables, the user determines the memory space allocated by a COM statement for the arrays V0$(), B1$(), B2$(), B3$(), and B4$().

The array V0$() is used in the Document Access subroutines to contain a document page. The maximum allowable size for a document page, and therefore the maximum COMmon size for V0$() is V0$(4181)1. To reduce the memory space allocated to V0$(), the COMmon size of V0$() can be decreased to equal the number of characters contained in the largest page accessed or created by the subroutines. The count of the number of characters in the largest page must include all format line characters, text characters, and page breaks, if any.

The COMmoned size for the B1$(), B2$(), B3$(), and B4$() arrays depends on how many documents the application program opens at any one time (maximum is four), and the slot numbers the user assigns to the documents. The following list shows two forms of the COM statement for each of the arrays. The form of the COM statement used in the applications program depends on whether or not the listed slot number is in use in the application program.

Slot Number	Slot Number In Use	Slot Number Not In Use
1	COM B1$(123)2	COM B1$(1)1
2	COM B2$(123)2	COM B2$(1)1
3	COM B3$(123)2	COM B3$(1)1
4	COM B4$(123)2	COM B4$(1)1

For example, an application program assigns the maximum memory allocation for page size, and opens four documents simultaneously. This is the optimum case, requiring the maximum amount of memory space allocated for the arrays. The following COM statement is required in the application program.

```
COM V0$(4181)1, B1$(123)2, B2$(123)2, B3$(123)2, B4$(123)2
```

Step 3

Each subroutine function is accomplished by loading several modules supplied with 2200 Word Processing Software. Refer to Appendix A for a list of modules necessary to accomplish each subroutine call. Since each subroutine requires that several modules are loaded, the multiple module form of the LOAD statement is used in Step 4. The application program must dimension a scratch variable that represents the names of the modules to load.

By 2200 Word Processing Software system convention, I9$() represents an alpha variable that specifies sequentially the names of the modules to load. The DIM statement for I9$() that must appear in the application is determined by the maximum number of modules that are loaded at any one time. For example, the maximum number of modules needed to support the Open Document subroutine call is 7. Since 14 is the maximum number of modules required to support a single subroutine call, the following DIM statement can be used in all applications that are designed to overlay the Document Access subroutines as they are required by application logic.

```
DIM I9$(14)8
```

If the statement in step 4 is used to load all subroutines required by program logic, the dimension for I9$() must equal the number of modules to load. The length remains 8.

Step 4

The application program must contain the following LOAD statement for each subroutine overlay. The statement loads into memory the desired Document Access subroutine modules. Line numbers in the range 4050 to 5500 are reserved for this overlay.

```
LOAD DCT #0, <I9(4)> I9$() 4050,5500 BEG [begin-line-number]
```

I9(4) = number of modules required to perform the subroutine or subroutines

I9$() = alpha variable specifying sequentially the names of the modules to load

begin-line-number = The line number of the application program where execution is to begin after the overlay is loaded into memory.

Step 5

The application program must contain calls (GOSUB') to the desired Document Access subroutines. Refer to Section 1.6 for the general form of the subroutine calls.

1.5.2 Software Selection Tool Method

When the Software Selection Tool, DAST.DAT, is used as a segment of the application program, it provides data statements needed to load the Document Access subroutines listed in Table 1-2. To use this method, the application program must overlay DAST.DAT. The DAST.DAT overlay supplies a marked subroutine (DEFFN'216) that accepts the prime numbers of the desired Document Access subroutines and returns the total number and names of the modules associated with the desired subroutines. The information returned from the DAST.DAT overlay can, in turn, be used to overlay the subroutine modules.

The programmer must take the following steps to ensure that the application program can successfully perform the DAST.DAT and Document Access subroutine module overlays. The application program must contain the following statements in the order presented. For the following disk access statements, it is assumed that all software is stored on the device in Slot 0.

Step 1

The application program must declare common variables required for the DAST.DAT overlay, and Document Access subroutine module overlays. The following COM statement must appear in the application program.

```
COM R9$(44)8, A6$6, B1$1, R1$1, U0$3, U3$1, U4$4, V5$(160)1, U0(23),
U5(10), V6(9), V9(9), V5(9), A1(2,4), A3(4), A9(4), A7(4), B7(4), B2(4),
U1$(256)1, V1$(3)82, V4$1, U2$(64)1, U3(9), U0, V5, U6, U7, V9, U8, U4,
V0, V2
```

For user convenience, the first COM statement required for the Document Access subroutines is provided in the program "DAST.COM". Before the programmer enters the application program, the LOAD command (LOAD DCT#0, "DAST.COM") can be issued to load the first COM statement into memory. After the LOAD command is executed, the programmer can enter the remaining statements of the application program. The statement number of the first COM statement can be changed to suit the application program.

Step 2

To allow the user to reduce the amount of memory reserved for COMmon variables, the user determines the memory space allocated by a COM statement for the arrays V0$(), B1$(), B2$(), B3$(), and B4$().

The array V0$() is used in the Document Access subroutines to contain a document page. The maximum allowable size for a document page, and therefore the maximum COMmon size for V0$() is V0$(4181)1. To reduce the memory space allocated to V0$(), the COMmon size of V0$() can be decreased to equal the number of characters contained in the largest page accessed or created by the subroutines. The count of the number of characters in the largest page must include all format line characters, text characters, and page breaks, if any.

The COMmoned size for the B1$(), B2$(), B3$(), and B4$() arrays depends on how many documents the application program opens at any one time (maximum is four), and the slot numbers the user assigns to the documents. The following list shows two forms of the COM statement for each of the arrays. The form of the COM statement used in the applications program depends on whether or not the listed slot number is in use in the application program.

Slot Number	Slot Number In Use	Slot Number Not In Use
1	COM B1$(123)2	COM B1$(1)1
2	COM B2$(123)2	COM B2$(1)1
3	COM B3$(123)2	COM B3$(1)1
4	COM B4$(123)2	COM B4$(1)1

For example, an application program assigns the maximum memory allocation for page size, and opens four documents simultaneously. This is the optimum case, requiring the maximum amount of memory space allocated for the arrays. The following COM statement is required in the application program.

```
COM V0$(4181)1, B1$(123)2, B2$(123)2, B3$(123)2, B4$(123)2
```

Step 3

The application program must dimension scratch variables required by DAST.DAT. The following DIM statement must appear in the application program.

```
DIM R9, R9$1, I9, I9$54, I9(5), I9$(14)8, I8$3
```

Step 4

The application program must contain the following LOAD statement to effect the DAST.DAT overlay. Line numbers in the range 4000 to 4050 are reserved for overlay.

```
LOAD DCT#0, "DAST.DAT" 4000,4050 BEG [begin-line-number]
```

begin-line-number = The line number of the application program where execution is to begin after the overlay is loaded into memory.

Step 5

The application program must contain a statement to call the marked subroutine (DEFFN '216) supplied by DAST.DAT. The subroutine returns the number and name of the Document Access subroutine modules (OUTPUT) associated with a user-supplied count and list of Document Access subroutine assignment numbers (INPUT). The returned information is required to overlay the Document Access subroutine modules as described in Step 7.

The general form of the call is as follows.

```
GOSUB '216 (I9, I9$)
```

INPUT: I9 = A count of the number of Document Access subroutines required by the application program.

I9$ = An alphanumeric variable consisting of the concatenation of the three digit DEFFN' assignment numbers (refer to Table 1-2) for the subroutines required by the application program. The DEFFN' assignement numbers may be listed in any order, but all assignment numbers must consist of three digits. In the case of an assignment number less than 100, leading zeros must be used in order to supply three digits.

OUTPUT: R9$ = Error return code.
= HEX(00) for a successful call.
= HEX(01) if an input DEFFN' assignment number is not found.
= HEX(02) if there are too many module names for the dimensioned size of R9$().

R9 = The number of Document Access subroutine modules required by the input subroutine numbers. This return variable is used in the LOAD statement described in Step 7.

R9$() = An array containing the list of module names required for the input subroutine numbers. This return variable is used in the LOAD statement described in Step 7.

An example of this statement is shown in program line 120 in the example application section.

Step 6

The application program should contain the following statement to check the error code returned from the subroutine call described in Step 5.

```
IF R9$ <> HEX(00) THEN [line-number]
```

line-number = Line number for the first statement in an error handling routine contained in the application program.

Step 7

The application program must contain the following LOAD statement. The statement loads into memory the desired Document Access subroutine modules using the return information from Step 5. Line numbers in the range 4050 to 5500 are reserved for this overlay.

```
LOAD DCT #0, <R9> R9$() 4050,5500 BEG [begin-line-number]
```

begin-line-number = The line number of the application program where execution is to begin after the overlay is loaded into memory.

Step 8

The application program must contain calls (GOSUB') to the desired Document Access subroutines. Refer to Section 1.7 for the general form for the subroutine calls.

Example Application

The application programmer determines that the following Document Access subroutines are required to append a page contained in variable V0$() to document 0015A. The application program opens one document and assigns that document to Slot 1.

Subroutine	DEFFN'
Initialize Data	205
Open Document	206
Go To Page	245
Append Page	252
Close Document	207

The following statements, contained in the application program, allow the program to make calls to the four Document Access subroutines previously listed. Refer to Section 1.6 for details regarding the general form of the Document Access subroutine calls.

```
0010 REM STEPS 1 AND 2, DECLARE COMMON VARIABLES
0020 COM  R9$(44)8,  B1$1,  R1$1,  A6$6,  U0$3,  U3$1,  U4$4,  V5$(160)1,
     U0(23),  U5(10),  V6(9), V9(9), V5(9), A1(2,4), A3(4), A9(4), A7(4),
     B7(4), B2(4), U1$(256)1, V1$(3)82, V4$1, U2$(64)1, U3(9)
0030 COM U0, V5, U6, U7, V9, U8, U4, V0, V2
0040 REM MAX PAGE SIZE, ONE OPEN DOCUMENT ASSIGNED TO SLOT NUMBER 1
0050 COM V0$(4181)1, B1$(123)2, B2$(1)1, B3$(1)1, B4$(1)1

0060 REM STEP 3, DIMENSION SCRATCH VARIABLES
0070 DIM R9, R9$1, I9, I9$54, I9(5), I9$(10)8, I8$3

0080 REM STEP 4, LOAD THE DAST.DAT OVERLAY
0090 LOAD DCT#0, "DAST.DAT" 4000,4050, BEG 120

0100 REM STEP 5, BRANCH TO MARKED SUBROUTINE SUPPLIED BY DAST.DAT
0110 REM PROGRAM REQUIRES FIVE SUBROUTINES ('205, '206, '245, '252, '207)
0120 GOSUB '216 (5,"205206245252207")

0130 REM STEP 6, BRANCH TO LINE 400 IF AN ERROR IS RETURNED
0140 IF R9$ <> HEX(00) THEN 0400

0150 REM STEP 7, LOAD REQUIRED DOCUMENT ACCESS SUBROUTINES
0160 LOAD DCT #0, <R9> R9$() 4050,5500 BEG 0190

0170 REM STEP 8, CALLS TO DOCUMENT ACCESS SUBROUTINES
0180 REM CALL TO INITIALIZE DATA SUBROUTINE
0190 GOSUB '205

0200 REM  CALL TO OPEN DOCUMENT SUBROUTINE, DOCUMENT IS "0015A", PASSWORD
     IS " ", SLOT NUMBER IS 1
0210 GOSUB '206 ("0015A", " ",1)
0220 IF B1$ <> HEX(00) THEN 0400
0230 REM PROCESSING TO CREATE V0$(), THE PAGE TO BE APPENDED
     .
     .
     .

0260 REM CALL GO TO PAGE SUBROUTINE, SLOT NUMBER IS 1
0270 REM GO TO PAGE POSITIONS TO LAST PAGE OF DOCUMENT IF 999 IS USED
0280 GOSUB '245 (1,999)
0290 IF B1$ <> HEX(00) THEN 0400

0300 REM CALL TO APPEND PAGE SUBROUTINE, SLOT NUMBER IS 1
0310 GOSUB '252 (1)
0320 IF B1$ <> HEX(00) THEN 0400

0330 REM CALL TO CLOSE DOCUMENT SUBROUTINE, SLOT NUMBER IS 1
0340 GOSUB '207 (1)
0350 IF B1$ <> HEX(00) THEN 0400
     .
     .
     .
0400 REM ERROR HANDLING ROUTINE
     .
     .
     .
```

1.5.3 Miscellaneous Programming Notes

When using the Document Access subroutines, a programmer must allow for the following programming restrictions.

1. Passwords should conform to the current 2200 Word Processing convention i.e. upper/lowercase alphabetic characters and numeric digits. Word processing will not allow editing of a document containing an illegal password.

2. A page of a document contains a maximum of 4181 characters, including all format line characters and the page break character. Each document contains a maximum of 116 pages.

3. Each page of a document must begin with a format line and each page, excluding the last page, must end with a page break. This convention must be followed when adding or replacing pages in a document.

4. If the format line of a document is greater than 80 characters, the document can not be edited in a 28K partition; a larger partition and the horizontal scroll feature are required. A 28K partition is adequate to open, close, and print the document.

NOTE

The Document Access subroutines cannot be modified by the programmer.

1.5.4 Error Codes

Table 1-3 lists some possible errors that result when using the document access subroutines. The error codes are returned in the variable B1$.

Table 1-3. Error Codes

Hex Error Code	Description
01	Volume Full
02	File or Volume already exists
03	File or Volume does not exist
04	No free device slots
05	Incorrect password
06	Open access type error - (File is open by another user, or was previously opened by this user)
07	File not open
08	Illegal File ID
09	Not enough room in file (to reuse scratched file)
10	File mess up
20	EOF reached unexpected (fatal)
22	Destination VAU not valid
23	Buffer variables not valid
24	No VAU's in file
25	Source & VAU # inconsistent
26	Volume init parameter inconsistent
27	Byte parameter error in replace
28	EOF reached normal (not fatal)
29	Data transfer with greater than 128 VAU's
A1	Page table full
A2	Last page cannot be deleted
A3	Page does not exist
A4	User defined slot number already assigned
A5	Illegal file name
A6	Library map not found on the selected disk
A7	Library has not been established
A8	Illegal page number
A9	Prototype doesn't exist
B0	Prototype not accessible
B1	Glossary not attached
B2	Glossary not verified
B3	Glossary index exceeds one sector
B4	Glossary entry not found
B5	Wrong numeric type for admin

Numeric Error Code	Description
80 through 89	Disk errors (refer to BASIC-2 manual)
90 through 99	I/O errors (refer to BASIC-2 manual)
** HEX(00)	Normal, successful return

1.6 REFERENCE GUIDE TO THE DOCUMENT ACCESS SUBROUTINES

The following subsections provide a reference for the Document Access Subroutine user. Each subsection is titled with the name of the subroutine, briefly describes the purpose of the subroutine, and states the DEFFN' assignment for the subroutine. A list of the parameters passed to the subroutine (INPUT), and parameters received from the subroutine (OUTPUT) is included. Error codes returned when a subroutine call is unsuccessful are described in Subsection 1.5.4.

1.6.1 Initialize Data

DEFFN'205

INPUT: None

OUTPUT: None

The application program initializes data and dimensions the variables that are used by the system with this subroutine. It must be the first Document Access subroutine call in the application program and should not be called again in the program.

1.6.2 Open Document (Exclusive Mode)

```
DEFFN'206 (R5$, A6$, B9)

INPUT:  R5$ = Document ID
        A6$ = Document password
         B9 = User-defined slot number (1-4)

OUTPUT: B1$ = Return code = HEX(00) if subroutine call is successful
            = Error code if subroutine call is unsuccessful
```

This subroutine opens the named document in Exclusive mode. The named document must be an existing word processing document when the subroutine call is made. Up to four documents can be open at a time. The user assigned slot number will remain associated with the document as long as the document stays open.

If a document is already open at the user defined slot, the current file will be left open, and the return code, B1$ = HEX(A4) will result.

If the user specified document is already open, by either the current user or another user's program, the return code B1$ = HEX(06) will result.

If the document is not password protected, A6$ is input as " ".

1.6.3 Close Document

```
DEFFN'207 (B9)

INPUT:    B9 = User-defined slot number (1-4)
             = 0 means close all open documents

OUTPUT: B1$ = Return code = HEX(00) if subroutine call is successful
             = Error code if subroutine call is unsuccessful
```

This subroutine closes the document associated with the specified slot number. If the number belongs to an attached glossary, it will detach the glossary. If no document associated with the specified slot number is open, the return code, B1$ = HEX(07) will result.

If the user supplies B9 = 0, the subroutine will close all the currently open documents, including attached glossary, if any.

1.6.4 Create Document

DEFFN'208 (R5$, A6$, B9)

```
INPUT:  R5$ = Document ID or "NEXTa" (where "a" represents the library)
        A6$ = Document password
        B9  = User-defined slot number (1-4)

OUTPUT: R5$ = Document ID created
        B1$ = Return code = Error code if subroutine call is unsuccessful
```

This subroutine creates a new document with only one page. The specified library must exist before the call to this subroutine is made. The document contains the same initial format as the prototype document for the specified library. The variable R5$ should either be a specific document ID or in the "NEXTa" form. "NEXTa" form means the next available document ID in library "a".

After successful document creation, the document will stay open in Exclusive mode, associated with the user-assigned slot number. R5$ contains the document ID of the newly created document.

1.6.5 Delete Document

DEFFN'209 (B9)

INPUT: B9 = User-defined slot number (1-4)

OUTPUT: B1$ = Return code = Error code if subroutine call is unsuccessful

This subroutine deletes the document associated with the user-specified slot number. The document must first be opened with the Open Document subroutine before it can be deleted with this subroutine.

1.6.6 Change Password

DEFFN'079 (A6$, B9)

```
INPUT:  A6$ = New password
         B9 = User-defined slot number (1-4)

OUTPUT: B1$ = Return code = Error code if subroutine call is unsuccessful
```

This subroutine assigns a new password to the document associated with the user-specified slot number. The document must be opened with the Open Document or Create New Document subroutine before a password can be assigned with this subroutine.

Embedded spaces in the password are not accepted by other Wang Word Processing systems.

1.6.7 Go To Page

```
DEFFN'245 (B9, A7)

INPUT:    B9 = User-defined slot number (1-4)
          A7 = Page number

OUTPUT:   A7 = Page number
         B1$ = Return code = Error code if subroutine call is unsuccessful
```

This subroutine positions a current page marker at the specified page. The page is not read by this subroutine. When going to an actual page number, the current page marker will be positioned to the specified page. If the specified page is 999 (A7=999), the current page marker is set to the last page of the document, and the variable A7 will equal the number of the last page. The Footer, Header, Work, and Extra page are numbered -3, -2, -1, and 0, respectively.

1.6.8 Read Current Page

DEFFN'247 (B9)

```
INPUT:   B9 = User-defined slot number (1-4)

OUTPUT: V0$() = Current page
          B1$ = Return code = Error code if subroutine call is
                unsuccessful
```

This subroutine reads the contents of a page and places the contents in the variable V0$(). The current page marker must be assigned by the Go To Page subroutine before calling this subroutine. The current page of the document associated with slot number B9 is read. The current page marker is not altered by this subroutine call.

1.6.9 Rewrite/Replace Page

DEFFN'248 (B9)

```
INPUT:      B9 = User-defined slot number (1-4)
         V0$() = Page buffer

OUTPUT:    B1$ = Return code = Error code if subroutine call is
                 unsuccessful
```

This subroutine places the contents of the page buffer, V0$(), into the document at the current page. The current page marker must be assigned by the Go To Page subroutine before calling this subroutine. The contents of the variable V0$() will replace the contents of the current page of the document associated with slot number B9.

The buffer must represent a valid 2200 Word Processing document page, as described in Section 1.2. Its correctness will not be verified. An error code is returned if the page is invalid.

The current page marker remains unchanged by this subroutine call.

1.6.10 Insert/Write New Page

DEFFN'249 (B9)

```
INPUT:      B9 = User-defined slot number (1-4)
          V0$() = Page buffer

OUTPUT:    B1$ = Return code = Error code if subroutine call is
                 unsuccessful
```

This subroutine inserts the contents of the page buffer, V0$(), as a new page directly before the current page of the document associated with slot number B9. The current page marker must be assigned by the Go To Page subroutine before calling this subroutine. The current page marker remains unchanged by this subroutine call.

For example, if the current page marker points to Page 3, after inserting a new page, the current page marker will remain 3, the original Page 3 will become Page 4, and the newly inserted page will become Page 3.

The page buffer, V0$(), must represent a valid 2200 Word Processing document page, as described in Section 1.2. Its correctness will not be verified. An error code is returned if the page is invalid.

This subroutine only inserts pages of text. To modify the Footer, Header, or Work page the Rewrite/Replace subroutine must be used.

The 2200 Word Processing Software system will allow a maximum of 116 text pages per document.

1.6.11 Delete Current Page

DEFFN'251 (B9)

INPUT: B9 = User-defined slot number (1-4)

OUTPUT: B1$ = Return code = Error code if subroutine call is unsuccessful

This subroutine deletes the current page from the document associated with the specified slot number. The current page marker must be assigned by the Go To Page subroutine before calling this subroutine. The current page marker will remain unchanged. For example, if Page 3 is deleted, Page 4 now becomes Page 3 and the current page marker would still be pointing to Page 3.

The system does not allow the user to delete the last remaining page of a document; the user must retain at least one regular text page in a document.

1.6.12 Append Page

DEFFN'252 (B9)

```
INPUT:      B9 = User-defined slot number (1-4)
          VO$() = Buffer page

OUTPUT:    B1$ = Return code = Error code if subroutine call is
                 unsuccessful
```

This subroutine allows the user to add the contents of the buffer page, VO$(), as a new last page of the document associated with slot number B9. The Go To Page subroutine must be called prior to calling the Append Page subroutine so the current page marker can be positioned to the last page of the document. Upon return from the Append Page subroutine call, the current page marker will point to the new last page.

The buffer must represent a valid 2200 Word Processing document page, as described in Section 1.2. Its correctness will not be verified. An error code is returned if the page is invalid.

The 2200 Word Processing Software system allows a maximum of 116 text pages.

1.6.13 <u>Search Text String</u>

```
DEFFN'253 (B8$, B6)

INPUT:  B8$ = Search string
        B6  = Byte location to begin search

OUTPUT:  B1 = Byte location where the search string begins in the page
        B1$ = Return code = Error code if subroutine call is unsuccessful
```

This subroutine searches the buffer page, V0$(), for the first occurrence of the specified search string, starting at the specified byte number in the page. The contents of the page must be placed in the variable V0$() by calling the Read Page subroutine prior to the call to this subroutine. If the string is found, the return variable, B1, will equal the byte number where the string begins in the page. If the string is not found, the return variable will be equal to zero.

This function performs a case insensitive search. For example, if the search string is "boston", the subroutine will locate "BOSTON", "Boston", and "boston".

1.6.14 Read Administrative Information

DEFFN'218 (B9)

INPUT: B9 = User-defined slot number (1-4)

```
OUTPUT:                B1$ = Return code
     A6$(), A7$(), A0() = Administrative information
              B0(), A9$() = Print defaults
```

This subroutine returns the current values of the administrative information of the document associated with the specified slot number. Administrative information is the statistical information presented in the Document Summary. This subroutine also returns the print defaults of the document. Print defaults are the information presented to the user on the Print Document menu.

The output from the Read Administrative Information subroutine call consists of returned values for the following variables. The list of variables contains a description of the administrative information or print default field represented by the variable and the length of each returned alphanumeric variable.

Alpha Array A6$(4)25

A6$(1)	Document Name	25 bytes
A6$(2)	Operator	20 bytes
A6$(3)	Author	20 bytes
A6$(4)	Comments	20 bytes

Alpha Array A7$(13)6

A7$(1)	Document ID	5 bytes
A7$(2)	Created Date	6 bytes
A7$(3)	Created Time	4 bytes
A7$(4)	Created Worktime	6 bytes
A7$(5)	Last Revised Date	6 bytes
A7$(6)	Last Revised Time	4 bytes
A7$(7)	Last Revised Worktime	6 bytes
A7$(8)	Last Printed Date	6 bytes
A7$(9)	Last Printed Time	4 bytes
A7$(10)	Last Archived Date	6 bytes
A7$(11)	Last Archived Time	4 bytes
A7$(12)	Archive ID	5 bytes
A7$(13)	Total Worktime	4 bytes

Numeric Array A0(5)

A0(1)	Created Keystrokes
A0(2)	Last Revised Keystrokes
A0(3)	Total Pages
A0(4)	Total Lines
A0(5)	Total Keystrokes

Numeric Array B0(11)

B0(1)	Print from Page
B0(2)	Print thru Page
B0(3)	Starting as Page No.
B0(4)	First Header Page
B0(5)	First Footer Page
B0(6)	Footer Begins on Line
B0(7)	Paper Length
B0(8)	No. of Originals
B0(9)	Character Set Number
B0(10)	Printer Number
B0(11)	Left Margin

Alpha Array A9$()1

A9$(1)	Device	1 byte
A9$(2)	Pitch	1 byte
A9$(3)	Format	1 byte
A9$(4)	Forms	1 byte
A9$(5)	Style	1 byte
A9$(6)	Summary	1 byte
A9$(7)	Delete	1 byte

When a document is printed by means of the 2200 Print Document menu, the Device, Pitch, Format, Forms, Style, and Summary options are selected by positioning an acceptance block. To supply these selections through the Document Access subroutines, the appropriate variable from the previous list equals HEX(01) to represent the first field selection, HEX(02) to represent the second field selection, HEX(03) to represent the third field selection, and so on.

1.6.15 Write Administrative Information

DEFFN'219 (B9)

INPUT: B9 = User-defined slot number (1-4)
A6$(), A7$(), A0() = Administrative information
B0(), A9$() = Print defaults

OUTPUT: B1$ = Return code = Error code if subroutine call is unsuccessful

This subroutine updates the administrative information and print default information for the document associated with the specified slot number. Specific variable assignments are described in the Read Administrative Information subroutine located in Subsection 1.6.14.

If successful calls to Document Access subroutines result in changes to the page count of a document, the page count is automatically adjusted in the document's administrative information. In this case, it is not necessary to update the page count by calling the Write Administrative Information subroutine.

1.6.16 Attach Glossary

DEFFN'229 (R5$, A6$, B9)

INPUT: R5$ = Glossary ID
A6$ = Glossary password
B9 = User-defined slot number (1-4)

OUTPUT: B1$ = Return code = Error code if subroutine call is unsuccessful

This subroutine attaches a glossary to the workstation for subsequent use in the modification of a document by the application program. Attaching a glossary opens the glossary for further use and uses up one of the four slots available for documents.

Only one glossary may be attached at a time. Only text recall glossaries are allowed. Because this subroutine opens a glossary file, the application program must close the file with the Detach Glossary subroutine or the Close Document (close all files option) subroutine.

1.6.17 Call Glossary

DEFFN'230 (A4$)

INPUT: A4$ = Glossary label

OUTPUT: V0$() = Glossary text
B1$ = Return code = Error code if subroutine call is unsuccessful

The glossary has to be attached before calling glossary text. This subroutine looks up the appropriate glossary entry by the specified glossary label and returns the glossary text in the variable V0$(). The subroutine is similar in function to the Read Page Subroutine. The application program must perform the actual insertion of this glossary text into the current page of a document.

1.6.18 Detach Glossary

DEFFN'233 (B9)

INPUT: B9 = User-defined slot number

OUTPUT: B1$ = Return code = Error code if subroutine call is
                  unsuccessful

This subroutine first detaches the glossary from the workstation, and then closes the detached glossary.

CHAPTER 2
2200 WORD PROCESSING MENU MODIFICATION

2.1 <u>INTRODUCTION</u>

2200 Word Processing Software provided by Wang Laboratories, Inc., supplies the purchaser with BASIC-2 programs that perform word processing functions. Specifically, the BASIC-2 program "609menu" performs the following functions. (The following description of functions and procedures used to execute "609menu" will be familiar to readers who have customized their 2200 system menus by modifying "@MENU".)

1. Displays the following 2200 Word Processing System menus: Word Processing menu, Special Print Functions menu, Advanced Function menu, Utilities menu, and Glossary Functions menu.

2. Accepts user-supplied selections from the displayed menu.

3. Loads the program or menu corresponding to the user selection.

Figure 2-1, the Advanced Functions menu, is a example menu that can be displayed by the execution of the BASIC-2 program "609menu" and a menu file.

```
WANG 2200 Word Processing System
      Advanced Functions



. Advanced Filing
. Convert WP Document to TC File
. Convert TC File to WP Format
. Document Merge
. Keyword Search
```

Figure 2-1. Advanced Functions Menu

When the user executes "609menu", the program calls into memory a menu file. When the 2200 Word Processing daily startup procedure is followed, the LOAD RUN command automatically executes "609menu". (Refer to the 2200 Word Processing Operator's Guide for a description of the daily startup procedure.) The first menu displayed by the automatic execution of "609menu" is the 2200 Word Processing menu, defined by the menu file "609MENU".

Each menu file consists of DATA statements with line numbers in the range 9000 through 9999. A menu file supplies the following information: the name of the menu file, a menu title, menu message, and menu selection text. A list of the 2200 Word Processing System menus and their menu file names follows.

2200 Menu	Menu File Name
Word Processing	609MENU
Special Print Functions	609SPRNT
Advanced Functions	609ADV
Utilities	609UTIL
Glossary Functions	609Glos

The following information from the sample menu shown in Figure 2-1 is contained in a menu file named "609ADV". Of the following information, only the menu file name is not displayed on the menu screen.

1. Menu file name, "609ADV"
2. Menu title, "WANG 2200 Word Processing System"
3. Menu message, "Advanced Functions"
4. Menu selection text, "Advanced Filing"
5. Menu selection text, "Convert WP Document to TC File"
6. Menu selection text, "Convert TC File to WP Document"
7. Menu selection text, "Document Merge"
8. Menu selection text, "Keyword Search"

In addition to the information displayed on the sample menu, the menu file also provides data to indicate if a selection invokes a program (P), or another menu (M), along with the name of the invoked program or menu.

2.2 MENU FILES

When the user executes "609menu", the program calls into memory a menu file. The following rules apply to menu files.

1. In 2200 Word Processing mode, the first menu displayed by executing "609menu" is the Word Processing menu, which corresponds to the menu file named "609MENU". Any valid file name can be used for subsequent menus invoked by selections from any 2200 Word Processing System menus.

2. The DATA statements contained in the all menu files must be in the range 9000 through 9999.

3. The maximum number of selections per menu is 15.

4. Programs or menus named in the DATA statements of a menu file must reside on same disk surface as the 2200 Word Processing Software.

2.3 MENU FILE STRUCTURE

The menu files must contain DATA statement that provide data in a specified sequence. The first, second, and last DATA statements have unique forms; but the intermediate DATA statements are similar in form. To add menu selections to a 2200 Word Processing menu, the user must add intermediate DATA statements to the appropriate menu file. Descriptions of the form for each DATA statement follow.

2.3.1 The First DATA Statement

The first DATA statement supplies four data values. The following is the general form of the first DATA statement.

General Form:

```
DATA "Menu file name", "Menu title", 3, n
```

Where:

Menu file name =	The name of the menu file currently in memory.
Menu title =	A menu title that appears on the first line of the displayed menu.
3 =	A data value that represents the number of data entries in the intermediate DATA statements.
n =	0, 1, or 2; a data value that represents the number of data entries in the second DATA statement. A value of zero for n indicates the second DATA statement is omitted.

2.3.2 The Second DATA Statement

The second DATA statement is an optional statement that supplies one, or two, data values depending on the user's screen design. The following is the general form of the second DATA statement.

General Form:

```
DATA "Menu file name"[, "Menu message"]
```

Where:

Menu file name =	The name of the menu to display when the user presses the CANCEL key instead of making a menu selection from the displayed menu. If " " is entered the Wang 2200 Word Processing menu is displayed.
Menu message =	A message that is displayed on the second line of the menu screen. If the data value is entered as " ", no message is displayed.

2.3.3 Intermediate DATA Statements

Each intermediate DATA statement supplies three data values. Since intermediate DATA statements correspond to a selection choice on the menu, the number of intermediate DATA statements is equal to the number of menu selections. The following is the general form of the intermediate DATA statements. Note that if the third item in any given intermediate DATA statement is not "P" or "M", the corresponding entry will not be displayed on the menu.

General Form:

```
DATA {"Program name", "Menu selection text", "P"   }
     {"Menu file name", "Menu selection text", "M" }
```

Where:

Menu selection text =	Text displayed on the menu screen for each menu selection. The maximum text length is 50 characters.
Program name =	The name of the program to execute when the corresponding menu selection text is chosen. Alternately, a list of program module names, separated by commas, may be supplied. When the alternate form is used, all the program modules are loaded at the same time before execution begins.
Menu file name =	The name of the menu to display when the corresponding menu selection text is chosen.

2.3.4 The Last DATA Statement

The last DATA statement supplies three literals. The following is the exact form of the last DATA statement in the menu file. The last DATA statement of every menu file must appear as shown in the exact form.

Exact Form:

```
DATA "no more","end of menu list"," "
```

2.4 AN EXAMPLE MENU FILE

The following sample menu file produces the Advanced Functions menu shown in Figure 2-1. The menu file name is "609ADV".

```
9000  REM "Menu file name", "Menu title", 3, 2 entries in next DATA
9010  DATA "609ADV", "WANG 2200 Word Processing System", 3,2
9020  REM "Menu data file name" for CANCEL, "Menu message"
9030  DATA "609MENU ", "Advanced Functions"

9040  REM "Menu or Program" to invoke, "Selection text", "P or M"
9050  DATA "609Fstrt", "Advanced Filing", "P"
9060  DATA "609WP100", "Convert WP Document to TC File", "P"
9070  DATA "609TC100", "Convert TC File to WP Document", "P"
9080  DATA "609MERGO,Prime62,609Open,JCAT,REC-GEN,REC-RD", "Document
      Merge", "P"

9090  DATA "no more","end of menu list"," "
```

If the following DATA statement is added to "609ADV", the sixth entry appearing on the Advanced Functions menu is "User-supplied program". Selection of the "User-supplied program" entry executes the program "PROG1".

```
9085  DATA "PROG1", "User-supplied program", "P"
```

CHAPTER 3
ACCESSING 2200 WORD PROCESSING FROM USER APPLICATIONS

3.1 INTRODUCTION

This chapter describes how an executing application program invokes 2200 Word Processing; how 2200 Word Processing can be terminated after it is invoked from an executing application; and how an executing application program can invoke a specific 2200 Word Processing function.

To create an application that invokes word processing, or a word processing function, the programmer should be familiar with 2200 Word Processing software conventions. In particular, an application that invokes 2200 Word Processing will observe the following conventions.

1. The alpha array R3$() is reserved by the word processing system as a system stack. Each element in the system stack is a program name. The stack is used by the word processing system to store and process multiple program names.

2. $PSTAT must be set equal to the name of the word processing system menu that will be displayed when the application enters the word processing system.

Normal 2200 word processing system flow is the result of the 2200 word processing daily startup procedure. When the procedure is followed, the user issues a LOAD RUN command that results in the automatic execution of the following four BASIC-2 programs: START, WPstart, 609start, and 609menu. The following list describes the purpose served by each program.

Program	Purpose
START	Closes all open data files and executes the WPstart program.
WPstart	Closes all open data files, sets $PSTAT = " ", and executes the 609start program.
609start	Sets up all common variables needed by 2200 Word Processing, removes the first element from the system stack, and executes the 609menu program.
609menu	Displays the 2200 Word Processing menu. Chapter 2 describes in more detail the purpose and use of this program.

3.2 ENTERING WORD PROCESSING FROM AN APPLICATION

The program 609start is the normal entry point into the 2200 Word Processing system from a user-application. 609start establishes the word processing enviroment without closing any files opened by the application. In addition, 609start removes the first program name in the system stack, executes the named program, and pops up succeeding elements in the stack. To properly execute 2200 Word Processing from a user-application, the following conditions must be established by the application.

1. The word processing menu driver program, 609menu, must be the first entry in the R3$() system stack.

2. If the application must provide an exit from word processing, the program that word processing must return control to is named in the system stack. Specifically, the second entry in the system stack must be the name of the program that word processing exits to.

3. Since $PSTAT determines the entry menu that will be displayed when the application enters the word processing system, it must be set equal to the name of a 2200 Word Processing menu. A list of the 2200 Word Processing menus, and the corresponding values that must be assigned to $PSTAT, follows.

2200 Menu	$PSTAT Value
Word Processing	609MENU
Special Print Functions	609SPRNT
Advanced Functions	609ADV
Utilities	609UTIL
Glossary Functions	609Glos
Supervisory Functions	609SUPER
System Management Functions	609MANAG

4. The application program loads 609start.

As a programming aid, 2200 Word Processing software contains the program 609ENTRY. 609ENTRY contains most of the statements needed in an entry module. However, depending on the application, additional statements should be added to 609ENTRY to define an exit point from word processing and to COMmon R3$(). The following statements are contained in 609ENTRY.

```
10 REM '609ENTRY' - ENTER 2200WP FROM IDEAS2
20 DIM C$(4)62
 : I=LEN(R3$())
 : MAT COPY -R3$()<1,I> to -R3$()<10,I>
 : R3$(1) = "609menu"
 : $PSTAT = "609MENU"
 : LOAD T "609start"
```

3.3 EXITING WORD PROCESSING

When 2200 Word Processing is entered by an application, the entry menu displayed is determined by the value $PSTAT assigned by the application. Once the word processing system is entered it functions in the same way as if the system were entered by means of the 2200 Word Processing daily startup procedure, with the following exeception. 2200 Word Processing software contains an exit routine that is available when the 2200 Word Processing entry menu is displayed.

To activate the word processing exit routine, the following exit procedure is followed: when the 2200 Word Processing entry menu is displayed, the user holds down the SHIFT key and simultaneously presses the CANCEL key. Caution should be taken to follow the word processing exit procedure only when the entry menu is displayed. If the exit procedure is followed at any other time unpredictable results can occur.

As described in Section 3.2, the program that word processing must return control to when the word processing exit procedure is followed is named in the system stack. Since entry into the word processing system pops the system stack, the name of the return program is the first element in the stack when the exit procedure is followed. If the system stack does not contain an entry and the exit procedure is followed, the 2200 Word Processing system is not exited.

3.4 SAMPLE PROGRAMS

The following examples illustrate programming techniques that allow applications to enter, and exit from, the 2200 Word Processing system.

```
 10 REM  PROGRAM 'ENTRY' ENTERS 2200 WORD PROCESSING
 20 REM  LINE 30 DECLARES R3$(10)9 AS A COMMON VARIABLE
 30 COM  R3$(10)9
 40 REM  LINES 50, 60 PUSH DOWN THE SYSTEM STACK
 50 I=LEN(R3$())
 60 MAT COPY -R3$()<1,I> to -R3$()<10,I>
 70 REM  LINE 80 SETS SYSTEM STACK
 80 R3$(1) = "609menu": R3$(2) = "EXIT"
 90 PRINT "ENTERING 2200WP"
100 REM  LINE 110 SETS $PSTAT EQUAL TO DESIRED 2200 WP MENU NAME
110 $PSTAT = "609MENU"
120 REM  LINE 130 LOADS "609start"
130 LOAD T "609start"
```

```
 10 REM  PROGRAM NAME IS "EXIT"
 20 REM  THIS PROGRAM EXITS FROM 2200 Word Processing
 30 PRINT "EXITING WORD PROCESSING"
```

APPENDIX A
DOCUMENT ACCESS SUBROUTINE MODULES

A.1 MODULE NAMES

Chapter 1 describes two methods that can be used to load the Document Access subroutines; the direct method and the DAST.DAT method. When the DAST.DAT method is used, module numbers and names are supplied in data statements contained in DAST.DAT. To use the direct method, the application must supply values for I9(4), the number of modules to load, and I9$(), the names of the modules to load. Module numbers and module names, grouped by the subroutine functions they perform, are shown in the following list.

Subroutine Function	DEFFN' Assignment	Number of Modules	Module Names
Initialize Data	205	2	DOC.INIT
			REC.INIT
Open Document	206	7	DOC.OPEN
			CAT.VLIS
			CAT.PRIM
			CAT.RD
			CAT.LIST
			CAT.FO/C
			CAT.OPEN
Close Document	207	5	DOC.CLOS
			CAT.FO/C
			CAT.PRIM
			CAT.OPEN
			CAT.RD
Create Document	208	14	DOC.CRET
			CAT.VLIS
			CAT.FCRE
			CAT.FO/C
			CAT.FLIS
			CAT.PRIM
			CAT.OPEN
			CAT.RD
			CAT.INS
			CAT.LIST
			REC.PRIM

Subroutine Function	DEFFN' Assignment	Number of Modules	Module Names
			REC.RD
			REC.INS
			REC.FREE
Delete Document	209	8	DOC.DEL
			CAT.FDEL
			CAT.PRIM
			CAT.RD
			CAT.DEL
			REC.PRIM
			REC.FREE
			REC.DEL
Change Password	079	9	DOC.PASS
			CAT.FMOD
			CAT.PRIM
			CAT.RD
			CAT.INS
			CAT.DEL
			REC.PRIM
			REC.INS
			REC.FREE
Go To Page	245	4	PAG.GOTO
			PAG.SUB
			REC.PRIM
			REC.RD
Read Page	247	4	PAG.READ
			PAG.SUB
			REC.PRIM
			REC.RD
Rewrite Page	248	8	PAG.REWT
			PAG.SUB
			REC.PRIM
			REC.RD
			REC.INS
			REC.FREE
			REC.DEL
			REC.REP
Insert Page	249	10	PAG.INS
			PAG.SUB
			REC.PRIM
			REC.RD
			REC.INS
			REC.FREE
			ADM.READ
			ADM.WRIT
			CAT.FFCB
			CAT.PRIM

Subroutine Function	DEFFN' Assignment	Number of Modules	Module Names
Delete Page	251	11	PAG.DEL PAG.SUB REC.PRIM REC.DEL REC.INS REC.FREE ADM.READ ADM.WRIT CAT.FFCB REC.RD CAT.PRIM
Append Page	252	10	PAG.APD PAG.SUB REC.PRIM REC.INS REC.FREE REC.RD ADM.READ ADM.WRIT CAT.PRIM CAT.FFCB
Search Text	253	2	PAG.SRCH PAG.SUB
Read Administrative Information	218	3	ADM.READ REC.PRIM REC.RD
Write Administrative Information	219	5	ADM.WRIT REC.PRIM REC.RD CAT.PRIM CAT.FFCB
Attach Glossary	229	7	GLS.ATCH CAT.VLIS CAT.PRIM CAT.RD CAT.LIST CAT.FO/C CAT.OPEN
Call Glossary	230	3	GLS.CALL REC.PRIM REC.RD
Detach Glossary	233	5	GLS.DTCH CAT.PRIM CAT.FO/C CAT.OPEN CAT.RD

INDEX

WANG

Customer Comment Form

Publications Number 700-6961

Title 2200 PROGRAMMER'S GUIDE TO WORD PROCESSING

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other ______

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other ______

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? ______

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) ______

Do you have any other comments or suggestions? ______

Name ______ **Street** ______

Title ______ **City** ______

Dept/Mail Stop ______ **State/Country** ______

Company ______ **Zip Code** ______ **Telephone** ______

Thank you for your help.

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
CHARLES T. PEERS, JR., MAIL STOP 1369
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

The completed order form should be mailed to:
WANG LABORATORIES, INC.
Supplies Division
51 Middlesex St.
No. Chelmsford MA 01863

To Order by Phone, Call:
(800)225-0234
From Mass., Hawaii, and Alaska
(617)256-1400
TELEX 951-743

Order Form for Wang Manuals and Documentation

① Customer Number (If Known)

② Bill To: ______ Ship To: ______

③ Customer Contact: ( ) ( ) Phone Name ④ Date ______ Purchase Order Number ______

⑤ Taxable Yes ☐ No ☐ ⑥Tax Exempt Number ⑦Credit This Order to A Wang Salesperson Please Complete ______ Salesperson's Name ______ Employee No. ______ RDB No.

⑧ Document Number	Description	Quantity	⑨ Unit Price	Total Price

⑩ ______ Authorized Signature ______ Date

☐ Check this box if you would like a free copy of the **Corporate Publications Literature Catalog** (700-5294)

Sub Total	
Less Any Applicable Discount	
Sub Total	
LocalStateTax	
Total Amount	

Ordering Instructions

1. If you have purchased supplies from Wang before, and know your Customer Number, please write it here.
2. Provide appropriate Billing Address and Shipping Address.
3. Please provide a phone number and name, should it be necessary for WANG to contact you about your order.
4. Your purchase order number and date.
5. Show whether order is taxable or not.
6. If tax-exempt, please provide your exemption number.
7. If you wish credit for this order to be given to a WANG salesperson, please complete.
8. Show part numbers, description and quantity for each product ordered.
9. *Pricing extensions and totaling can be completed at your option; Wang will refigure these prices and add freight on your invoice.*
10. Signature of authorized buyer and date.

Wang Supplies Division Terms and Conditions

1. **TAXES** — Prices are exclusive of all sales, use, and like taxes.
2. **DELIVERY** — Delivery will be F.O.B. Wang's plant. Customer will be billed for freight charges; and unless customer specifies otherwise, all shipments will go best way surface as determined by Wang. Wang shall not assume any liability in connection with the shipment nor shall the carrier be construed to be an agent of Wang. If the customer requests that Wang arrange for insurance the customer will be billed for the insurance charges.
3. **PAYMENT** — Terms are net 30 days from date of invoice. Unless otherwise stated by customer, partial shipments will generate partial invoices.
4. **PRICES** — The prices shown are subject to change without notice. Individual document prices may be found in the Corporate Publications Literature Catalog (700-5294)
5. **LIMITATION OF LIABILITY** — In no event shall Wang be liable for loss of data or for special, incidental or consequential damages in connection with or arising out of the use of or information contained in any manuals or documentation furnished hereunder.

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 NO. CHELMSFORD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
Supplies Division
c/o Order Entry Dept.
M/S 5511
51 Middlesex St.
No. Chelmsford, MA 01863

Fold

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
700-6961
8-82-2M