2200

2200/VS Local Communications Option Programmer's Reference Guide

SOFTWARE

2200/VS TERM/V DISK REV: 2:00	731-80[illegible]1A	360K
2200/VS ACCESS SUB DISK REV 2:00	731-8012A	360K
VS CONTROL/MCODE VOL:2258 REV 2.00 DISK 1	735-9333A	360K
VS CONTROL/MCODE VOL:2258 REV 2.00 DISK 2	735-7011	360K

2200

2200/VS Local Communications Option Programmer's Reference Guide

2nd Edition — May 1991
Copyright © Wang Laboratories, Inc., 1986, 1991
All rights reserved.
715-0562A

WANG

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVE., LOWELL, MA 01851, TEL. (508) 459-5000, TELEX 172108

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual. However, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which the product was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the product, the accompanying manual, or any related materials.

Software Notice

All Wang Program Products (software) are licensed to customers in accordance with the terms and conditions of the Wang Standard Software License. No title or ownership of Wang software is transferred, and any use of the software beyond the terms of the aforesaid license, without the written authorization of Wang, is prohibited.

Warning

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device, pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Restricted Rights Legend

Use, duplication or disclosure by the Government of this technical data is subject to restrictions stated in paragraph (A) (15) of the Rights in Technical Data and Computer Software clause at 48-CFR-252-227-7013.

CONTENTS

PREFACE

CHAPTER 1 INTRODUCTION

1.1 Overview 1-1
1.2 2200/VS LCO Filing Services 1-1

CHAPTER 2 2200 VDISK ACCESS

2.1 Introduction 2-1
2.2 Using VDISKs 2-2
 Using VDISK With Existing 2200 Programs 2-2
 Submitting Programs That Access VDISK 2-3
2.3 VDISK Performance Considerations 2-4
2.4 Improving VDISK Performance 2-5
2.5 Moving Existing 2200 Files to VDISK 2-6

CHAPTER 3 NATIVE DMS ACCESS

3.1 Introduction 3-1
3.2 Brief Description of the DMS Access Subroutines 3-2
3.3 How To Use the DMS Access Subroutines 3-4
3.4 DMS Access Programming Requirements and
 Performance Considerations 3-5
 Submitting Programs That Access Native DMS Files 3-6
3.5 General Notes on the DMS Access Subroutines 3-6
 Variables Reserved by the DMS Access Subroutines 3-6
 Using Variables or Literals for Input Parameters 3-8
3.6 Moving Files From the 2200 to the VS 3-9
3.7 Moving Files From the VS to the 2200 3-10
3.8 Sample Programs 3-10

CONTENTS (continued)

CHAPTER 4 DETAILED DESCRIPTION OF THE DMS ACCESS SUBROUTINES

4.1 Introduction 4-1
4.2 Notes on the General Form Section 4-2
GENERAL OPEN 4-3
GENERAL CLOSE 4-5
CONSECUTIVE READ 4-6
CONSECUTIVE WRITE 4-9
CONSECUTIVE REWRITE 4-11
CONSECUTIVE SKIP 4-13
CONSECUTIVE LOCK 4-14
CONSECUTIVE UNLOCK 4-16
INDEXED READ 4-17
INDEXED READ NEXT 4-19
INDEXED WRITE 4-20
INDEXED REWRITE 4-21
INDEXED DELETE 4-22
INDEXED FIND 4-23
INDEXED LOCK 4-25
INDEXED UNLOCK 4-27
RELATIVE READ 4-28
RELATIVE WRITE 4-31
RELATIVE REWRITE 4-33
RELATIVE DELETE 4-35
BLOCK READ 4-36
BLOCK WRITE 4-37
CREATE FILE 4-38
DELETE FILE 4-41
RENAME FILE 4-42
GET FILE ATTRIBUTES 4-44

APPENDIX A ADDITIONAL INFORMATION

A.1 Introduction A-1
A.2 Return Codes A-1
A.3 Extended File Sharing (EFS) Header A-2
Error Classes A-4
Error Codes A-5
A.4 File Attribute Information A-9
Attribute Structure for General Open A-9
Attribute Structure for Get File Attributes A-10
Attribute Structure for General File Attributes A-11
Attribute Structure for Alternate Key File Attributes A-13
A.5 Create File Information A-14
Create File Parameters A-14
Alternate Key Structure A-16

INDEX

TABLES

Table 3-1 DMS Access Subroutines 3-2
Table 3-2 DMS Catalog Functions 3-4
Table 3-3 Variables Used by the DMS Access Subroutines 3-7
Table 4-1 CONSECUTIVE READ Status Information 4-7
Table 4-2 RELATIVE READ Status Information 4-29
Table A-1 Return Codes A-2
Table A-2 EFS Header A-3
Table A-3 Error Classes A-4
Table A-4 Error Codes A-5
Table A-5 GENERAL OPEN - File Attribute Information Structure A-9
Table A-6 GET FILE ATTRIBUTES - File Attribute Information Structure A-10
Table A-7 General File Attribute Data A-11
Table A-8 Alternate Key File Attribute Data A-13
Table A-9 Create Input Parameter A-14
Table A-10 Alternate Key Input Parameter A-16

PREFACE

The Wang 2200/VS Local Communications Option (LCO) enables Wang 2200 users to emulate up to four VS workstations and store and access files on a connected VS system. This guide provides information on programming requirements for accessing the Wang 2200/VS LCO filing services. The primary audience for this document is the Wang 2200 programmer. Experience with programming on a Wang 2200 system is recommended for all users.

Chapter 1 provides background information on the Wang 2200/VS LCO and serves as an introduction to the guide.

Chapter 2 explains programming requirements for accessing VDISK files (2200 disk image files) stored on a VS system.

Chapter 3 explains programming requirements for accessing native DMS files stored on a VS.

Chapter 4 explains the functions of the DMS access subroutines.

Appendix A contains additional information on return codes, the Extended File Sharing header, file attribute information, and the create and alt-key parameters.

Additional information on the Wang 2200/VS LCO is available in the following Wang publications:

- *2200/VS Local Communications Options User's Guide* (715-0564)
- *2200 BASIC-2 Disk Reference Manual* (700-4081)
- *2200 BASIC-2 Error Codes Booklet* (700-7170)
- *2200 BASIC-2 Language Reference Manual* (700-4080)
- *2200 Introductory Guide* (700-4613)
- *VS Data Management System Reference* (800-1124-01)
- *VS File Management Utilities Reference*

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

The 2200/VS Local Communications Option (LCO) is a data communications hardware (the 2258 controller) and software option that enables a Wang 2200MVP, -LVP, or Micro-VP system to communicate with a Wang VS computer system. The 2200/VS LCO enables you to perform the following functions:

- Log on to the VS and run VS application programs that do not require the downloading of microcode to a VS workstation.
- Run 2200 application programs that store data to and retrieve data from 2200 disk-image files (VDISKs) stored on the VS system.
- Run 2200 application programs that store data to and retrieve data from native VS Data Management System (DMS) files by using subroutines provided with the 2200/VS LCO package.

This programmer's reference guide explains the programming requirements for accessing VDISKs and native DMS files stored on the VS system with new and existing programs. For more information on using VS Workstation Emulation and Filing Services utilities, refer to the *2200/VS Local Communications Option User's Guide*.

1.2 2200/VS LCO FILING SERVICES

The Filing Services utilities of the 2200/VS LCO package enables a 2200 user to access (create, read, and write) VS DMS files through existing and new 2200 BASIC-2 application programs. The VS DMS files are stored on an attached VS system. The VS Data Management System manages all disk file space, and services all file I/O requests on the VS system.

The 2200/VS LCO enables you to store and access information on a VS system in the following forms:

- VDISK
- Native VS DMS files

VDISKs are 2200 disk-image files that are stored on a VS system connected to a 2200 system through the 2200/VS LCO. A VDISK acts like a disk platter to the attached 2200 and enables you to access information stored at the VDISK address as records, files, or sectors.

You create VDISKs on the VS using utilities provided with the 2200/VS LCO software. VDISKs can be shared with other 2200 systems equipped with the 2200/VS LCO package. VDISKs can also be accessed by the VS system. However, you would have to alter the VS application programs to use the files stored on VDISK. How you access VDISKs from an attached 2200 is explained in Chapter 2 of this guide.

Native VS DMS files are formatted by the VS DMS. DMS filing services supports several different file types, including: consecutive, relative, and indexed files. Native DMS files can be accessed both by VS application programs and by other 2200 systems attached to the VS and equipped with the 2200/VS LCO package.

To access native DMS files you must use the DMS access subroutines included with the 2200/VS LCO package. How you access Native DMS files is explained in Chapter 3 of this guide.

CHAPTER 2
2200 VDISK ACCESS

2.1 INTRODUCTION

Using the 2200/VS LCO package to access information stored on an attached VS system through a VDISK is just like using any other 2200 disk facility.

First you must create the VDISKs and assign them disk addresses using the Filing Services utilities that come with the 2200/VS LCO software. For more information on the VDISK Filing Services utilities, refer to the *2200/VS Local Communications Options User's Guide*.

Once you create a 2200 VDISK on the VS, you can use it like any other formatted disk with existing or new 2200 BASIC-2 application programs to write data to and read data from the VDISK.

VDISKs can be opened in Exclusive, Read Only, or Shared mode. In Exclusive mode, VDISKs can only be used by the 2200 system they were opened on. In Shared mode they can be shared with other 2200 systems that are equipped with the 2200/VS LCO package. Read-Only mode is a form of Shared mode, except that the VDISKs are write protected. VDISKS can also be accessed by VS application programs; however, it is not recommended.

When the VDISKs are opened, any user on an attached 2200 can run an application program and access the disks. The 2258 controller responds to the disk address it receives from the application program and passes filing requests to the VS Filing Services program. When the application program is finished processing, you can then run another 2200 application program.

2.2 USING VDISKS

You can use VDISKs with both new and existing BASIC-2 application programs. To access VDISKs, you must know which ones are available. You can use the View VDISK function (included in the 2200/VS LCO File Services utilities) to view a list of available VDISKs. Once you know which VDISKs are available, you can alter existing programs (if necessary) or create new ones to access the VDISKs.

2.2.1 Using VDISK With Existing 2200 Programs

Using VDISKs with existing 2200 BASIC-2 programs should require little or no change to the application programs, except in the following cases:

- If the disk address is hard-coded in the BASIC-2 application program.
- If the BASIC-2 application program and its data file share the same disk address.
- If the program contains a disk address verification program that does not recognize the VDISK addressing scheme.

What To Do When the Disk Address Is Hard-Coded

If the disk address is hard-coded in the application program, you can go through the program and change all references to the disk address to the new VDISK address.

For example, if the disk address was hard-coded in the program as follows:

```
10 SELECT #5 320
```

You could go through the program and change every occurrence of 320 to an existing VDISK address. You can write the example above as follows:

```
10 SELECT #5 D31
```

In this example, 320 is changed to D31, specifying a valid VDISK address.

You can also substitute a variable for the disk address and have the program request the address from the user at runtime. You can then assign to the variable the value received from the user.

What To Do When the Program and Data Files Share the Same Disk

When the program and data files are stored on the same disk, it may not be advisable to use VDISK for performance reasons (refer to Section 2.3).

However, if the data file is sufficiently large to warrant the use of VDISK, you can follow the suggestions in the previous section and hard-code the changes or request the user to input the disk address. In this case, it would be advisable to separate the application program from the data file and store only the data files on the VDISK.

For more information, refer to the appropriate BASIC-2 reference and disk manuals.

What To Do When Verification Procedures Do Not Recognize the Address

If a program contains a disk verification routine that does not recognize the disk addressing scheme used for VDISK, update the verification routine to allow valid VDISK addresses. The valid ranges of addresses for VDISK are as follows:

- DX0 through DXF
- DY0 through DYF

The possible values for X are 1, 2, or 3. You set the value for X in the 2258 controller itself. The value of Y is dependent on the value you assigned to X as follows:

- If X is 1, then Y must be 5.
- If X is 2, then Y must be 6.
- If X is 3, then Y must be 7.

For more information on how to set the address, refer to the *2200/VS Local Communications Options User's Guide*.

2.2.2 Submitting Programs That Access VDISK

Before you can access a file stored on VDISK, you must make certain that the following procedures have been implemented:

- The VDISK has been created using the VDISK utilities.

- The 2200 is actively connected to the VS (the attach procedure has been run).

- The program you submit has or requests a valid VDISK address.

For more information on how to create VDISKs and how to run the attach program, refer to the *2200/VS Local Communications Options User's Guide*. Once you have implemented these procedures, you can submit programs to access VDISK.

2.3 VDISK PERFORMANCE CONSIDERATIONS

VDISK performance is strongly affected by the following components of the 2200/VS LCO software package:

- 2258 firmware
- VS serial I/O processor

2258 Firmware

The 2258 firmware can handle up to four separate tasks; however, only one of these tasks can be assigned to VS Filing Services. The method for attaching to VS Filing Services is described in the *2200/VS Local Communications Options User's Guide*. Once the VS Filing Services task is active, all 2200 partitions (up to 16) can invoke VS Filing Services through the task.

It is important to remember that the 2258 controller can receive requests from any of the 16 possible 2200 partitions and that these requests are handled similarly to 2200 disk requests. As a result, the more requests that are channeled to the VS through a single 2258 controller, the slower the response time experienced by each individual partition.

VS Serial I/O Processor

The VS serial I/O processor (SIOP) handles requests in a serial manner, one request at a time. Each file request requires both a transmission to the SIOP and a response from the SIOP. Since the SIOP handles these requests in a serial manner and since the 2258 controller must handle requests for all partitions, VDISK performance can be adversely affected by the number of requests being processed at any given time.

VDISK performance can also be adversely affected by opening VDISKs in Shared mode. Since VDISKs are actually VS files, response time can be reduced by the additional overhead in the VS system associated with shared files.

When you open VDISKs in Exclusive mode, the disks are open to all partitions on the same 2200. You should only open VDISKs in Shared mode when the VDISK must be shared with another 2200 or when it must be accessed through another 2258 controller on the same 2200.

As the number of VDISKs opened in Shared mode increases, disk commands execute more slowly. When a disk command is executed, the 2258 firmware uses VS DMS commands to lock each VDISK opened in Shared mode. The VDISKs are locked to prevent other 2258 controllers from accessing the VDISK.

If there are many VDISKs open in Shared mode, the locking procedure can add significantly to the processing time required to execute each disk command.

2.4 IMPROVING VDISK PERFORMANCE

To improve, VDISK performance, you can implement the following recommendations::

- Use VDISK only to store large data files for data-intensive programs and for backup storage of program files.
- Do not load programs off VDISK, particularly programs that use program overlays.
- Do not use VDISK to store program-required files, such as screen or message files.
- Avoid opening VDISKs in Shared mode. However, if you must use VDISKs in Shared mode, you may improve performance by bracketing each string of disk commands with a $OPEN and a $CLOSE to reduce processing by the 2258 controller for files opened in Shared mode.
- In large systems or where heavy use of the 2258 data link is expected, using multiple 2258 controllers for VDISK access might increase throughput proportionally. By adding additional 2258 controllers to a single system you can off-load some of the traffic to the additional controllers.

  Note that because of the 2200 disk addressing scheme you are limited to a maximum of three 2258 controllers on a single system for VDISK purposes. You can, however, add additional 2258 controllers either for native DMS access or add more terminals for VS Workstation Emulation. Adding additional 2258 controllers might also improve VDISK performance in large systems.

- The 2258 controllers support utilities enable you to define multiple VDISK maps that assign 2200 platter addresses to an equivalent number of 2200 disk image files. In certain cases you may improve performance by picking a specific VDISK map for a particular application. Once the application is run, you can then return the VDISK map to your normal processing configuration. However, it is recommended that you maintain one VDISK map throughout each session whenever possible.

2.5 MOVING EXISTING 2200 FILES TO VDISK

To move existing 2200 files to VDISK, use the Move Files utility. You access the Move Files utility from the System Utilities menu. For more information on how to use the Move Files utility, refer to the *2200 Introductory Guide*.

CHAPTER 3
NATIVE DMS ACCESS

3.1 INTRODUCTION

The 2200/VS LCO package enables you to create and access files on the attached VS system directly from your BASIC-2 application programs. To create and access native DMS files using the 2200/VS LCO, you must use the DMS access subroutines that come with the 2200 package. Files created in this manner are referred to as native DMS files because they are managed by the VS DMS.

Native DMS files can be accessed by BASIC-2 application programs on any 2200 system attached to the VS and equipped with the 2200/VS LCO package. Native DMS files can also be accessed by VS application programs.

The 2200 LCO software supports the following DMS file types: consecutive, indexed, and relative.

Consecutive

Consecutive file types allow you to access records sequentially and read records on disk directly by record sequence number. Records can only be added at the end of the file and cannot be deleted. This structure is appropriate for most data entry and batch update applications. Consecutive files are supported for all types of I/O devices and are used for specialized purposes, such as printer files and system-maintained journals.

Indexed

Indexed file types allow you to access records through a key field that contains unique data values. Indexed files can only be created and stored on disk storage devices. This structure supports sequential record retrieval, and rapid nonsequential retrieval of single records from disk files by key value. You can add, update, or delete records by specifying the primary key value of the desired record. DMS supports both primary key and alternate key indexed files.

Relative

Relative files contain sequential, fixed length record slots and can only be created and accessed on disk storage devices. Relative files allow you to access records either sequentially or directly by record sequence number. You can add, update, or delete records within a relative file. However, you must preallocate space for adding records. Deleting records does not reduce the size of the file. You should choose a relative file structure if speed of access and ability to modify and delete existing records is a major consideration. Relative files are not supported on the VS-50 or VS-80 computers.

Note: *Relative files cannot be opened in Shared mode.*

For more information about DMS file structures, refer to the *VS Data Management System Reference*.

3.2 BRIEF DESCRIPTION OF THE DMS ACCESS SUBROUTINES

Table 3-1 lists the DMS access subroutines available, provides a description of each subroutine, and lists the subroutine's corresponding function number.

Table 3-1. DMS Access Subroutines

Function	Description	Function Number
GENERAL OPEN	Opens any DMS file	'101
GENERAL CLOSE	Closes any DMS file	'102
CONSECUTIVE READ	Reads a consecutive record	'103
CONSECUTIVE WRITE	Writes a consecutive record	'104
CONSECUTIVE REWRITE	Rewrites a consecutive record	'105
CONSECUTIVE SKIP	Skips a specified number of consecutive records	'106
CONSECUTIVE LOCK	Locks a consecutive file	'107

(continued)

Table 3-1. DMS Access Subroutines (continued)

Function	Description	Function Number
CONSECUTIVE UNLOCK	Unlocks a consecutive file	'108
INDEXED READ	Reads an indexed file	'109
INDEXED READ NEXT	Reads the next record in an indexed file	'110
INDEXED WRITE	Writes to an indexed file	'111
INDEXED REWRITE	Rewrites an indexed record to a file	'112
INDEXED DELETE	Deletes an indexed record from a file	'113
INDEXED FIND	Finds a specified indexed record in an indexed file	'114
INDEXED LOCK	Locks an indexed file	'115
INDEXED UNLOCK	Unlocks an indexed file	'116
RELATIVE READ	Reads a record form a relative file	'117
RELATIVE WRITE	Writes a record to a relative file	'118
RELATIVE REWRITE	Rewrites a record to a relative file	'119
RELATIVE DELETE	Deletes a relative record from a relative file	'120
BLOCK READ	Reads a block of data from a block file	'121
BLOCK WRITE	Writes a block of data to a block file	'122

The DMS access subroutines also provide functions for creating, deleting, and renaming files and for getting file attributes. These subroutines are called the DMS catalog functions. Table 3-2 provides a brief description of these functions.

Table 3-2. DMS Catalog Functions

Function	Function Description	Number
CREATE FILE	Creates a DMS file	'200
DELETE FILE	Deletes a DMS file	'201
RENAME FILE	Renames a DMS file	'202
GET FILE ATTRIBUTES	Retrieves the value of one or more attributes groups associated with the opened file	'203

Refer to Chapter 4 of this guide for a detailed description of the DMS access subroutines and DMS catalog functions listed above.

3.3 HOW TO USE THE DMS ACCESS SUBROUTINES

The DMS access subroutines are stored in the following files that are included with the 2200/VS LCO software:

VSACESS0 — This file contains the GENERAL OPEN and GENERAL CLOSE subroutines. This file also includes the subroutine used to communicate all requests to the 2200 LCO controller.

VSACESS1 — This file contains all the subroutines that deal with consecutive DMS files.

VSACESS2 — This file contains all the subroutines that deal with indexed DMS files.

VSACESS3 — This file contains all the subroutines that deal with relative DMS files.

VSACESS4 — This file contains all the subroutines that enable you to access DMS files in block mode.

VSACESS9 — This file contains the DMS catalog functions.

To use the DMS access subroutines, perform the following steps:

1. Copy the files containing the required DMS access subroutines into your BASIC-2 application program. Usually you are required to copy VSACESS0 and one other file into your program to open and close files and to handle all other file processing.

   ***Note:** When you copy the DMS access subroutines into your program, be certain they do not overlay lines of code in your program.*

2. Once you have copied the DMS access subroutines into your program, you access the subroutines by writing a GOSUB ' to the specific function you want to perform.

   For example, if you are working with existing consecutive files, you copy VSACESS0 and VSACESS1 into your program. The subroutines in VSACESS0 allow you to open and close. The subroutines in VSACESS1 allow you to perform other functions, such as reading and writing to and from the file.

   The following statement is an example of how you code a GOSUB ' to perform a GENERAL OPEN:

   ```
   0100 GOSUB '101 (N$, T$, M$)
   ```

   The variables (N$, T$, and M$) pass the name and organization of the file, and the mode the file is to be opened in to the subroutine.

3.4 DMS ACCESS PROGRAMMING REQUIREMENTS AND PERFORMANCE CONSIDERATIONS

To use the DMS access subroutines, you must copy the required files into your program.

To save space, copy only the files required by the program. For example, if your program uses only existing indexed files, you need only to copy in VSACESS0 and VSACESS2. If your program uses more than one file type, you have to copy more modules into your program.

If your program also creates, renames, or deletes files, you also have to copy in the DMS catalog functions VSACESS9.

Once you have copied the DMS access subroutines into your program, you can save the program with the SR parameter. Saving the program with the SR parameter removes the REM statements from the program, thus saving you partition space.

3.4.1 Submitting Programs That Access Native DMS Files

Before you can access a native DMS file, you must make certain that the following procedures have been implemented:

- The 2200 is actively connected to the VS (that the attach procedure has been run).
- The program contains the required DMS access subroutines.

For more information on how to run the attach program, refer to the *2200/VS Local Communications Options User's Guide*. Once you have implemented these procedures, you can submit programs from the 2200 to access native DMS files stored on the VS.

3.5 GENERAL NOTES ON THE DMS ACCESS SUBROUTINES

This section provides background on and general information common to all the DMS access subroutines.

3.5.1 Variables Reserved by the DMS Access Subroutines

All the variables used by the DMS access subroutines start with the letter V. If you have any variables in your programs that begin with the letter V, you can either change the variable in your program or change the variable in the DMS access subroutine.

Table 3-3 lists the variables used by the DMS access subroutines and provides a brief explanation of their purpose.

Table 3-3. Variables Used by the DMS Access Subroutines

Variable Name	Description	Length (in bytes)
V$	Holds the file name.	32
V0	Is a work variable for the CONSECUTIVE SKIP subroutine. Holds the number of records to be skipped.	8
V0$	Holds the return code. Return codes are explained in Appendix A.	2
V1	Is a work variable.	8
V1$	Holds the file organization identifier.	1
V2	Is a work variable.	8
V2$	Holds the open mode identifier.	1
V3	Holds the key position for indexed files.	8
V3$	Holds the hold option identifier.	1
V4	Holds the key path for indexed files.	8
V4$	Holds the alternate mask for indexed files.	2
V5	Holds the key length for indexed files.	8
V5$	Holds the search criteria for the INDEXED FIND subroutine.	2
V6	Holds the time out value in seconds for hold option.	8
V6$	Holds the key value for indexed files.	6

(continued)

Table 3-3. **Variables Used by the DMS Access Subroutines** (continued)

Variable Name	**Description**	**Length (in bytes)**
V7	Holds the number of records to read.	8
V7$	Holds the Extended File Sharing (EFS) header.	32
V8$	Is a work scalar variable.	16
V8$()	Is a work array.	256
V9$	Holds the file identifier number.	2
V9$()	Is the data buffer array for reading in information form the file.	4096

3.5.2 Using Variables or Literals for Input Parameters

The DMS access subroutines enable you to pass input parameters in the form of variables or literals. For example, a GOSUB ' to perform a CONSECUTIVE WRITE can be written as follows:

```
3000 DIM H1$32,D$50,H$2
3005 H$=V9$: V9$ is the file identifier received from the OPEN
3010 H1$ = V7$:REM V7$ is the EFS header received from the OPEN
3020 D$ = "This statement is written to the file as a record."
3030 GOSUB '104 (H$, H1$, D$)
```

This statement can also be written with a literal for D$ as follows:

```
3000 GOSUB '104 (H$, V7$, "This is written to the file as one
record.")
```

In this example, a literal is used for the data buffer parameter.

Note: In the example above, the DMS access subroutine variable (V7$), which stores the External File Sharing (EFS) information, is used to specify this information in the GOSUB ' call to the CONSECUTIVE WRITE subroutine. You can only use the DMS access subroutines variables within your GOSUB ' statements if your program has only one file open at a time. Otherwise, you must assign the values stored in the DMS access subroutine variables to other variables within your program, and pass the appropriate variables to the DMS access subroutines.

3.6 MOVING FILES FROM THE 2200 TO THE VS

To use the 2200/VS LCO software package to move files from the 2200 to the VS, you need to write a utility program to perform the following tasks:

1. Open the 2200 file.

2. Create the file on the VS, if the file does not exist on the VS. To create a file on the VS, you can include the CREATE FILE subroutine in your utility program or you can use the CREATE utility provided on the VS. For more information, refer to the detailed description of the CREATE FILE subroutine in Chapter 4 of this guide.

   Use the GENERAL OPEN (GOSUB '101) DMS access subroutine, to open the file, if the file already exists on the VS.

3. Read a record from the 2200 file using 2200 disk access statements.

4. Write a record to the VS using the appropriate DMS access subroutine.

5. Repeat steps 3 and 4 until all the records are written to the VS file.

6. Close the VS file using the GENERAL CLOSE DMS access subroutine (GOSUB '102).

7. Close the 2200 file.

Once you write the utility program, you must follow the procedures for submitting programs to access DMS files provided in this chapter.

3.7 MOVING FILES FROM THE VS TO THE 2200

To use the 2200/VS LCO software package to move files from the VS to the 2200, you need to write a utility program to perform the following tasks:

1. Open the DMS file on the VS using the GENERAL OPEN (GOSUB '101) DMS access subroutine.

2. Read the file using the appropriate DMS access subroutine (either consecutive, indexed, relative, or block).

3. Use the DATA SAVE DC OPEN statement, if the file does not exist on the 2200 system.

   Use the DATA LOAD DC OPEN statement, if the file already exists on the 2200 system.

4. Write the file to the 2200 using the familiar 2200 BASIC-2 statements.

Once you have written the utility program, you must follow the procedures for submitting programs to access DMS files provided in this chapter.

3.8 SAMPLE PROGRAMS

There are three sample programs included on the VSACCESS diskette shipped with the 2200/VS LCO software package. The sample programs use the DMS access subroutines to create, access and write records to and from native DMS files. Each program deals with a different file type, either consecutive, relative, or indexed. The following list provides the name of the files containing the sample programs:

TESTCON1 -- Deals with consecutive files
TESTREL1 -- Deals with relative files
TESTIDX1 -- Deals with indexed files

Once you have the software installed, you can display these files on your terminal, or print them out, to get a better idea of how to use the DMS access subroutines.

CHAPTER 4
DMS ACCESS SUBROUTINES

4.1 INTRODUCTION

This section provides a detailed description of the DMS Access Subroutines. The description of each subroutine includes the following information:

General Form -- This section shows the general format of the GOSUB' statement used and includes a description of the required input parameters.

Purpose -- This section explains the function the subroutine performs.

Returns -- This section explains information returned by the subroutine.

Example -- This section provides an example of how the subroutine is used.

General Notes -- This section is also included for certain subroutines to provide additional information.

4.2 NOTES ON THE GENERAL FORM SECTION

In the General Form section, these basic rules of syntax are followed.

1. Symbols must be included in your BASIC-2 statements exactly as they appear in the General Form of the statement:

Uppercase letters	A through Z
Comma	,
Double quotation marks	"
Parentheses	( )
Pound sign	#
Slash	/

2. Lowercase letters and words in the General Form of a statement represent items whose values must be assigned by the programmer. For example, if the lowercase word "name" appears in a General Form, the programmer must substitute a specific file name (such as "PROG1"), or an alphanumeric variable containing the name, in the actual statement. Similarly, where the lowercase letter n appears, the programmer must substitute an actual file number (from 0 to 64) or a variable containing a file number.

3. All information that appears between parentheses must be included in the GOSUB' statements.

4. Blanks (spaces) are used to improve readability and are meaningless.

5. The sequence that the terms are listed in must be followed.

Note: *The 2258 interface subroutine, '199 uses the device table designator #3. The user should include the statement, "SELECT #3/(communications address plus 3)" in the program accessing DMS subroutines. Refer to page 2-6 of the* 2200/VS Local Communication Option User's Guide *for further information.*

GENERAL OPEN

General Form

```
GOSUB '101 (file-name , org , mode)
```

where

file-name — is the name of the file. The file name can include the //SYSTEM/VOLUME/LIBRARY/FILENAME. The file name can also be written as ///VOLUME/LIBRARY/FILENAME. The SYSTEM, LIBRARY, and FILENAME can each be up to 8 characters in length. The VOLUME can be up to 6 characters in length.

org — is the organization of the file. The valid file organization parameters and their meanings follow:

- C - Consecutive file
- I - Indexed file
- R - Relative file
- B - Block Access Method

mode — is the mode the file is opened in. The valid mode parameters and their meanings follow:

- R - Read-only access
- S - Shared access
- X - Exclusive access
- E - Extended access

Purpose

The GENERAL OPEN ('101) subroutine enables you to open any DMS file. The subroutine enables you to specify the file name, including the system, the volume, the library, and actual name of the file. The library, volume, and file name are required; the system name is optional. The GENERAL OPEN subroutine also enables you to specify the file type (Indexed, Consecutive, or Relative) and the access mode (Read-Only, Shared, Exclusive, or Extended.)

Notes: *Relative files cannot be opened in Shared mode.*

If the mode specified is "Shared" for a consecutive file, the file is opened in "Shared I/O" mode.

Returns

The GENERAL OPEN subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 is the length of valid data in V9$() array (the file data field).
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$ is the file identifier assigned to the file.
- V9$() is an array of 64 or 75 bytes that contains file attribute information. Refer to Appendix A for more information.

Example (GENERAL OPEN)

```
10 DIM N$32,T$1,M$1
20 N$="//SYSNAM/ANYVOL/ANYLIB/FILENAME": T$="I", M$="S"
30 GOSUB '101 (N$, T$, M$)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN OPEN"
60 H$=V9$: REM V9$ is the file identifier
```

In this example, variables are used to represent the name, org, and mode parameters. This example could have been written as follows:

```
10 GOSUB '101 ("///ANYVOL/ANYLIB/FILENAME","I", "S")
20 IF V0$=HEX(FF) THEN 40
30 STOP "ERROR IN OPEN"
40 REM Good general open. Continue processing.
50 H$=V9$: REM V9$ is the file identifier
```

In the examples above, the specified files are opened. In the second example, the system name is replaced with a slash (/).

GENERAL CLOSE

General Form

```
GOSUB '102      (file-id , efs)
```

where

file-id — is an alphanumeric variable that represents the file identifier assigned to the file.

efs — is an alphanumeric variable that represents the EFS header information for the specified file.

Purpose

The GENERAL CLOSE ('102) subroutine enables you to close any DMS file that had been previously opened for I/O functions. You must specify the file identifier assigned to the file.

Attempting to close a file that was not previously opened by an OPEN statement causes a recoverable program error at runtime.

Returns

The GENERAL CLOSE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 is the length of valid data in V9$() array (the file data field).
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() is an array containing 3 bytes and should be HEX(00).

Example (GENERAL CLOSE)

```
10 H1$=V7$:REM V7$ is the EFS header.
20 H$=V9$: REM V9$ is the file identifier
30 GOSUB '102 (H$,H1$)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN CLOSE"
60 ...
```

In this example, a variable (H$) is used to represent the file identifier and the EFS information (H1$).

CONSECUTIVE READ

General Form

GOSUB '103 (file-id , efs , hold , time)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS information for the specified file.
hold	indicates the hold option and can be either "H", which holds the record for exclusive processing, or " ", which allows other programs to access the record.
time	is the amount of time in seconds the system waits if the record is being held by another user. The time parameter must be zero, unless the file is opened in Shared mode. Specifying a value of zero indicates that the system waits indefinitely.

Purpose

The CONSECUTIVE READ ('103) subroutine enables you to read the next consecutive record in a specified file. The file must have previously been opened.

Returns

The CONSECUTIVE READ subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains eight bytes of status information and the data read from the file. Refer to the CONSECUTIVE READ General Notes section for more information.

General Notes

In the CONSECUTIVE READ subroutine, the V9$() array is used to store status information and the data read from the file. The first eight bytes of the array are used to store the status information. Table 4-1 describes the status information.

Table 4-1. CONSECUTIVE READ Status Information

Byte(s)	Description
01	Contains internal processing information. This value should always be equal to 01.
02 and 03	Contain the number of records read.
04 and 05	Contain the number of bytes in the byte block.
06	Contains the data block ID. This value should always be equal to 01.
07 and 08	Contains the number of bytes read. The value of these two bytes can be up to 64K.

The remaining bytes of the array (starting at the ninth byte) are used to store the data read from the file. If you know the length of the records read, you know how many bytes of V9$() are used to store the data. If you do not know the length of the records in the file, or the file contains variable length records, you can use the VAL function on the 7th and 8th bytes of the V9$() array to get the length of the data read. Refer to the following example for more information.

The V9$() array is originally dimensioned to hold 4,096 bytes of information (including the status information). However, you can decrease the size of the array depending on your needs.

Example (CONSECUTIVE READ)

```
10 DIM C1$5,N1$30,B1$3,W$8
20 H$=V9$:H1$=V7$:T=25
30 W$=HEX(A005A01EA0035205)
40 GOSUB '103 (H$, H1$, "H", T)
40 IF V0$=HEX(FF) THEN 60 ELSE 50
50 STOP "ERROR IN READ"
60   $UNPACK (F=W$) STR(V9$(),9,VAL(STR(V9$(),7,2),2)) TO
     C1$,N1$,B1$,A1
```

In this example, the next consecutive record is read. If the record is being held by another user, the system waits 25 seconds before an error occurs.

In line 60 the VAL function is used on the seventh and eighth bytes of the V9$() array to determine the number of bytes read.

CONSECUTIVE WRITE

General Form

GOSUB '104 (file-id , efs , data)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the file indicated by V9$ (the file identifier).
data	can be either an alphanumeric literal or an array designator. If a literal string is used, the information must be enclosed in double quotation marks.

Purpose

The CONSECUTIVE WRITE ('104) subroutine enables you to write the next sequential record to a specified consecutive file.

To write the information contained in more than one variable to a file at one time, you can use the $PACK statement (or some other appropriate BASIC-2 statement) to pack the information into a single variable.

Returns

The CONSECUTIVE WRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following five bytes of status information:
  - Byte 01 contains internal status information. This value should always be equal to 01.
  - Bytes 02 and 03 contain the number of bytes written.
  - Bytes 04 and 05 contain the number of bytes in the byte block and should be HEX(0000).

Example (CONSECUTIVE WRITE)

```
10 DIM C1$5, N1$30, B1$3, W$8, D$43, H$2, H1$32
20 H$=V9$:H1$=V7$
30 W$=HEX(A005A01EA0035205)
40 $PACK (F=W$) D$ FROM C1$, N1$, B1$, A1
50 GOSUB '104 (H$, H1$, D$)
60 IF V0$=HEX(FF) THEN 80
70 STOP "ERROR IN WRITE"
80 ...
```

In this example, the data held in the variable D$ is written to the file specified by H$.

Note: *Since 2200 will not be able to compute the length of a variable length record with trailing spaces, the users should implicitly pass the length of the record to the CONSECUTIVE WRITE subroutine by ending the record with a non-space dummy character.*

CONSECUTIVE REWRITE

General Form

GOSUB '105 (file-id , efs , len , data)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the file indicated by V9$ (the file identifier).
len	indicates the length of the record to be rewritten. The value of the len parameter must match the record length exactly, including trailing spaces.
data	can be either an alphanumeric literal or an array designator. If a literal string is used the information must be enclosed in double quotation marks.

Purpose

The CONSECUTIVE REWRITE ('105) subroutine enables you to overwrite an existing record in a consecutive file. The record must have been previously read with the HOLD option.

To write the information contained in more than one variable to a file at one time using the REWRITE subroutine, you can use the $PACK statement to pack the information into a single variable or some other appropriate BASIC-2 statement.

Returns

The CONSECUTIVE REWRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.

- V9$() contains the following three bytes of status information:

  - Byte 01 contains the number of words in the word block that should be equal to HEX(00).

  - Bytes 02 and 03 contain the number of bytes in the byte block that should be equal to HEX(0000).

Example (CONSECUTIVE REWRITE)

```
10 DIM C1$5, N1$30, B1$3, W$8, D$43, H$2, H1$32
20 W$=HEX(A005A01EA0035205)
30 $PACK (F=W$) D$ FROM C1$, N1$, B1$, A1$
40 GOSUB '105 (H$, H1$, 43, D$)
50 IF V0$ = HEX(FF) THEN 70
60 STOP "ERROR IN REWRITE"
70 ...
```

CONSECUTIVE SKIP

General Form

GOSUB '106 (file-id , efs , nnnnnnnn)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
nnnnnnnn	is the number of records to skip; nnnnnnnn must be between 2^{32} = -2^{32}.

Purpose

The CONSECUTIVE SKIP ('106) subroutine positions a consecutive file forward or backward a given number of records in the file. For example, if the first record of a file has been read, a SKIP value of 2 causes the next record read to be record 4. A SKIP value of -1 causes the same record to be reread by the next CONSECUTIVE READ ('103). A SKIP value of 0 is ignored.

Returns

The CONSECUTIVE SKIP subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following three bytes of status information:
  - Byte 01 contains the number of words in the word block that should be equal to HEX(00).
  - Bytes 02 and 03 contain the number of bytes in the byte block that should be equal to HEX(0000).

Example (CONSECUTIVE SKIP)

```
10 GOSUB '106 (H$, H1$, 30)
20 IF V0$=HEX(FF) THEN 50:STOP "ERROR IN SKIP"
```

In this example, the next 30 records in the specified file are skipped.

CONSECUTIVE LOCK

General Form

GOSUB '107 (file-id , efs , mode)

where

file-id — is an alphanumeric variable that represents the file identifier assigned to the file.

efs — is an alphanumeric variable that represents the EFS header information for the specified file.

mode — is the mode the file is opened in. The following lists the valid mode parameters and their meanings:

- R - Read-only access
- S - Shared access
- X - Exclusive access
- E - Extended access

Purpose

The CONSECUTIVE LOCK ('107) subroutine enables you to have exclusive rights to a consecutive file. No other program can access the file until you unlock the file.

Returns

The CONSECUTIVE LOCK subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following three bytes of status information:
  - Byte 01 contains the number of words in the word block that should be equal to HEX(00).
  - Bytes 02 and 03 contain the number of bytes in the byte block that should be equal to HEX(0000).

Example (CONSECUTIVE LOCK)

```
10 H$=V9$
20 GOSUB '107 (H$, H1$, "X")
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN LOCK"
50 REM File was locked successfully.  Continue processing.
```

In this example, the "X" indicates the specified file is held for exclusive use. H1$ identifies the EFS header information.

Notes: *The file LOCK is of limited use to users of the 2200/VS Local Communication Option accessing VS/DMS files. The VS/DMS functions are available to users of 2200 through the 2200 Server running as a foreground task on one of the VS workstation emulation windows. Since all users of 2200, attached to a particular 2258 controller, send their VS/DMS requests to this server, which in turn passes the requests to VS/DMS task, the VS fails to recognise that the DMS requests are from different users of 2200.*

However, once a VS/DMS file is locked by a user on 2200 through one of the 2200 workstations, the file will not be available to other users logged on to VS until the file is unlocked. But the file can be shared for "dirty" read by other users on the VS as well as 2200.

CONSECUTIVE UNLOCK

General Form

GOSUB '108 (file-id , efs)

where

file-id — is an alphanumeric variable that represents the file identifier assigned to the file.

efs — is an alphanumeric variable that represents the EFS header information for the specified file.

Purpose

The CONSECUTIVE UNLOCK ('108) subroutine enables you to release a file from exclusive use so that other programs can access the file.

Returns

The CONSECUTIVE UNLOCK subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following three bytes of status information:
  - Byte 01 contains the number of words in the word block that should be equal to HEX(00).
  - Bytes 02 and 03 contain the number of bytes in the byte block that should be equal to HEX(0000).

Example (CONSECUTIVE UNLOCK)

```
10 H$=V9$
20 GOSUB '108 (H$, H1$)
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN UNLOCK"
50 REM Unlock was successful.  Continue processing.
```

In this example, the specified file is released from exclusive use.

INDEXED READ

General Form

```
GOSUB '109 (file-id , efs , hold , time , key , length , value)
```

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
hold	indicates the hold option and can be either "H", which holds the record for exclusive processing or " ", which allows other programs to access the record.
time	is the amount of time in seconds the system waits if the record is being held by another user. Specifying a value of zero (0) indicates that the system waits indefinitely. Nonzero values should be used only when the file is opened in Shared mode.
key	is the key path, either the primary (0) or alternate key (1-16).
length	is a numeric variable or expression that specifies the length of the key.
value	is an alphanumeric variable or literal that indicates the value of the key.

Purpose

The INDEXED READ ('109) subroutine enables you to read an indexed file. The file must have previously been opened. You can use either a primary or alternate key. For more information, see the following General Notes section.

Returns

The INDEXED READ subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 contains the length of data read (in bytes) in the V9$() array.
- V4$ is the alternate key mask.
- V9$() the data read from the file.

General Notes

DMS allows a primary key and up to 16 alternate keys. In the INDEXED READ subroutine, the primary key is indicated by a 0 and the alternate keys are indicated by the numbers 1 through 16. An example of each is provided in the following example section.

The V9$() array is originally dimensioned to hold 4096 bytes of information (including the status information). However, you can decrease the size of the array depending on your needs.

Example (INDEXED READ)

```
10 DIM C1$5, N1$30, B1$3, W$8
20 W$=HEX(A005A01EA0035205)
30 C1$="00001"
40 GOSUB '109 (H$, H1$, "H", 0, 0, 5, C1$)
50 IF V0$=HEX(FF) THEN 70
60 STOP "ERROR IN INDEXED READ"
70 REM Successful indexed read.  Continue processing
80 $UNPACK (F=W$) STR(V9$(),1,V1) to K$,N1$,B1$,A1
```

In this example, the path is identified as the primary key by the zero (0) in the key parameter position. The key length is identified as 5 characters in length, and the value of the key is equal to the value of C1$. Note also that the hold option is used.

The following example shows how to indicate an alternate key.

```
10 DIM C1$5, N1$30, B1$3, W$8
20 W$=HEX(A005A01EA0035205)
30 C1$="00001"
40 GOSUB '109 (H$, H1$, "H", 0, 4, 5, C1$)
50 IF V0$=HEX(FF) THEN 70
60 STOP "ERROR IN INDEXED READ"
70 REM Successful indexed read.  Continue processing
80 $UNPACK (F=W$) STR(V9$(),1,V1) to K$,N1$,B1$,A1
```

In this example, the path is identified as the fourth alternate key by the four (4) in the key parameter position. The other parameter values remain the same.

INDEXED READ NEXT

General Form

```
GOSUB '110 (file-id , efs , hold , time)
```

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
hold	indicates the hold option and can be either "H", which holds the record for exclusive processing or " ", which allows other programs to access the record.
time	is the amount of time in seconds the system waits if the record is being held by another user. Nonzero values should only be used when the file is opened in Shared mode.

Purpose

The INDEXED READ NEXT ('110) subroutine enables you to read an indexed file sequentially. The file must have previously been opened.

Returns

The INDEXED READ NEXT subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 contains the length of data read (in bytes) in V9$() array.
- V4$ is the alternate key mask.
- V9$() is the data read from the file.

Example (INDEXED READ NEXT)

```
10 DIM C1$5,N1$30,B1$3,W$8,K1$1,K2$5
20 W$=HEX(A005A01EA0035205)
30 GOSUB '110 (H$, H1$, "H", 25)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN READ NEXT"
60 REM Successful READ NEXT.  Continue processing
70 ...
```

In this example, the next record in the file indicated by H$ is read. If the record is being held by another user, the program waits 25 seconds before generating an error return code.

INDEXED WRITE

General Form

```
GOSUB '111 (file-id , efs , alt , data)
```

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
alt	represents the alternate key mask.
data	can be an alphanumeric, an array designator, or a literal. If a literal string is used, the information must be enclosed in double quotation marks.

Purpose

The INDEXED WRITE ('111) subroutine enables you to write a keyed record to an indexed file.

To write the information contained in more than one variable to a file at one time using the WRITE subroutine, you can use the $PACK statement (or some other appropriate BASIC-2 statement) to pack the information into a single variable.

Returns

The INDEXED WRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.

Example (INDEXED WRITE)

```
10 DIM N1$30,B1$3,W$8,K1$1,K$5
20 W$=HEX(A005A01EA0035205)
30 PACK ( F=W$ ) D$ FROM C1$, N1$, B1$, A1$
40 GOSUB '111 (H$, H1$, H2$, D$)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN INDEXED READ"
60 REM Good indexed read.  Continue processing.
```

Note: *Since 2200 will not be able to compute the length of a variable length record with trailing spaces, the users should implicitly pass the length of the record to the INDEXED WRITE subroutine by ending the record with a non-space dummy character.*

INDEXED REWRITE

General Form

```
GOSUB '112 (file-id , efs , alt , len , data)
```

where

file-id — is an alphanumeric variable that represents the file identifier assigned to the file.

efs — is an alphanumeric variable that represents the EFS header information for the specified file.

alt — represents the alternate key mask.

len — indicates the length of the record to be rewritten. The value of the len parameter must match the record length exactly, including trailing spaces.

data — can be an alphanumeric variable, an array designator, or a literal. If a literal string is used, the information must be enclosed in double quotation marks.

Purpose

The INDEXED REWRITE ('112) subroutine enables you to overwrite an existing record in an indexed file. The rewritten record size is the same as that of the existing record.

To write the information contained in more than one variable to a file at one time using the REWRITE subroutine, you can use the $PACK statement (or some other appropriate BASIC-2 statement) to pack the information into a single variable.

Returns

The INDEXED REWRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.

General Notes

A record can be rewritten only if the record is read with the hold option equal to "H".

Example (INDEXED REWRITE)

```
10 DIM C1$5,N1$30,B$3,W$8,D$43
20 W$=HEX(A005A01EA0035205)
30 $PACK ( F = W$ ) D$ FROM C1$, N1$, B1$, A1
40 GOSUB '112 (H$, H1$, H2$, 43, D$)
50 IF V0$=HEX(FF) THEN 70
60 STOP "ERROR IN INDEXED REWRITE"
70 ...
```

INDEXED DELETE

General Form

GOSUB '113 (file-id , efs)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.

Purpose

The INDEXED DELETE ('113) subroutine enables you to delete a specific record from an indexed file.

Returns

The INDEXED DELETE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.

General Notes

A record can be deleted only if the record is read with the hold option equal to "H".

Example (INDEXED DELETE)

```
10 GOSUB '113 (H$, H1$)
20 IF V0$ = HEX(FF) THEN 40
30 STOP "ERROR IN INDEXED DELETE"
40 REM Indexed delete successful.  Continue processing.
```

In this example, H$ identifies the file that the INDEXED DELETE subroutine operates on.

INDEXED FIND

General Form

```
GOSUB '114 (file-id , efs , select , post , path , length , value)
```

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
select	represents the selection criteria. The following values are valid entries for this parameter: • "8080" indicates equal to • "2080" represents greater than • "6080" indicates greater than or equal to
post	indicates the starting position of the key in the record.
path	indicates either primary (0) or alternate key (1-16) path number.
length	is a variable or numeric expression that specifies the length of the key.
value	is a variable or an alpha or numeric expression that indicates the value of the key.

Purpose

The INDEXED FIND ('114) subroutine enables you to read an indexed file based on a comparison expressed in the select input parameter to the primary or alternate key.

Returns

The INDEXED FIND subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 contains the length of data read (in bytes) in the V9$() array.
- V4$ is the alternate key mask.
- V9$() is the data read from the file.

Example (INDEXED FIND)

```
 10 DIM H$2,H1$32,H3$2,K$5
 20 K$="00003":REM KEY VALUE
 30 K1=12:REM KEY POSITION
 40 K2=5:REM KEY LENGTH
 50 K3=0:REM KEY PATH
 60 H3$=HEX(2080):REM FIND CRITERIA = GREATER THAN
 70 REM FIND RECORD WITH A KEY VALUE GREATER THAN "00003"
 80 GOSUB '114(H$, H1$, H3$, K1, K2, K3, K$)
 90 IF V0$=HEX(FF) THEN 90
100 STOP "ERROR IN INDEXED FIND"
110 REM GOOD INDEXED FIND.  CONTINUE PROCESSING."
120 ...
```

In this example, the selection criterion is set to greater than (2080). This value indicates that the next record read will have a primary key value greater than the value of C1$.

INDEXED LOCK

General Form

```
GOSUB '115 (file-id , efs , post , key , length , value)
```

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
post	represents the position in the record where the key starts.
key	is the key path, either the primary (0) or alternate key (1-16).
length	is a numeric variable or literal that specifies the length of the key.
value	is an alphanumeric variable or an alphanumeric literal that indicates the value of the key.

Purpose

The INDEXED LOCK ('115) subroutine enables you to to have exclusive rights to an indexed file. No other program can access the file until you unlock the file.

You can use either the primary or alternate key. Refer to the General Notes section for more information.

Returns

The INDEXED LOCK subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.

General Notes

DMS allows a primary key and up to 16 alternate keys. In the INDEXED READ subroutine, the primary key is indicated by a 0, the alternate keys are indicated by the numbers 1 through 16.

Example (INDEXED LOCK)

```
10 H$ = V9$
20 GOSUB '115 (H$, H1$, P, K, L, K1$)
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN LOCK"
50 REM Indexed file lock successful.  Continue processing.
60 ...
```

Notes: *The file LOCK is of limited use to users of 2200/VS Local Communication Option accessing VS/DMS files. The VS/DMS functions are available to users of 2200 through 2200 Server running as a foreground task on one of the VS workstation emulation windows. Since all users of 2200, attached to a particular 2258 controller, send their VS/DMS requests to this server, which in turn passes the requests to VS/DMS task, the VS fails to recognise that the DMS requests are from different users of 2200.*

However, once a VS/DMS file is locked by a user on 2200 through one of the 2200 workstations, the file will not be available to other users logged on to the VS untill the file is unlocked. But the file can be shared for "dirty" read by other users on the VS as well as 2200.

INDEXED UNLOCK

General Form

GOSUB '116 (file-id , efs)

where

file-id   is an alphanumeric variable that represents the file identifier assigned to the file.

efs   is an alphanumeric variable that represents the EFS header information for the specified file.

Purpose

The INDEXED UNLOCK ('116) subroutine enables you to release an indexed file so that other programs can access the file.

Returns

The INDEXED UNLOCK subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.

Example (INDEXED UNLOCK)

```
10 H$ = V9$
20 GOSUB '116 (H$, H1$)
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN UNLOCK"
50 REM Indexed file unlock successful. Continue processing.
60 ...
```

RELATIVE READ

General Form

GOSUB '117 (file-id , efs , hold , rec-num , number)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS information for the specified file.
hold	indicates the hold option and can be either "H", which holds the record for exclusive processing or " ", which allows other programs to access the record.
rec-num	is the relative number of the record to be read.
number	is a numeric variable or expression indicating the number of relative records to be read.

Purpose

The RELATIVE READ ('117) subroutine enables you to read a specified record in a relative file. The file must have previously been opened.

Returns

The RELATIVE READ subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains eight bytes of status information and the data read from the file. Refer to the General Notes section for more information.

General Notes

In the RELATIVE READ subroutine, the V9$() array is used to store status information and the data read from the file. The first eight bytes of the array are used to store the status information. Table 4-2 describes the status information.

Table 4-2. RELATIVE READ Status Information

Byte(s)	Description
01	Contains the number of words in the word block. This value should always be equal to HEX(01).
02 and 03	Contain the number of records read.
04 and 05	Contain the number of bytes in the byte block.
06	Contains the data block ID. This value should always be equal to 01.
07 and 08	Contains the number of bytes read. The value of these two bytes can be up to 64K.

The remaining bytes of the array (starting at the ninth byte) are used to store the data read from the file. If you know the length of the records read, you know how many bytes of V9$ are used to store the data. If you do not know the length of the records in the file, or the file contains variable length records, you can use the VAL function on the seventh and eighth bytes of the V9$() array to get the length of the data read. Refer to the following example for more information.

The V9$() array is originally dimensioned to hold 4096 bytes of information (including the status information). However, you can decrease the size of the array depending on your needs.

Example (RELATIVE READ)

```
10 DIM C1$5,N1$30,B1$3,W$8,H$2,H1$32
15 H$=V9$
20 W$=HEX(A005A01EA0035205)
30 GOSUB '117 (H$, H1$, "H", 125, 1)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN RELATIVE READ"
60 REM Good relative read.  Continue processing.
70 $UNPACK (F=W$) Q$() to C1$,N1$,B1$,A1
```

In this example, the 125th record of the specified file is read.

RELATIVE WRITE

General Form

GOSUB '118 (file-id , efs , number , data)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS information for the specified file.
number	is a numeric variable or expression indicating the number of relative records to be read.
data	can be an alphanumeric variable, an array designator, or a literal. If a literal string is used, the information must be enclosed in double quotation marks.

Purpose

The RELATIVE WRITE ('118) subroutine enables you to write the next record to a specified relative file.

To write the information contained in more than one variable to a file at one time, you can use the $PACK statement (or some other appropriate BASIC-2 statement) to pack the information into a single variable.

Returns

The RELATIVE WRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following five bytes of status information:
  - Byte 01 contains the number of words in the word block. This value should always be equal to HEX(01).
  - Bytes 02 and 03 contain the number of bytes written.
  - Bytes 04 and 05 contain the number of bytes in the byte block and should be HEX(0000).

Example (RELATIVE WRITE)

```
10 DIM C1$5,N1$30,B1$3,W$8,D$41,H$2,H1$32
20 W$=HEX(A005A01EA0035205)
30$PACK (F=W$) D$ FROM C1$,N1$,B1$,A1
40 GOSUB '118 (H$, H1$, 1, D$)
50 IF V0$=HEX(FF) THEN 70
60 STOP "ERROR IN RELATIVE WRITE"
70 REM Good relative write.  Continue processing.
80 ...
```

In this example, the value of D$ is written to the file as one record.

RELATIVE REWRITE

General Form

GOSUB '119 (file-id , efs , number , len , data)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
number	represents the number of the relative record to be rewritten.
len	indicates the length of the record to be rewritten. The value of the len parameter must match the record length exactly, including trailing spaces.
data	can be an alphanumeric variable, an array designator, or a literal. If a literal string is used, the information must be enclosed in double quotation marks.

Purpose

The RELATIVE REWRITE ('119) subroutine enables you to overwrite an existing record in a relative file. The record must have been previously read with the HOLD option.

To write the information contained in more than one variable to a file at one time using the REWRITE subroutine, you can use the $PACK statement to pack the information into a single variable or some other appropriate BASIC-2 statement.

Returns

The RELATIVE REWRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following three bytes of status information:
  - Byte 01 contains the number of words in the word block and should be equal to HEX(00).
  - Bytes 02 and 03 contain the number of bytes written and should be equal to HEX(0000).

Example (RELATIVE REWRITE)

```
10 DIM C1$5,N1$30,B1$3,W$8,D$43,H$2,H1$32
20 W$=HEX(A005A01EA0035205)
30 $PACK (F=W$) D$ FROM C1$,N1$,B1$,A1
40 GOSUB '104 (H$, H1$, 5, 43, D$)
50 IF V0$=HEX(FF) THEN 70
60 STOP "ERROR IN REWRITE"
70 REM Good relative rewrite.  Continue processing.
80 ...
```

In this example, the fifth relative record of the specified file is rewritten to the file.

RELATIVE DELETE

General Form

GOSUB '120 (file-id , efs , number)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
number	represents the number of the relative record to be deleted.

Purpose

The RELATIVE DELETE ('120) subroutine enables you to delete a specific record from a relative file.

Returns

The RELATIVE DELETE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following three bytes of status information:
  - Byte 01 contains the number of words in the word block and should be equal to HEX(00).
  - Bytes 02 and 03 contain the number of bytes written and should be equal to HEX(0000).

Example (RELATIVE DELETE)

```
10 H$ = V9$:H1$ = V7$
20 GOSUB '120 (H$, H1$, 5)
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN DELETE"
50 REM Good relative record delete.  Continue processing.
60 ...
```

In this example, H$ indicates the file that the fifth relative record is deleted from.

BLOCK READE

General Form

GOSUB '121 (file-id , efs , block-num)

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
block-num	represents the block number of the block to be read.

Purpose

The BLOCK READ ('121) subroutine enables you to read a 2K block of information from a consecutive, indexed, or relative file.

Returns

The BLOCK READ subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V9$() contains the data read from the file. This array is 2048 bytes long.

General Notes

To use the BLOCK READ the file organization must be specified as Block Access Method when the file is opened. The file cannot be opened in Shared mode. Each block read contains 2048 bytes of data.

Example (BLOCK READ)

```
10 H$ = V9$:H1$ = V7$:B1=5
20 GOSUB '121 (H$, H1$, B1):REM Read block number 5.
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN BLOCK READ"
50 REM Good block read.  Continue processing.
60 ...
```

In this example, block number five of the specified file is read.

BLOCK WRITE

General Form

```
GOSUB '122 (file-id , efs , block-num , data)
```

where

file-id	is an alphanumeric variable that represents the file identifier assigned to the file.
efs	is an alphanumeric variable that represents the EFS header information for the specified file.
block-num	represents the block number of the block to be written.
data	can be either an alphanumeric literal or an array designator. If a literal string is used, the information must be enclosed in double quotation marks.

Purpose

The BLOCK WRITE ('122) subroutine enables you to write a 2K block of information to a consecutive, indexed, or relative file.

Returns

The BLOCK WRITE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.

General Notes

To use the BLOCK READ the file organization must indicate Block Access Method when the file is opened. Each block contains 2048 bytes of data. When you use the BLOCK WRITE subroutine, you want to write approximately 2048 bytes of data to the file.

Example (BLOCK WRITE)

```
10 H$ = V9$:H1$ = V7$:B1=5
20 GOSUB '122 (H$, H1$, B1, D$):REM Write block number 5.
30 IF V0$=HEX(FF) THEN 50
40 STOP "ERROR IN BLOCK WRITE"
50 REM Good block write.  Continue processing.
60 ...
```

In this example, block number five is written to the specified file.

CREATE FILE

General Form

GOSUB '200 (file-name , org , mode , opt-flag , create , alt-key)

where

file-name — represents the name of the file. The file name can include the //SYSTEM/VOLUME/LIBRARY/FILENAME. The file name can also be written as ///VOLUME/LIBRARY/FILENAME. The SYSTEM, LIBRARY, and FILENAME can be up to 8 characters in length. The VOLUME can be up to 6 characters in length.

org — is the organization of the file. The following lists the valid file organization parameters and their meanings:

- C - Consecutive file
- I - Indexed file
- R - Relative file
- B - Block Access Method

mode — is the mode the file is opened in. The following lists the valid mode parameters and their meanings:

- R - Read only access
- S - Shared access
- X - Exclusive access
- E - Extended access

opt-flag — indicates whether the file is to be created as follows:

- C - Created
- T - Temporary
- O - Created and opened

create — specifies the attribute data for the file. Refer to the following General Notes section for more information.

alt-key — specifies the alternate key information if required. Refer to the following General Notes section for more information.

Purpose

The CREATE FILE ('200) subroutine enables you to create a DMS file. The subroutine enables you to specify the file name, including the system, the library, and actual name of the file. The library, volume, and file name are required; the system name is optional. The CREATE FILE subroutine also enables you to specify the file type (indexed, consecutive, or relative,) and the access mode (Read-Only, Shared, Exclusive, or Extended).

Returns

The CREATE FILE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the following three bytes of status information:
  - Byte 01 contains the number of words in the word block that should be equal to HEX(00).
  - Bytes 02 and 03 contain the number of bytes in the byte block that should be equal to HEX(0000).

General Notes

You can use the VS CREATE utility to create and maintain data files. Through CREATE you can add, delete, modify, or examine data in the data files created using the utility. For more information on the CREATE utility, refer to the *VS File Management Utilities Reference*.

The create input parameter requires 40 bytes of information for consecutive, indexed, and relative files. If the file being created is an indexed file with one or more alternate keys, an additional eight bytes of information is required for each alternate key. This information is passed to the subroutine in the alt-key input parameter.

If the file being created does not contain an alternate key, the alt-key parameter should be set to spaces and must still be passed to the subroutine.

For more information on the contents of the create and alt-key parameters, refer to Appendix A.

Example (CREATE FILE)

```
10 DIM N$32, T$1, M$1, A$40, A1$8
20 N$="//SYSTEM/ANYVOL/ANYLIB/FILENAME"
30 T$="I"
40 M$="S"
50 GOSUB '200 (N$, T$, M$, "O", A$, A1$)
60 IF VO$=HEX(FF) THEN 80
70 STOP "ERROR IN CREATE"
80 REM Good file create.  Continue processing.
90 ...
```

In this example, variables are used to represent the name, organization, mode, file attributes, and alternate key attribute data parameters. The opt-flag parameter value of "O" indicates that the file is created and opened.

DELETE FILE

General Form

GOSUB '201 (file-name)

where

file-name — represents the name of the file. The file name can include the //SYSTEM/VOLUME/LIBRARY/FILENAME. The file name can also be written as ///VOLUME/LIBRARY/FILENAME. The SYSTEM, LIBRARY, and FILENAME can be up to 8 characters in length. The VOLUME can be up to 6 characters in length.

Purpose

The DELETE FILE ('201) subroutine enables you to delete any DMS file.

Returns

The DELETE FILE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 is the length of valid data in the V9$() array.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains three bytes of internal status information and should be equal to three HEX zeros (HEX(000000)).

Example (DELETE FILE)

```
10 DIM N$32
20 N$="///ANYVOL/ANYLIB/FILENAME"
30 GOSUB '201 (N$)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN DELETE FILE"
60 REM File successfully deleted.  Continue processing.
70 ...
```

In this example, the file specified by N$ (///ANYVOL/ANYLIB/FILENAME) is deleted from the system.

RENAME FILE

General Form

GOSUB '202 (old-name , new-name)

where

old-name	represents the name of the file as it currently exists on the system. The file name can include the //SYSTEM/VOLUME/LIBRARY/FILENAME. The file name can also be written as ///VOLUME/LIBRARY/FILENAME. The SYSTEM, LIBRARY, and FILENAME can be up to 8 characters in length. The VOLUME can be up to 6 characters in length.
new-name	represents the name of the file as it will be known to the system after the subroutine executes. The file name can include the //SYSTEM/VOLUME/LIBRARY/FILENAME. The file name can also be written as ///VOLUME/LIBRARY/FILENAME. The SYSTEM, LIBRARY, and FILENAME can be up to 8 characters in length. The VOLUME can be up to 6 characters in length.

Purpose

The RENAME FILE ('202) subroutine enables you to rename any DMS file.

Returns

The RENAME FILE subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 is the length of valid data in the V9$() array.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains three bytes of internal status information and should be equal to three HEX zeros (HEX(000000)).

Example (RENAME FILE)

```
10 DIM O#32, N$32
20 O$="///OLDVOL/OLDLIB/OLDNAME"
30 N$="///NEWVOL/NEWLIB/NEWNAME"
40 GOSUB '202 (O$, N$)
50 IF V0$=HEX(FF) THEN 70
60 STOP "ERROR IN RENAME FILE"
80 REM File successfully renamed.  Continue processing.
70 ...
```

In this example, the file specified by N$ (///ANYVOL/ANYLIB/FILENAME) is deleted from the system.

GET FILE ATTRIBUTES

General Form

GOSUB '203 (file-name)

where

file-name — represents the name of the file. The file name can include the //SYSTEM/VOLUME/LIBRARY/FILENAME. The file name can also be written as ///VOLUME/LIBRARY/FILENAME. The SYSTEM, LIBRARY, and FILENAME can be up to 8 characters in length. The VOLUME can be up to 6 characters in length.

Purpose

The GET FILE ATTRIBUTES ('203) subroutine enables you to retrieve the value of one or more attributes associated with the specified file. The file must be opened first.

Returns

The GET FILE ATTRIBUTES subroutine returns the following information:

- V0$ is the return code. Refer to Appendix A for more information.
- V1 is the length of valid data in V9$() array.
- V7$ is the EFS header information for the file indicated by V9$ (the file identifier). Refer to Appendix A for more information.
- V9$() contains the file attribute information returned. Refer to Appendix A for an explanation of the attributes returned.

Example (GET FILE ATTRIBUTES)

```
10 DIM N$32
20 N$="///ANYVOL/ANYLIB/FILENAME"
30 GOSUB '203 (N$)
40 IF V0$=HEX(FF) THEN 60
50 STOP "ERROR IN GET FILE ATTRIBUTES"
60 REM Good read on file attributes.  Continue processing.
70 ...
```

In this example, the file attributes for the file ///ANYVOL/ANYLIB/FILENAME are returned and held in the V9$() array.

APPENDIX A
ADDITIONAL INFORMATION

A.1 INTRODUCTION

This appendix contains the following additional information on:

- The return codes
- The file attributes returned by subroutines, GENERAL OPEN and GET FILE ATTRIBUTE
- The create and alternate key input parameters for the CREATE FILE subroutine

A.2 RETURN CODES

The return codes generated by the DMS Access Subroutines are in HEX format. Table A-1 lists and explains the return codes generated by the DMS Access Subroutines.

The EFS header explained in Section A.3 provides additional information on the error.

Table A-1. Return Codes

Return Decimal	Code Hex	Description	Recovery Action
00	00	Syntax error	Check your program for possible syntax error.
90	5A	2258 firmware error	Ensure the 2200/VS LCO task is operational and the DMS task has been assigned.
165	A5	Error in SMB block	Refer to Table A-3.
255	FF	Successful	Always check EFS header for any possible error.

A.3 EXTENDED FILE SHARING (EFS) HEADER

The DMS access subroutines use the Extended File Sharing (EFS) protocol to control file information. The EFS header is 32 bytes long. The DMS access subroutines store the EFS header in variable V7$.

Table A-2 describes the EFS header in detail.

Table A-2. EFS Header

Byte(s)	Description	Initial Value (in HEX)
01 through 04	Contain header identification.	FF534D42
05	Mandatory; values other than HEX(FF) indicate error.	FF
06	Returns error class; refer to the section on Error Classes in this Appendix for further information.	00o
07	DMS Access Subroutines store the extended command code in this byte.	(Initialized by subroutine invoked.)
08 and 09	Returns two-byte error code; refer to the section on Error Codes for further information.	0000
10	Reserved for future use.	00
11 through 22	Reserved for future use.	All zeroes
23 and 24	Mandatory.	FFFF
25 and 26	Reserved for future use.	0000
27 and 28	Reserved for future use.	0000
29 and 30	These bytes are updated by the GENERAL OPEN subroutine with the User Identification number.	0000
31 and 32	Reserved for future use.	0000

A.3.1 Error Classes

The error class indicates which DMS Server task was invoked when the error occurred. Table A-3 lists and explains the values returned for the error classes.

Table A-3. Error Classes

Error Class		Explanation/Indication
Decimal	HEX	
07	07	Reserved for 2200SRV
08	08	Catalog server
09	09	File server
10	0A	WITA server
11	0B	Print server
12	0C	QLI server
13	0D	QLI:FORMATER server
14	0E	QUEUE:JOB server
15	0F	DMPACK server
20	14	User server

A.3.2 Error Codes

The error codes explain why the error has occurred. Table A-4 describes the error codes returned.

Table A-4. Error Codes

Error Codes		
Decimal	**Hex**	**Description**
000	0000	Successful execution of requested function.
001	0001	Invalid function specified to the server.
002	0002	File not found or there is a duplicate file.
003	0003	Library not found.
004	0004	Too many files have been opened.
005	0005	The user has insufficient access rights.
006	0006	Invalid handle has been supplied.
007	0007	Server processing error.
008	0008	Insufficient space allocated to perform the function.
009	0009	VTOC error.
010	000A	The parameters passed are invalid for the function.
011	000B	Invalid file format.
012	000C	Open access mode specified is invalid.
013	000D	Disk space or extents error.
014	000E	Invalid function has been specified for I/O mode.

(continued)

Table A-4. Error Codes (continued)

Error Codes		
Decimal	**Hex**	**Description**
015	000F	Volume required is not mounted.
016	0010	Delete errors encountered.
017	0011	Invalid device specified.
018	0012	NODATA Read attempted on a file opened in Shared mode.
019	0013	Invalid function attempted on a Relative file or Write attempted on a Read-Only file.
020	0014	The file already exists.
021	0015	File possession conflict.
022	0016	Invalid key size specified.
023	0017	Invalid key value specified.
024	0018	Boundary violation has occurred.
025	0019	End of file.
026	001A	Invalid attempt to REWRITE a compressed record.
028	001C	Invalid alternate key specified.
029	001D	Invalid function specified for alternate indexed file.
030	001E	Permanent I/O error has occurred.
031	001F	Undefined position specified for READ NEXT function.

(continued)

Table A-4. Error Codes (continued)

Error Codes		
Decimal	Hex	Description
032	0020	Disk problems encountered.
033	0021	Recovery problems encountered.
034	0022	File organization needs to be specified.
036	0024	Invalid WRITE issued to Relative or Read-Only file.
037	0025	Invalid function for a file opened in Shared mode.
038	0026	Invalid function for a file opened in non-Shared mode.
039	0027	Invalid function for PAM access method.
040	0028	The requested device is in use.
041	0029	The requested device is detached.
042	002A	Access denied.
043	002B	Invalid function sequence for Delete, BAM access method, and REWRITE on consecutive file.
044	002C	START WAIT has been issued when no I/O is pending.
045	002D	No wait was issued for previous I/O.
046	002E	Timeout on a shared resource wait.
047	002F	Indexed file is requested but FDR indicates it is not an indexed file.
048	0030	Relative files may not be compressed.

(continued)

Table A-4. Error Codes (continued)

Error Codes		
Decimal	Hex	Description
049	0031	The file is locked by another task.
051	0033	Read/Write issued to a file temporarily locked.
052	0034	Invalid name.
054	0036	File is already open.
055	0037	OPEN in EXTENDED mode is invalid for indexed file, but the file may be created.
056	0038	The file cannot be closed.
057	0039	The record count is not updated during Close.
059	003B	There is no space on SHARED block.
081	0051	The server is paused.
096	0060	Severe DMS errors encountered.
098	0062	Task problems encountered; restart the task indicated by error class.
099	0063	Nonspecific file system error encountered.
100	0064	The system name cannot be parsed.
102	0066	Length inconsistency, VS returned fewer bytes than expected.

A.4 FILE ATTRIBUTE INFORMATION

This section describes the file attribute information returned by the subroutine, GENERAL OPEN ('101) or GET FILE ATTRIBUTES ('203).

A.4.1 Attribute Structure for General Open

The structure of the file attribute information returned by GENERAL OPEN ('101) is shown in Table A-5.

Table A-5. GENERAL OPEN - File Attribute Information Structure

Byte(s)	Description	Remarks
01 and 02	Count of data bytes	
03	Mandatory data block identification	HEX(01)
04 and 05	Size of this data block	
06 through 144	General file attribute data	Refer to Table A-7
145	Mandatory data block identification	HEX(01)
146 and 147	Size of this data block	
148 onwards	Alternate key file attribute data (8 bytes of information for each alternate key defined for the file)	Refer to Table A-8

Notes: *Add a displacement of 6 bytes to the General File Attribute Data described in Table A-7.*

Add a displacement of 148 bytes to the Alternate Key File Attribute Data described in Table A-8.

A.4.2 Attribute Structure for Get File Attributes

The structure of the file attribute information returned by GET FILE ATTRIBUTES ('203) is shown in Table A-6.

Table A-6. GET FILE ATTRIBUTES - File Attribute Information Structure

Byte(s)	Description	Remarks
01	Mandatory word counter	HEX(00)
02 and 03	Count of data bytes	
04	Mandatory data block identification	HEX(01)
05 and 06	Size of this data block	
07 through 145	General file attribute data	Refer to Table A-7

Note: *Add a displacement of 5 bytes to the General File Attribute Data described in Table A-7.*

A.4.3 Attribute Structure for General File Attributes

Table A-7 describes the general file attribute data returned by GENERAL OPEN ('101) as well as GET FILE ATTRIBUTES ('203).

Table A-7. General File Attribute Data

Byte(s)	Description
01	The file organization: C for Consecutive, I for Indexed, A for alternate indexed, P for print, W for WP, O for object, R for Relative or B for Block.
02	The record type: F for fixed length records or V for variable length records.
03	The compression flag: Y indicates the file is compressed and N indicates the file is not compressed.
04	The file class: A-Z, #, $ or @.
05 through 08	The number of records in the file.
09 through 12	The size of the record.
13 through 16	The number of extents of disk space allocated.
17 through 20	The number of blocks used.
21 through 24	The number of blocks allocated for the file.
25 through 32	The identification of the creator of the file.
33 through 38	The date of creation (YYMMDD) of the file.
39 through 44	The date of modification (YYMMDD) of the file.
45 through 50	The date of expiration (YYMMDD) of the file.
51	Whether WP prologue sector is present; Y for yes N for no.
52 and 53	The position of primary key for the indexed file. This value is HEX(0000) for non-indexed file.

(continued)

Table A-7. General File Attribute Data (continued)

Byte(s)	Description
54 and 55	The length of the primary key for the indexed file. This value is HEX(0000) for a non-indexed file.
56 and 57	The number of alternate keys for the indexed file. An indexed file can have 1 to 16 alternate keys. This value is HEX(0000) for a non-indexed file.
58 and 59	The alternate key mask for the indexed file. This value is HEX(0000) for a non-indexed file.
60 through 85	Write access (W) for the different file classes (A-Z).
86 through 111	Read access (R) for the different file classes (A-Z).
112 through 137	Execute access (E) for the different file classes (A-Z).
138 and 139	Number of records in the last block.

A.4.4 Attribute Structure for Alternate Key File Attributes

Table A-8 describes the alternate key file attribute data returned by GENERAL OPEN ('101) subroutine.

Table A-8. Alternate Key File Attribute Data

Byte(s)	Description
01	Y indicates records with duplicate alternate key values are allowed. N indicates duplicates are not allowed.
02	Y indicates the alternate key is compressed. N indicates the alternate key is not compressed.
03 and 04	The ordinal number of the alternate key ranges from 1 to 16.
05 and 06	The starting position of the alternate key.
07 and 08	The length in bytes of the alternate key.

A.5 CREATE FILE INFORMATION

This section describes the information required for the create input parameter and the alternate key input parameter for the CREATE FILE ('200) subroutine.

A.5.1 Create File Parameters

The create input parameter for the CREATE FILE ('200) requires 41 bytes of information. Table A-9 describes the create input parameter.

Table A-9. Create Input Parameter

Byte(s)	Description
01	The file organization: C for Consecutive, I for Indexed, R for Relative or B for Block.
02	The record type: F for fixed length records or V for variable length records.
03	The file class: A-Z, #, $ or @.
04	The compression flag: Y indicates the file is to be compressed and N indicates the file is not to be compressed.
05 through 08	The approximate number of records in the file.
09 through 12	The size of the record.
13 through 16	The number of blocks to be allocated for the file.
17 through 20	The size of the block.
21 through 26	The date of expiration (YYMMDD) of the file.
27	Whether WP prologue sector is present; Y for yes N for no.
28 and 29	The position of primary key for the indexed file. This value is HEX(0000) for non-indexed file.
30 and 31	The length of the primary key for the indexed file. This value is HEX(0000) for non-indexed file.

(continued)

Table A-9. Create Input Parameter (continued)

Byte(s)	Description
32 and 33	The number of alternate keys for the indexed file. An indexed file can have 1 to 16 alternate keys. This value is HEX(0000) for a non-indexed file.
34	Whether to take the created space. N indicates no.
35	For print files only; otherwise HEX(00). Indicates print form number. Print form numbers are established by the user site operations.
36 and 37	For print files only; otherwise HEX(0000). Indicates printer device number. Printer device numbers are established by the user site operations.
38	For print files only; otherwise HEX(00). Indicates print class. Print classes are established by the user site operations.
39	For print files only; otherwise HEX(00). Indicates number of copies to be printed.
40	For print files only; otherwise HEX(00). Indicates the status; either R for release or H for hold.
41	For print files only; otherwise HEX(00). Indicates the disposition; either R for release or H for hold.

A.5.2 Alternate Key Structure

An additional 8 bytes of information for each alternate key is required for the alt-key input parameter of the CREATE FILE ('200) subroutine. Table A-10 describes the alternate key input parameter for indexed files with alternate keys.

Table A-10. Alternate Key Input Parameter

Byte(s)	Description
01	Y indicates records with duplicate alternate key values are allowed. N indicates duplicates are not allowed.
02	Y indicates the alternate key is to be compressed. N indicates the alternate key is not to be compressed.
03 and 04	The ordinal number of the alternate key ranges from 1 to 16.
05 and 06	The starting position of the alternate key.
07 and 08	The length in bytes of the alternate key.

INDEX

B

BLOCK READ, 3-3, 4-36
BLOCK WRITE, 3-3, 4-37

C

Consecutive file, 3-1
CONSECUTIVE LOCK, 3-2, 4-14, 4-15
CONSECUTIVE READ, 3-2, 4-6 to 4-8
CONSECUTIVE REWRITE, 3-2, 4-11, 4-12
CONSECUTIVE SKIP, 3-2, 4-13
CONSECUTIVE UNLOCK, 3-3, 4-16
CONSECUTIVE WRITE, 3-2, 4-9, 4-10
CREATE FILE, 3-4, 4-38 to 4-40,
    A-14 to A-16

D

Data Management System (DMS), 1-1,
    4-17, 4-32
DELETE FILE, 3-4, 4-41
Disk address
  modifying the, 2-2
  not recognized, 2-3
DMS, *see* Data Management System
DMS access, 3-1
  catalog functions, 3-4
  performance considerations, 3-5,
    3-6
  programming requirements, 3-5
    3-6
  sample programs, 3-10
  submitting programs, 3-6
DMS access subroutines, 3-2 to 3-4,
    4-1 to 4-44
  general notes on, 3-6
  using, 3-4 to 3-5, 4-1, 4-2
  using literals with, 3-8
  using variables with, 3-8
  variables reserved for, 3-6 to 3-8
DMS files, 1-2
  access, 1-2, 3-1 to 3-10
  file types, 3-1, 3-2

E

EFS, *see* Extended File Sharing
    Header
error classes, A-4
error codes, A-5 to A-8
Exclusive mode, 2-1
Extended File Sharing Header, A-2,
    A-3

F

File attribute, A-9 to A-13
Filing services utility, 1-1

G

GENERAL CLOSE, 3-2, 4-5
GENERAL OPEN, 3-2, 4-3, 4-4, A-8
    to A-11
GET FILE ATTRIBUTES, 3-4, 4-44,
    A-9 to A-12
GOSUB '101, 3-2, 4-3, 4-4, A-8 to
    A-11

GOSUB '102, 3-2, 4-5
GOSUB '103, 3-2, 4-6 to 4-8
GOSUB '104, 3-2, 4-9, 4-10
GOSUB '105, 3-2, 4-11, 4-12
GOSUB '106, 3-2, 4-13
GOSUB '107, 3-2, 4-14, 4-15
GOSUB '108, 3-3, 4-16
GOSUB '109, 3-3, 4-17, 4-18
GOSUB '110, 3-3, 4-19
GOSUB '111, 3-3, 4-20
GOSUB '112, 3-3, 4-21
GOSUB '113, 3-3, 4-22
GOSUB '114, 3-3, 4-23, 4-24
GOSUB '115, 3-3, 4-25, 4-26
GOSUB '116, 3-3, 4-27
GOSUB '117, 3-3, 4-28 to 4-30
GOSUB '118, 3-3, 4-31, 4-32
GOSUB '119, 3-3, 4-33, 4-34
GOSUB '120, 3-3, 4-35
GOSUB '121, 3-3, 4-36
GOSUB '122, 3-3, 4-37
GOSUB '200, 3-4, 4-38 to 4-40
    A-14 to A-16
GOSUB '201, 3-4, 4-41
GOSUB '202, 3-4, 4-42, 4-43
GOSUB '203, 3-4, 4-44
    A-9 to A-12

I

INDEXED DELETE, 3-3, 4-22
Indexed file, 3-1
INDEXED FIND, 3-3, 4-23, 4-24
INDEXED LOCK, 3-3, 4-25, 4-26
INDEXED READ, 3-3, 4-18
INDEXED READ NEXT, 3-3, 4-19
INDEXED REWRITE, 3-3, 4-21
INDEXED UNLOCK, 3-3, 4-27
INDEXED WRITE, 3-3, 4-20

L

Local Communications Option (LCO),
    1-1, 4-17, 4-32

M

Moving files
  DMS, 3-9, 3-10
  VDISK, 2-6

N

Native DMS, 1-2, 3-1
  access, 3-1
  files, 1-2

P

Performance
  VDISK, 2-3, 2-4, 2-5
  DMS, 3-5

R

Read Only mode, 2-1
Relative file, 3-2
RELATIVE DELETE, 3-3, 4-35
RELATIVE READ, 3-3, 4-28 to 4-30
RELATIVE REWRITE, 3-3, 4-33, 4-34
RELATIVE WRITE, 3-3, 4-31, 4-32
RENAME FILE, 3-4, 4-42, 4-43
Return codes, A-1, A-2

S

Sample programs, 3-10
Shared mode, 2-1
Submitting programs
  DMS, 3-6
  VDISK, 2-3

V

VDISK, 1-2, 2-1
  accessing, 2-1
  creating, 2-1
  improving performance, 2-4, 2-5
  mode, 2-1
  moving files to, 2-6
  performance considerations, 2-3
    to 2-5
  using, 2-2, 2-3

WANG

To Order by Phone, Call:
1-(800) TEL-WANG

ORDER FORM FOR WANG MANUALS AND DOCUMENTATION

Customer Number ______________________________

Bill To ______________________________

____

____

Ship To ______________________________

____

____

Customer Contact ______________ Phone ______________

Date ______________ Purchase Order Number ______________

Taxable ☐ Yes ☐ No Tax Exemption No. ______________

Salesperson ______________ Empl. No. ________ RDB No. ______________

Document Number	Description	Qty.	Unit Price	Total Price
			Subtotal	
			Less Applicable Discount	
			Subtotal	
			Local Sales Tax	
			Total Amount	

____

Authorized Signature **Date**

☐ Check this box if you would like a free copy of the ***Wang Express Software & Literature Catalog*** **(711-0888).**

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 give 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 Terms and Conditions

1. **TAXES** – Prices are exclusive of all sales, use, and like taxes.
2. **DELIVERY** – Delivery will be F.O.B. Wang plant. The customer will be billed for freight charges, and unless the 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 the date of invoice. Unless otherwise stated by the 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 **Wang Express Software & Literature Catalog** (711-0888).
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.

Printed in U.S.A. 14-3141B 4-90

WANG

Cut along dotted line

Fold here

WANG

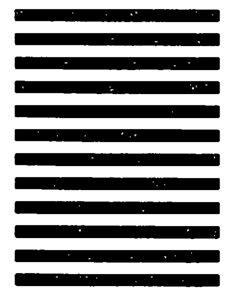
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG EXPRESS MS 017-110
WANG LABORATORIES INC
1005 PAWTUCKET BOULEVARD
LOWELL MA 01854-9984

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES



Fold here

Customer Comment Form

Publication Number 715-0562A

Title 2200/VS LOCAL COMMUNICATIONS OPTION PROGRAMMER'S REF. GD.

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
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.

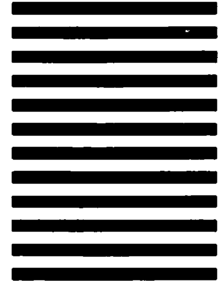
All comments and suggestions become the property of Wang Laboratories, Inc.

WANG

Cut along dotted line

Fold here

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES



WANG

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

TECHNICAL PUBLICATIONS MS 012-260
WANG LABORATORIES INC
ONE INDUSTRIAL AVENUE
LOWELL MA 01851-9971

Fold here

Corporate Headquarters

Wang Laboratories, Inc.
1 Industrial Avenue
Lowell
MA 01851
U.S.A.

Tel: (508) 459-5000

Europe

Wang Belgium N.V./S.A.
Zweefvliegtuigstraat 10 Rue du Planeur
Brussel 1130 Bruxelles

Tel: (02) 244.22.11

Wang Deutschland GmbH
Lyoner Straße 26
Postfach 710570
6000 Frankfurt am Main 71

Tel: 069/66750

Wang España S.A.
Autopista Aeropuerto
Barajas KM 13
28042 Madrid

Tel: (91) 337-1100

Wang France S.A.
Tour Galliéni 1
78/80, Av. Galliéni
93174 BAGNOLET CEDEX

Tel: (1)48.97.06.06

Wang Ireland Ltd.
Harcourt Centre
Harcourt Street
Dublin 21

Tel: 01-757-931

Wang Italia SPA
Centro Terziario
Strada Statale Padana Superoire
Milano

Tel: (02) 250-4021

Wang Nederland B.V.
P.O. Box 4
4100 AA Culemborg

Tel: (03) 450-70911

Wang Österreich GmbH
Linke Wienzeile 234
A-1150 Wien

Tel: (0222) 85 85 33

Wang (Schweiz) AG
Talackerstraße 7
8152 Glattbrugg/Zürich

Tel: (01) 829-7111

Wang (Suisse) SA
Air Center Blandonnet
16, Chemin des Cloquelicote
1214 Vernier-Genève

Tel: 022-41-36-00

Wang Svenska AB
Box 1196
S-171 23 Solna

Tel: 08-705 85 00

Wang (UK) Ltd.
1000 Great West Road
Brentford
Middlesex TW8 9HL

Tel: 01 568 4444

Americas

Wang Canada Ltd.
66 Leek Crescent
Richmond Hills
Ontario L4B 1J7
Tel: (416) 764-1999

Wang De Mexico, S.A. De C.V.
Avenida Paseo De La Reforma, 295
3er Piso, CP 06500
Mexico, D.F.
Tel: (525) 207-5111

Computadoras Wang De Panama S.A.
Apartado 6425, Zona 5
Samuel Lewis y Gerardo Ortega
Panama 5
Tel: (507) 635-566

Wang Computadoras, Inc. (Puerto Rico)
Metro Office Park
Call Box 2106
Caparra Heights
Puerto Rico 00922
Tel: (809) 793-9264

Wang Laboratories (U.S.A.) Inc.
1 Industrial Avenue
Lowell
MA 01851
Tel: (508) 459-5000

Asia/Pacific

Wang Australia Pty. Ltd.
Northside Gardens
168 Walker Street
North Sydney
N.S.W. 2060
Tel: (61) 2-925-5678

Wang Computer China Ltd.
Office 1, 23/F, CITIC Bldg.
19 Jianguomenwai Street
Beijing
Tel: (86) 1-500-2255 Rm. 2310

Wang Pacific (Hong Kong) Ltd.
31/F Hennessy Centre
500 Hennessy Road
Hong Kong
Tel: (852) 805-7333

Wang Computer (Japan) Ltd.
Hazama Building 14F
5-8 Kita Aoyama 2-Chome
Minato-Ku
Tokyo
Tel: (81) 3-478-2870

Wang Computer Korea Ltd.
46/47th Floor, DLI 63 Bldg.
60, Yoida-Dong
Youngdeungpo-ku
Seoul 150
Tel: (82) 2-784-6111

Wang New Zealand Ltd.
Wang Terraces
9 City Road
Auckland
Tel: (64) 9-796-372

Wang Computers (Singapore) (Pte) Ltd.
101 Thomson Road 12-00
12-00 United Square
Singapore 1130
Tel: (65) 250-9595

Wang Industrial Co. (Taiwan) Ltd.
8/F 56 Tun Hwa N. Road
Taipei
Tel: (886) 2-721-6121

WANG

ONE INDUSTRIAL AVENUE, LOWELL, MA 01851
TEL. (508) 459-5000, TELEX 172108

Printed in U.S.A. 715-0562A 5-91