StorHouse®

Callable Interface Programmer's Guide

Release 1.8 of the
StorHouse Host Software

Publication Number
900013 Rev. Q
June 14, 2004

FileTek

No part of this publication may be reproduced, translated, stored in any electronic retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of FileTek, Inc.

This publication Copyright © 2004 by FileTek, Inc. As an Unpublished Licensed Work. All Rights Reserved. Publication Number: 900013 Rev. Q

NOTICE: U.S. GOVERNMENT USERS
This notice applies to all acquisitions of this work by or for the U.S. Government ("Government"), or by any prime contractor or subcontractor (at any tier) under any contract, cooperative agreement or other activity with the Government. By accepting delivery of this work, the Government agrees that this work and the Licensed Program(s) described herein qualify as "commercial" computer software within the meaning of the acquisition regulation(s) applicable to this procurement. The terms of conditions of the license for the Licensed Program(s) shall pertain to the Government's use and disclosure of this work and the Licensed Program(s), and shall supersede any conflicting contractual terms or conditions. If the license for this work and the Licensed Program(s) fails to meet the Government's need or is inconsistent in any respect with Federal law, the Government agrees to return this work and the Licensed Program(s), unused, to FileTek, Inc. The following additional statement applies only to acquisitions governed by DFARS Subpart 227.4 (October 1988) "Restricted Rights – Use, duplication and disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 (OCT. 1988)." Unpublished licensed work property of FileTek, Inc. Unauthorized use, duplication or distribution prohibited. All rights reserved. A copyright notice on this work and/or on the Licensed Program(s) by itself does not constitute publication or public disclosure of this work or the Licensed Program(s). The contractor/manufacturer is:

FileTek, Inc.
9400 Key West Avenue
Rockville, Maryland 20850

Information in this document is subject to change without notice and does not represent a commitment on the part of FileTek, Inc. Further, FileTek, Inc. reserves the right to supplement the document with information not available at the time of creation of the document. FILETEK, INC. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, AND CANNOT WARRANT THE RESULTS YOU MAY OBTAIN USING THE DOCUMENT. IN NO EVENT SHALL FILETEK, INC. BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OR DATA, INTERRUPTION OF BUSINESS, OR FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, EVEN IF FILETEK, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES ARISING FROM ANY DEFECT OR ERROR IN THIS PUBLICATION. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

FileTek and StorHouse are registered U.S. trademarks of FileTek, Inc. VRAM is a U.S. trademark of FileTek, Inc. All other brand or product names are trademarks or registered trademarks of their respective owners.

Documentation for FileTek's StorHouse product. Protected by the following U.S. Patents: 4,864,572; 5,247,660; 5,727,197; 6,049,804.

Contents

Welcome xiii

Purpose of This Document xiii

Intended Audience xiii

Contents xiv

Related Documentation xiv

Notational Conventions xv

Chapter 1: Introduction 1-1

Operating Environment 1-1

Callable Interface Function Hierarchy 1-2

- Session Control Functions 1-2
- File Operation Functions 1-3
- Data Transfer Control Functions 1-3
- StorHouse Command Functions 1-4

Notes on Multitasking 1-4

Chapter 2: StorHouse Parameters and Data Descriptions 2-1

Session and Data Transfer Link Identifiers 2-1

StorHouse Accounts 2-1

- Account Identification Code 2-2
- Account Password 2-2
- Default Access Groups and Rights 2-2
- StorHouse Privileges 2-2

StorHouse Files and File Access Groups 2-3
StorHouse File Names 2-3
File Access Groups 2-4
Group and File Access Passwords 2-4
Group Passwords 2-4
File Passwords 2-5
File Versions 2-5
File Revisions 2-6
File Data Representation 2-6
Directory Information 2-7

Chapter 3: File Positioning 3-1

Record Sequencing 3-1
Entry Sequence 3-1
Key Sequence 3-1
Current Record Position 3-2
Read Functions and Current Record Position 3-2
Record Sequencing Example 3-2

Chapter 4: Control Structures 4-1

Parameter Values 4-1
Character Strings 4-1
Externally Specified Parameters 4-1
Return Codes 4-2
Indicative Text Messages 4-2

Chapter 5: Callable Interface Functions 5-1

Callable Interface Entry Point Names 5-1
Special Considerations for CICS Programmers 5-1
Defining the CICS Interface Programs 5-2
Error Handling 5-3
Restrictions 5-3
Synchronous and Asynchronous Functions 5-4
Synchronous Form 5-4
Asynchronous Form 5-4
Function Statement Format 5-5

Session Control Functions 5-5

CONNECT 5-6
- Statement Format for COBOL 5-6
- Working Storage Section for COBOL Program 5-6
- Parameter Overview 5-6
- Return Codes 5-7
- Detailed Function Description 5-7
- Notes 5-8
- Cross-Reference to Sample Program 5-8

DISCONNECT 5-9
- Statement Format for COBOL 5-9
- Working Storage Section for COBOL Program 5-9
- Parameter Overview 5-9
- Return Codes 5-9
- Detailed Function Description 5-10
- Notes 5-10
- Cross-Reference to Sample Program 5-10

File Operation Functions 5-11

OPEN 5-12

OPEN-SEQ 5-13
- Statement Format for COBOL 5-13
- Working Storage Section for COBOL Program 5-13
- Parameter Overview 5-14
- Return Codes 5-18
- Detailed Function Description 5-18
- Notes 5-18
- Cross-Reference to Sample Program 5-19

CREATE-OPEN 5-20
- Statement Format for COBOL 5-20
- Working Storage Section for COBOL Program 5-20
- Parameter Overview 5-21
- Return Codes 5-24
- Detailed Function Description 5-25
- Notes 5-25
- Cross-Reference to Sample Program 5-26

OPEN-VRAM 5-27
- Statement Format for COBOL 5-27
- Working Storage Section for COBOL Program 5-27
- Parameter Overview 5-28
- Return Codes 5-30
- Detailed Function Description 5-31
- Notes 5-31

Cross-Reference to Sample Program 5-32

CHECKPOINT 5-33
Statement Format for COBOL 5-33
Working Storage Section for COBOL Program 5-33
Parameter Overview 5-33
Return Codes 5-34
Detailed Function Description 5-34
Notes 5-34
Cross-Reference to Sample Program 5-34

CLOSE 5-35
Statement Format for COBOL 5-35
Working Storage Section for COBOL Program 5-35
Parameter Overview 5-35
Return Codes 5-36
Detailed Function Description 5-36
Notes 5-37
Cross-Reference to Sample Program 5-37

Data Transfer Control Functions 5-38

READ 5-39
Statement Format for COBOL 5-39
Working Storage Section for COBOL Program 5-39
Parameter Overview 5-39
Return Codes 5-40
Detailed Function Description 5-40
Notes 5-40
Cross-Reference to Sample Program 5-40

READ-SEQ 5-41
Statement Format for COBOL 5-41
Working Storage Section for COBOL Program 5-41
Parameter Overview 5-41
Return Codes 5-42
Detailed Function Description 5-42
Notes 5-42
Cross-Reference to Sample Program 5-42

READ-RECORD 5-43
Statement Format for COBOL 5-43
Working Storage Section for COBOL Program 5-43
Parameter Overview 5-43
Return Codes 5-44
Detailed Function Description 5-44
Notes 5-44
Cross-Reference to Sample Program 5-44

READ-KEYED 5-45
Statement Format for COBOL 5-45
Working Storage Section for COBOL Program 5-45
Parameter Overview 5-45
Return Codes 5-46
Detailed Function Description 5-46
Notes 5-47
Cross-Reference to Sample Program 5-47

READ-NEXT-KEY 5-48
Statement Format for COBOL 5-48
Working Storage Section for COBOL Program 5-48
Parameter Overview 5-48
Return Codes 5-49
Detailed Function Description 5-49
Notes 5-49
Cross-Reference to Sample Program 5-49

WRITE 5-50
Statement Format for COBOL 5-50
Working Storage Section for COBOL Program 5-50
Parameter Overview 5-50
Return Codes 5-51
Detailed Function Description 5-51
Notes 5-51
Cross-Reference to Sample Program 5-51

WRITE-KEY 5-52
Statement Format for COBOL 5-52
Working Storage Section for COBOL Program 5-52
Parameter Overview 5-52
Return Codes 5-53
Detailed Function Description 5-53
Notes 5-53
Cross-Reference to Sample Program 5-54

DELETE 5-55
Statement Format for COBOL 5-55
Working Storage Section for COBOL Program 5-55
Parameter Overview 5-55
Return Codes 5-55
Detailed Function Description 5-56
Note 5-56
Cross-Reference to Sample Program 5-56

CHANGE 5-57
Statement Format for COBOL 5-57
Working Storage Section for COBOL Program 5-57

Parameter Overview 5-57
Return Codes 5-58
Detailed Function Description 5-58
Note 5-58
Cross-Reference to Sample Program 5-58

StorHouse Command Submission 5-59

SM-CMD-INTF 5-60
Statement Format for COBOL 5-60
Working Storage Section for COBOL Program 5-60
Parameter Overview 5-61
Return Codes 5-62
Detailed Function Description 5-62
Note 5-63
Cross-Reference to Sample Program 5-63

General Usage Functions 5-64

CHECK 5-65
Statement Format for COBOL 5-65
Working Storage Section for COBOL Program 5-65
Parameter Overview 5-65
Return Codes 5-65
Detailed Function Description 5-66
Notes 5-66
Cross-Reference to Sample Program 5-66

ECBADDR 5-67
Statement Format for COBOL 5-67
Working Storage Section for COBOL Program 5-67
Parameter Overview 5-67
Return Codes 5-68
Detailed Function Description 5-68
Note 5-68
Cross-Reference to Sample Program 5-68

EMSG 5-69
Statement Format for COBOL 5-69
Working Storage Section for COBOL Program 5-69
Parameter Overview 5-69
Return Codes 5-70
Detailed Function Description 5-70
Notes 5-70
Cross-Reference to Sample Program 5-70

ABORT 5-71
Statement Format for COBOL 5-71
Working Storage Section for COBOL Program 5-71

Parameter Overview 5-71
Return Codes 5-71
Detailed Function Description 5-72
Notes 5-72
Cross-Reference to Sample Program 5-72

Chapter 6: Sample Program 6-1

COBOL Sample Program 6-2

Appendix A: Pass-Through Functions A-1

PTOPEN A-2
Statement Format for C A-2
Parameter Overview A-3
Return Codes A-4
Detailed Function Description A-4
Notes A-4

PTWRTOSM A-5
Statement Format for C A-5
Parameter Overview A-5
Return Codes A-6
Detailed Function Description A-6
Notes A-6

PTRDFRSM A-7
Statement Format for C A-7
Parameter Overview A-7
Return Codes A-8
Detailed Function Description A-8
Notes A-8

CONFIG A-9
Statement Format for C A-9
Parameter Overview A-9
Return Codes A-10
Detailed Function Description A-10
Note A-10

Appedix B: ALC Macro Definition B-1

LSMCALL – Call the Callable Interface Program B-1
Required Parameters B-3
Optional Parameters B-3

Remaining Keywords B-4

Assembly Language Standard Call B-6
Example: CALL Macro B-7
Example: LSMCALL Macro B-8

Appendix C: Checkpoint/Restart and Programming Guidelines C-1

Checkpoint/Restart C-1
Examples C-1
Example 1 ..... C-2
Example 2 ..... C-2
Example 3 ..... C-3
Example 4 ..... C-3

Programming Guidelines C-4
Defining Resources C-4
Examples C-5
User Guidelines C-6
Permanent Fixes C-6

Index Index-1

Figures

Figure 1-1: Callable Interface Function Hierarchy 1-2

Tables

Table 2-1: Printable ASCII Characters 2-3
Table 2-2: File System Types 2-7

Table B-1: LSMCALL Macro Instruction B-2
Table B-2: CALL Macro B-7
Table B-3: LSMCALL Macro B-8

Table C-1: DATAFILE Revisions C-2
Table C-2: Example of Open Statements Requiring the Same Resource C-5

Tables

Table 3-3: Printable ASCII Characters 2-3
Table 3-4: File System Types 2-7
Table 3-5: LSMCALL Macro Instruction B-2
Table 3-6: CALL Macro B-7
Table 3-7: LSMCALL Macro B-8
Table 3-8: DATAFILE Revisions C-2
Table 3-9: Example of Open Statements Requiring the Same Resource C-5

Welcome

The *Callable Interface Programmer's Guide* describes the StorHouse® Callable Interface for IBM™ MVS™ hosts. This interface provides access to StorHouse from end-user applications. For information on the StorHouse Callable Interface for all other hosts except IBM MVS, please refer to the *Generic Callable Interface Programmer's Guide*, publication number 900046.

Purpose of This Document

This document is a reference manual that describes the StorHouse Callable Interface functions. The standard COBOL format, parameter overview, return codes, detailed function description, and cross-reference to the sample program in Chapter 6, "Sample Program," are presented for each function.

Intended Audience

The *Callable Interface Programmer's Guide* was written for programmers who write applications that invoke the StorHouse Callable Interface to access information that resides on StorHouse.

Contents

The *Callable Interface Programmer's Guide* contains six chapters and three appendices.

- Chapter 1, "Introduction," describes the Callable Interface operating environment.
- Chapter 2, "StorHouse Parameters and Data Descriptions," contains general information about StorHouse session and data transfer link identifiers, accounts, file access groups, and files.
- Chapter 3, "File Positioning," discusses record resequencing and explains how the various read functions affect record positioning for files.
- Chapter 4, "Control Structures," discusses parameter values, return codes, and text messages.
- Chapter 5, "Callable Interface Functions," defines each Callable Interface function in detail. This chapter groups the functions by purpose.
- Chapter 6, "Sample Program," presents a sample COBOL program for the Callable Interface.
- Appendix A, "Pass-Through Functions," describes pass-through functions that allow application programs direct access to StorHouse.
- Appendix B, "ALC Macro Definition," explains how to access the Callable Interface from programs coded in IBM Assembler language.
- Appendix C, "Checkpoint/Restart and Programming Guidelines," discusses checkpoint/restart operations and offers programming guidelines.

Related Documentation

Users of the Callable Interface should be familiar with these StorHouse documents:

- The *Messages and Codes Manual*, publication number 900032, describes the messages and return codes generated by the StorHouse system and host software.
- The *Command Language Reference Manual*, publication number 900005, explains StorHouse Command Language in detail.
- The *StorHouse Concepts and Facilities Manual*, publication number 900026, explains StorHouse concepts, structures and functions.

- The *StorHouse Glossary*, publication number 900027, defines terminology used in FileTek® publications. You can use the glossary as a stand-alone reference manual or as a companion to the StorHouse User Document Set.
- The *Host Software Installation and Operations Guide*, publication number 900011, explains how to install the host software for StorHouse.

Notational Conventions

This book uses the following conventions for illustrating command formats, presenting examples, and identifying special terms:

Convention	Meaning
Angle brackets (< >)	Enclose optional entries
Braces ({ })	Enclose descriptive terms or a choice of entries
Courier font	Code
Ellipses (...)	A repetition of the preceding material
Italics	New terms and emphasized text
lower case Helvetica font	User entries
UPPER CASE	System responses and StorHouse terms

Chapter

1

Introduction

The StorHouse Callable Interface provides access to StorHouse from user applications. This interface is implemented as a subroutine invoked from assembly language (ALC), PL/1, COBOL, FORTRAN, or C programs. By supplying parameters to this subroutine, a programmer can establish connections, access files, and transfer records between a host computer system and StorHouse.

The Callable Interface can be used by an application run from a TSO session, a batch job, an IMS transaction, or a CICS transaction. The user application does not require authorization, and all of the Callable Interface code that executes in the user's address space is re-entrant.

Operating Environment

The Callable Interface operates in either an MVS/SP™ (Release 1.3 or later) or MVS/XA™ environment. This interface tolerates the MVS/ESA™ environment but does not exploit ESA capabilities. User applications can be compiled with any of the following:

- Any MVS Assembler
- PL/1 Optimizing Compiler Release 5.1
- COBOL VS II Compiler 1.1
- FORTRAN VS Compiler 4.1
- Any other compiler that generates standard call-by-reference parameter lists.

Callable Interface modules for a TSO/batch environment use standard MVS system functions, such as GETMAIN, FREEMAIN, LOAD, and WAIT. The system programmer responsible for the installation can change the use of these functions through installation exits. Unless the exits are changed, the Callable Interface should only be used in environments where use of these system functions is acceptable.

For the CICS Interface, Callable Interface modules use standard command level functions. For more information about the CICS Interface, see Chapter 5, “Callable Interface Functions.”

Callable Interface Function Hierarchy

Figure 1-1 illustrates how Callable Interface functions are organized in a hierarchical structure.

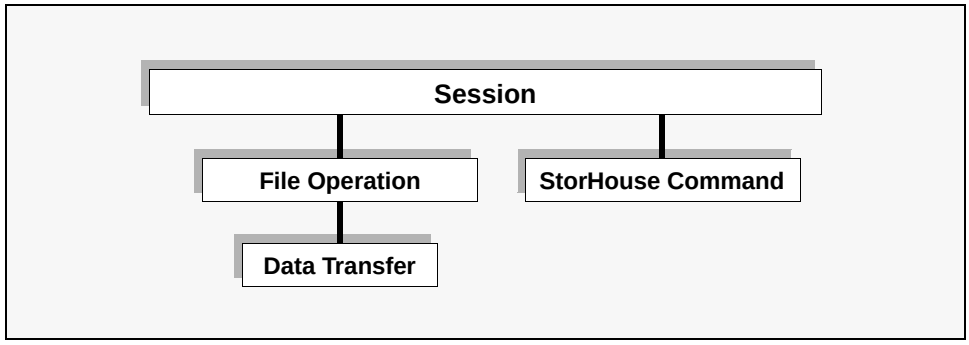

Figure 1-1: Callable Interface Function Hierarchy

To perform any StorHouse operation, an application must:

1. Establish a session with StorHouse, using the session function CONNECT.
2. Within a session, initiate file operations, using one of the open functions.
3. Once a file is opened, issue data transfer operations, using one of the read/write functions, or issue record update operations using the CHANGE or DELETE function.

In addition, within an established session, a program can issue StorHouse commands such as SHOW FILE or SHOW ACCOUNT.

Session control, file operation, data transfer, and StorHouse command functions are described in the following sections. Chapter 5, “Callable Interface Functions,” explains all parameters for each Callable Interface function.

Session Control Functions

There are two Callable Interface session control functions: CONNECT and DISCONNECT.

- CONNECT establishes a session between a user application and StorHouse. The application must supply an account identifier and password for CONNECT so that the StorHouse security system can validate the session and set session privileges and defaults.

- DISCONNECT ends the established session and releases all session-related host and StorHouse resources.

One application can establish several sessions. StorHouse assigns each session a unique session identifier to allow explicit application control of the operations that are performed in that session. CONNECT returns the session identifier, or C-TOKEN, to the application as a 32-bit integer value.

File Operation Functions

There are five file operation functions: OPEN-SEQ, CREATE-OPEN, OPEN-VRAM, CHECKPOINT, and CLOSE.

- OPEN-SEQ allows file-oriented operations that read or create a complete StorHouse file.

- CREATE-OPEN creates a new VRAM™ file on StorHouse, and then establishes a data transfer link for writing data to that file.

- OPEN-VRAM allows read, write, delete, and update access to individual records in StorHouse VRAM files.

- CHECKPOINT synchronizes file transfer by ensuring that all previously written records have been received and processed by StorHouse.

- CLOSE terminates the file operation (indicating end-of-file for write operations) and releases all resources used by the transfer operation.

OPEN-SEQ, CREATE-OPEN, and OPEN-VRAM establish a data transfer link between the user application and StorHouse. These functions return the O-TOKEN, a 32-bit integer value that identifies the data transfer path within a given session. The O-TOKEN also identifies the file being processed in all subsequent transfer-oriented functions (for example, read and write).

Data Transfer Control Functions

Data transfer operations can be performed once a session has been established and files have been opened. Data transfer control functions (read and write) allow an application to send records to and receive records from StorHouse. Individual records in a VRAM file can be retrieved by record number or by key, and changed or deleted accordingly.

A record is an arbitrary unit of data. The user completely controls record size. The user also controls record content, unless the user specifies a translation to ASCII. Then data should contain only EBCDIC characters that have ASCII equivalents.

StorHouse Command Functions

Within a session, StorHouse command functions allow an application to send selected Command Language commands to StorHouse and to retrieve response text from those commands. These functions also allow administrative operations to be directed from an application rather than from a user at a terminal through the Interactive Interface.

Notes on Multitasking

A session established by one task can be used from a subtask (in other words, under a different Task Control Block [TCB]). However, only one function can be performed at a time. Serialization of calls is the application programmer's responsibility. DISCONNECT must be called from the same task that issued CONNECT, and CLOSE must be called from the same task that issued OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

If a file is opened and closed under one TCB and read or written from another TCB, the two tasks must share Subpool 0 storage. If one of these tasks is a subtask of the other, this is accomplished by the SZERO=YES operand on the ATTACH MACRO (this is the default value).

Chapter

2

StorHouse Parameters and Data Descriptions

This chapter contains general information about StorHouse session and data transfer link identifiers, accounts, file access groups, and files. For more information about these topics, refer to the *StorHouse Concepts and Facilities Manual* and the *Command Language Reference Manual*.

Session and Data Transfer Link Identifiers

There are two types of link identifiers: session and data transfer.

- *Session link identifier* – CONNECT returns the C-TOKEN, or session link identifier.
- *Data transfer link identifier* – OPEN-SEQ, CREATE-OPEN, and OPEN-VRAM return the O-TOKEN, or data transfer link identifier.

The C-TOKEN and O-TOKEN are 32-bit integer values that must be passed to all other Callable Interface functions that perform operations with the session or data link. Token values can be moved from one variable to another, but they must not be subjected to any arithmetic operations, including type conversions.

StorHouse Accounts

An *account* is a collection of administrative data that StorHouse uses to control a session. Each account includes an identification code (AID) and a password. Generally, each account has a default access group, access rights to that group, and a set of privileges. The following sections describe StorHouse accounts.

Account Identification Code

An *account identification code*, or AID, is similar to a TSO user ID in MVS; it provides StorHouse with the user's identity. An AID must contain 1 to 12 characters and include only the following characters: A-Z, 0-9, $, and _ (underscore). An example of a valid AID is USER.

A program must include an account identifier to establish a session in the Callable Interface. Multiple programs can use the same or different accounts and can access StorHouse at the same time.

Account Password

The *account password* helps maintain system security. Passwords must contain 0 (null) to 32 characters and include only the following characters: A-Z, 0-9, $, or _ (underscore). Generally, long passwords provide better system protection than short passwords. Passwords of three or fewer characters offer only marginal protection.

Default Access Groups and Rights

Usually, each account has a *default access group* and *default access rights* to that group. When a command accesses a StorHouse file, StorHouse assumes that the file is in the default group unless a different group name is specified in the command.

An account may be set up to give read, write, delete, or no default access to the default group. These are the access rights to the default group. Thus, to perform operations on files in the default group, it is not necessary for a program to supply a group access password unless the operation requires an access that differs from the account's default access.

In any case, if the default group has a null password, a program automatically receives the corresponding type of access without having default access or specifying a password. A program can switch to a different default access group during a session.

For more information about how to specify account information, see Chapter 6, "Sample Program."

StorHouse Privileges

Each account has a set of privileges that falls into two categories: access and command.

- *Access privileges* allow the program using the account to bypass various system security checks.

- *Command privileges* permit the program using the account to perform specific commands or groups of commands.

The privileges assigned to an account determine the functions that can be performed by an application. For a complete list of access and command privileges, refer to the *Command Language Reference Manual* in the StorHouse User Document Set.

StorHouse Files and File Access Groups

A *file* is a named collection of logically related data located on a medium and treated as a unit by StorHouse. Any collection of data generated by a host program can be stored in StorHouse. Each StorHouse file has a set of attributes that govern where and how it is stored. Each file can be protected by passwords. A file version can also be locked to prevent programs using other accounts from reading or writing it and unlocked to make it available to other programs using other accounts.

StorHouse File Names

A *file name* is a character string that contains 1 to 56 bytes. StorHouse uses file names to identify files. The name must be left-justified within the field and padded with blanks. Uppercase characters are distinct from lowercase characters. At least one character must be non-blank.

StorHouse file names must consist of printable ASCII characters and/or the ASCII space character as shown in Table 2-1.

Table 2-1: Printable ASCII Characters

Printable ASCII Characters					
A-Z	uppercase letters	+	plus sign	()	parentheses
a-z	lowercase letters	~	tilde	< >	angle brackets
0-9	digits	,	comma	[ ]	square brackets
!	exclamation point	-	hyphen	{ }	braces
“	quote	.	period	\	backslash
#	number sign	/	slash	^	circumflex
$	dollar sign	:	colon	_	underscore
%	percent sign	;	semicolon	|	vertical bar
&	ampersand	=	equal sign	‘	reverse apostrophe

Table 2-1: Printable ASCII Characters (continued)

Printable ASCII Characters					
’	apostrophe	?	question mark		space
*	asterisk	@	at sign		

File Access Groups

A *file access group* is a set of named files. Program access may be restricted to a file access group. To manipulate files in the set, a program must specify a group name. Group names must contain 1 to 8 characters and include only the following characters: A-Z, 0-9, $, or _ (underscore). If the group is protected by passwords, a program must also specify the group’s read, write, and/or delete password.

Each file in StorHouse is a member of one group. Within that group, each file name is unique. However, two files may have the same name if they are located in different file access groups.

Group and File Access Passwords

StorHouse allows the specification of group and file passwords to protect user files from unauthorized access. The following sections explain how to use group and file passwords.

Group Passwords

Group passwords may be null or contain 1 to 8 characters, and may include only the following characters: A-Z, 0-9, $, and _ (underscore).

Group passwords are used as follows:

- If a group has a read password, a program must specify the correct read password to read a file from StorHouse or display group or file directory information.
- If a group has a write password, a program must specify the correct write password to write a file into the group or to UNDELETE a file.
- If a group has a delete password, a program must specify the correct delete password to delete the group, delete a file from the group, change the group’s passwords, or change passwords or attributes of files in the group.
- If a group has a null password, a program may gain the corresponding type of access by specifying a null password or by not specifying a password.

- If a program specifies a null password where the group has a non-null password, StorHouse does not grant that type of access (in other words, read, write, or delete access). However, StorHouse does not return an error unless that type of access is required.

File Passwords

A program can also give individual files read, write, and delete passwords. These *file passwords* control access to files in the same way that group passwords control access to file access groups.

File passwords can be null or contain 1 to 8 characters. Like group passwords, file passwords may contain only the following characters: A-Z, 0-9, $, or _ (underscore).

File passwords are used as follows:

- If a file has a read password, the program must specify the correct read password to read the file from StorHouse or display directory information about the file.
- If a file has a write password, the program must supply the write password to write a new version of the file to StorHouse or to UNDELETE a file.
- If a file has a delete password, the program must specify the correct delete password to delete the file from StorHouse, or change the file's attributes or passwords.
- If a file has a null password, the program may gain the corresponding type of access by specifying a null password or by not specifying a password.
- If a program specifies a null password where the file has a non-null password, StorHouse does not grant that type of access (in other words, read, write, or delete access). However, StorHouse does not return an error code unless that type of access is required.

The account used by the program must have read, write, or delete access to a file's group before the system allows the program to gain the corresponding access to the file.

For more information about how to specify file access group names and group and file access passwords, consult the "Data Transfer Control Functions" section of Chapter 5 and Chapter 6, "Sample Program."

File Versions

A new *file version* is created whenever a program transfers a non-VRAM file from the host to StorHouse. A new version of a VRAM file is created whenever a CREATE FILE command is performed or a CREATE-OPEN call is issued with a checkpoint of 0. The

new file receives version number 0. If a file of the same name and the same access group already exists in StorHouse, the number of each previous version decreases by one. Previous versions may range in number from -1 through -32767 or from -1 through the minimum version number allowed by the LIMIT attribute. If there was a previous version -32767, it is deleted when a new version is added to StorHouse.

For example, when the file DATAFILE is first added to StorHouse, it becomes version 0. When a new version of DATAFILE is added, that becomes version 0, and the previous version 0 of DATAFILE becomes version -1.

Refer to the OPEN-SEQ and OPEN-VRAM function descriptions in Chapter 5, "Callable Interface Functions" for information about specifying version numbers.

File Revisions

For RECORD, KEYED, or KEYSEQUENTIAL VRAM files, StorHouse assigns revision number 1 to a file version when the file is created on StorHouse. Each time a user changes the contents of the file version, StorHouse increments the revision number by 1. A user can change the contents of a file version by opening the file; changing, deleting, or adding records; and closing the file. Thus, a file version can have multiple revisions, each identified by a unique revision number.

Revision numbers can be expressed as relative or absolute numbers. Relative revision numbers range from 0, the current revision, through -65534, the oldest revision. Absolute revision numbers range from 1 through 65535. For example, assume that relative version 0 of the file DATAFILE has four revisions. A user can refer to the most current revision of this file as relative revision number 0 or as absolute revision number 4.

Refer to the OPEN-VRAM function description in Chapter 5, "Callable Interface Functions" for information about specifying values for revision numbers.

File Data Representation

StorHouse stores the record stream written by the user application program either as *binary* (bitstream) data or as an *ASCII character stream*. The file format is determined when the file is created, either by OPEN-SEQ for sequential files, or by the CREATE FILE command or the CREATE-OPEN function for VRAM files. The record data is treated as a bitstream unless the DATA-XLATE-FLAG in the file attributes array for OPEN-SEQ is positive, or the ASCII modifier is specified on the CREATE FILE command. For information about CREATE FILE, refer to the *Command Language Reference Manual*.

Files created through the Callable Interface are considered transportable by StorHouse. They can be accessed by host computers from different manufacturers running different operating systems. For binary records, the user application program is responsible for any required data conversion. For ASCII files, the host translates from ASCII to the host character mode (EBCDIC for IBM).

Directory Information

The StorHouse directory entry for a file indicates whether the file's record format is binary bitstream or ASCII character stream by the value of the file system type. File system type is set to 65 for ASCII files and to 66 for binary files. Files created by FileTek host dataset utility programs are given other file system type identifiers, based on the specific utility used to copy the dataset. These other file system types are shown in Table 2-2.

Table 2-2: File System Types

File Type	File System Type	Host Type	Description
01			Standard, StorHouse Framed File
	03	17	MVS files copied using DF/DSS
	65	n/a	Transportable ASCII Character Stream
	66	n/a	Transportable BINARY Stream

Chapter

3

File Positioning

This chapter discusses record resequencing and explains how the various read functions affect record positioning for files.

Record Sequencing

Records in a keyed VRAM file are sequenced by both entry and key. Entry sequenced records are sequenced by the order they were written to the file. Key sequenced records are sequenced by the values of key fields in each record.

Entry Sequence

The write operation that builds a file determines the entry sequence for records in that file. Each new record is appended to the end of the file, independent of record content. *Entry sequence* determines the order in which sequential read operations retrieve records. Entry sequence has no effect on the order in which key value read operations and next-key sequence read operations retrieve records.

Key Sequence

Each key that is defined for a file determines a key sequence for records in that file. In key sequence, records are ordered by the increasing value of their key field, which is considered only as a binary bitstring. *Key sequence* determines the order in which next-key read operations retrieve records. The order in which records with duplicate keys are returned is not necessarily the same as their entry sequence.

Current Record Position

Any opened StorHouse file has a current record position. A keyed VRAM file has two current record positions:

- *Sequential position*, which follows record entry sequence.
- *Key position*, which follows key sequence.

A file can have several keys but only one key position. Key position is always relative to the last key that was used to read a record.

For a keyed VRAM file, open sets the sequential record position to the beginning of the file, which is the first record in entry sequence. Open does not initialize key position.

Read Functions and Current Record Position

Four functions can be used to read a VRAM file:

- READ-SEQ – retrieves the next record in entry sequence order.
- READ-KEYED – retrieves a record by exact match of a specified value for a given key.
- READ-NEXT-KEY – retrieves the next record in key sequence order.
- READ-RECORD – retrieves a specific record by record number.

Read functions maintain current record position for a file as follows:

- Every read operation updates sequential record position.
- Only READ-KEYED and READ-NEXT-KEY update key position. READ-KEYED sets the current key position and must be called at least once prior to calling READ-NEXT-KEY.

Record Sequencing Example

The following example shows a VRAM file with two keys, NAME and ENUM. The file was created by writing the following records, where record number matches the entry sequence.

Record Number	Value of Key NAME	Value of Key ENUM
1	Jones	327
2	Smith	409
3	Doe	427
4	Johnson	283
5	Smith	265
6	Brooks	301

If the file is opened and read sequentially, then the records are read in the following order: 1, 2, 3, 4, 5, 6.

The following table indicates how each read function affects the file's current record position.

Read Operation Function		Record Number	NAME key	ENUM key
1.	OPEN-VRAM	None		
2.	READ-SEQUENTIAL	1	Jones	327
3.	READ-RECORD 4	4	Johnson	283
4.	READ-SEQUENTIAL	5	Smith	265
5.	READ-KEYED (KEY=NAME (VALUE=Doe)	3	Doe	427
6.	READ-NEXT-KEY	4	Johnson	283
7.	READ-SEQUENTIAL	5	Smith	265
8.	READ-KEYED (KEY=ENUM) (VALUE=283)	4	Johnson	283
9.	READ-NEXT-KEY	6	Brooks	301
10.	READ-KEYED (KEY=NAME) (VALUE=Smith)	2	Smith	409
11.	READ-NEXT-KEY	5	Smith	265
12.	READ-SEQUENTIAL	6	Brooks	301

Note that in the preceding table, operations 10 and 11 may return record number 5, then record number 2. The order of duplicate key records may be changed by the file's update and delete history.

Chapter

4

Control Structures

This chapter discusses parameter values and return codes.

Parameter Values

Parameter values can be specified as character strings or externally through JCL statements.

Character Strings

All character strings must be left-justified and blank-filled. The character set is A-Z, a-z, 0-9, and the following special characters:

, : “ ' ! - (). * /

Some character string fields have additional restrictions on the character set and are documented as required in Chapter 5, “Callable Interface Functions.”

Externally Specified Parameters

The values for some parameters can be externally specified through JCL statements. These parameters are documented specifically for each function. All such parameters are strings and link to a JCL statement when the string value specified in the program has the format (illustrated for COBOL):

```
05 name    PIC X(12) VALUE 'DD=ddname   '.
```

The value must begin exactly with the characters “`DD=`” in uppercase. The JCL statement referenced is named by “`ddname`”. The actual value assigned to the parameter is taken from a field on the associated JCL statement. The specific field

differs for various parameters and is documented in the specific function parameter description.

Return Codes

Return codes are binary values returned in a word (32 bits). The parameter is named R-CODE in all function call descriptions. All returned values are positive.

The return code 0 always indicates normal completion. All return code values are documented in the *Messages and Codes Manual* in the StorHouse User Document Set. Common return codes for each function are documented in the function description section of this manual.

Indicative Text Messages

A Callable Interface function may generate text messages that provide commentary, warnings, and error diagnostics associated with the processing of the function. These messages are text strings that can be printed or displayed at a terminal.

These messages are not returned directly by the function. They are placed in a message stack and can be retrieved only by calling the EMSG function. These messages may be ignored. The stack is cleared when the next function request is made.

The indicative text message stack is normally cleared when the session or the data transfer operation is ended. However, clearing the text message stack in this manner also deletes any messages that were generated during the session disconnect or during the transfer close operation. It is the user's responsibility to indicate whether these messages will be retrieved, either when the session is established or when the data transfer is opened. The definition sections for CONNECT, OPEN-SEQ, CREATE-OPEN, and OPEN-VRAM document the use of a flag to control this message retention capability.

A non-zero return code does not guarantee that an indicative message is available. Conversely, a zero return code does not guarantee that there are no messages in the stack.

Chapter

5

Callable Interface Functions

Chapter 5 contains a detailed definition of all Callable Interface functions. The functions are grouped into the following categories:

- Session control
- File operation
- Data transfer
- StorHouse command interface
- General usage.

Chapter 5 also explains Callable Interface entry point names, special considerations for CICS programmers, and how to issue functions in synchronous and asynchronous form.

Callable Interface Entry Point Names

Callable Interface functions are invoked through two entry points: LSMCALL and LSMCICS.

- All TSO/batch programs use the entry point LSMCALL. For FORTRAN programs, LSMCAL is an alias for LSMCALL.

- All CICS programs must use the entry point LSMCICS.

Special Considerations for CICS Programmers

All material presented in this document applies to both the Callable Interface and the CICS Interface with the exception of the program names as described in the previous paragraph.

All function parameter lists are identical for LSMCALL and LSMCICS with the following exception:

Note For every CICS Interface function, the first two parameters in the CICS Interface parameter list must be DFHEIBLK and DFHCOMMAREA. DFHEIBLK and DFHCOMMAREA are then followed by the standard parameter list that is documented for each function listed in this manual.

Defining the CICS Interface Programs

The CICS Interface uses CICS Distributed Transaction Processing facilities. It consists of one transaction, LSMC, and five programs: LSMCFLH, LSMCICS, LSMLSMC, LSMLOAD, and LSMUXSSN. LSMLOAD and LSMUXSSN are components of the standard (non-CICS) StorHouse Host Interface. LSMCICS is the interface stub link-edited with a user transaction program. It provides functions in a CICS environment equivalent to those of LSMCALL, the interface stub used in a Batch or TSO environment.

When a user program CALLs the LSMCICS stub, LSMCICS executes a CICS LINK to program LSMCFLH. When first establishing a StorHouse session (CONNECT function), LSMCFLH starts server transaction LSMC, which then invokes program LSMLSMC. The LSMC transaction remains active for the life of a user's StorHouse session (CONNECT to DISCONNECT) and processes user requests from LSMCFLH to the Host Subsystem and StorHouse. During a StorHouse session, a user application may consist of one CICS conversational transaction, or it may span multiple CICS pseudo-conversational tasks across multiple CICS regions and systems. Application designers and programmers must work with CICS system programmers to develop and define the required CICS resource definitions. (Refer to Chapter 7 of the *Host Software Installation and Operations Guide* in the StorHouse User Document Set.)

Please note the following important requirement for O-TOKENs and C-TOKENs, the session identifiers that are returned to a user program after OPEN and CONNECT function requests, respectively:

Note CICS pseudo-conversational transaction programs that use the CICS interface must be written so that O-TOKENs and C-TOKENs are saved and passed to subsequent transactions and programs that use the same StorHouse session.

This can be accomplished by defining the tokens in a COMMAREA specified on various CICS control transfer commands.

For example, it is desired that PROGRAM 1 issue a CONNECT request and then return to CICS specifying that transaction NEXT gets control. Transaction NEXT will then do more StorHouse processing. To achieve this, PROGRAM 1 defines the following in its WORKING-STORAGE:

```
01 C-TOKEN           PIC S9(8) COMP VALUE +0.

01 COMMAREA-FOR-TRANSACTION-NEXT.
   05PASS-C-TOKEN    PIC S9(8) COMP.

.
. (other storage definitions)
.
```

PROGRAM 1 states the following in its PROCEDURE DIVISION:

```
CALL 'LSMCICS'    USING DFHEIBLK,DFHCOMMAREA,
                  CONNECT,C-TOKEN,R-CODE,
                  ...other parameters...
```

When it is time to pass control to transaction NEXT, PROGRAM 1 states:

```
MOVE C-TOKEN TO PASS-C-TOKEN.
EXEC CICS RETURN TRANSID ('NEXT')
         COMMAREA (COMMAREA-FOR-NEXT-TRANSACTION)
            LENGTH (LENGTH OF COMMAREA-FOR-NEXT-TRANSACTION)
```

Error Handling

When interface programs LSMCFLH and LSMLSMC encounter errors associated with their processing, rather than StorHouse-related errors, both write error messages to CSMT, the CICS transient data queue in the CICS region where they are executing. They also write error messages to CEBRtermid, a temporary storage queue in the CICS region that owns the terminal associated with the transaction. To view these messages, the terminal operator can invoke the CEBR transaction. By default, CEBR browses CEBRtermid. This is the same convention used by COBOL II. Messages written to the CSMT DESTID appear in the CICS region SYSOUT output. To assist in finding Interface messages within this large output dataset, all messages begin with an exclamation mark (!). When using a product like SDSF, a FIND ! command locates these messages.

Restrictions

The CICS environment imposes the following restrictions on an applications programmer:

- All application programs must be written in Command Level CICS using Assembler, COBOL, COBOL II, PL/1, or C.
- Asynchronous forms of Callable Interface functions are not allowed.

- As with any database-type system (such as DB2), CICS resources should not be held across calls to LSMCICS.
- Non-terminal related tasks can CALL LSMCICS to access StorHouse. However, any such transaction must complete the StorHouse session. That is, the transaction must perform CONNECT, OPEN, READ/WRITE, CLOSE, and DISCONNECT within the scope of its execution.
- User applications employing CICS Asynchronous Processing techniques such as the following example cannot change terminals, unless the specified transaction does not access StorHouse:

  ```
  EXEC CICS START (transid) FROM (startdata) TERMID (xxx)
  ```

- All transaction programs that CALL LSMCICS must have the original EIBTRMID that performed the CONNECT.

Synchronous and Asynchronous Functions

Functions can be issued in synchronous or asynchronous form as explained below. The functions described in this chapter are presented in synchronous form. To change a function from synchronous to asynchronous, prefix AS to the function name. For example, the asynchronous form of READ-SEQ is ASREAD-SEQ.

Synchronous Form

In *synchronous form*, control is returned to the user program only when a function has completed. In other words, a request was passed to the StorHouse Subsystem, and the Subsystem returned a response.

For some functions, the StorHouse Subsystem signals completion only after it passes a request to the library device and the library device signals completion back to the StorHouse Subsystem. For other functions, completion means only that the user program can continue as though all processing associated with the function has been completed, even though the Subsystem may have only partially acted on the request. For example, WRITE signals completion when data has been moved from the user buffer to an assembly buffer in the Subsystem.

Asynchronous Form

In *asynchronous form*, control from a function call is returned to the user program as soon as the request has been forwarded to the StorHouse Subsystem. The user must call CHECK prior to using the results of the request. The user can obtain the address

of an Event Control Block (ECB) that is POSTed when the function completes by calling ECBADDR.

Function Statement Format

This chapter shows all function statement formats for COBOL. For FORTRAN, PL/1, assembler (ALC), or C programs, infer the parameter list from the COBOL examples.

PL/1 programs must declare the StorHouse LSMCALL as follows:

```
DECLARE LSMCALL EXTERNAL OPTIONS (ASSEMBLER,INTER);
```

Session Control Functions

Two session control functions allow an application to begin or end a StorHouse session. These functions are:

- CONNECT
- DISCONNECT.

CONNECT and DISCONNECT are described in the following sections.

CONNECT

CONNECT establishes a session with StorHouse. A session must exist before any other functions can be performed. CONNECT requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING CONNECT,C-TOKEN,R-CODE,MESSAGE-FLAG,
                     ACCOUNT,PASSWORD,SM-IDENTIFIER,
                     SUBSYSTEM-IDENTIFIER.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     CONNECT,C-TOKEN,R-CODE,MESSAGE-FLAG,
                     ACCOUNT,PASSWORD,SM-IDENTIFIER,
                     SUBSYSTEM-IDENTIFIER.
```

Working Storage Section for COBOL Program

```
01 CONNECT                    PIC X(16)  VALUE 'CONNECT'.
01 C-TOKEN                    PIC S9(8)  COMP SYNC.
01 R-CODE                     PIC S9(8)  COMP SYNC.
01 MESSAGE-FLAG               PIC S9(8)  COMP SYNC.
01 ACCOUNT                    PIC X(12).
01 PASSWORD                   PIC X(32).
01 SM-IDENTIFIER              PIC X(6).
01 SUBSYSTEM-IDENTIFIER       PIC X(4).
```

Parameter Overview

C-TOKEN — Session identifier set by CONNECT. The application program should not manipulate (in particular, not cause arithmetic conversion to) the result. It should only be used as the C-TOKEN parameter to other function calls related to this session.

R-CODE — Final status from the requested operation; see the following section “Return Codes.”

MESSAGE-FLAG
An integer set to zero or non-zero. If non-zero, MESSAGE-FLAG indicates that the caller requires text messages from all session errors including CONNECT/DISCONNECT function errors. If zero, messages may not be retrievable if the session has terminated.

ACCOUNT
A 12-byte character string containing the StorHouse account identification code (AID) that is used for the session. This field allows only a restricted character set. Lowercase characters may be specified but will be treated as uppercase. The only special characters allowed are _ (underscore) and $.

PASSWORD
A 32-byte character string containing the StorHouse password associated with the account. (See the description of the ACCOUNT parameter.) Only a restricted character set is allowed. A blank character string indicates that no password was specified.

SM-IDENTIFIER
A 6-byte character string identifying the specific StorHouse system to be accessed. If blank, the default or only StorHouse system for the specified subsystem (see below) is accessed.

SUBSYSTEM-IDENTIFIER
A 4-byte character string containing the name for the StorHouse Subsystem. If blank, the default subsystem name LSMS is used. The default may be overridden by a //LSMSssnm DD DUMMY statement inserted into execution JCL, where "ssnm" is the subsystem name to be used.

Return Codes

Any non-zero value indicates that the session was not established. In this case, do not call DISCONNECT. However, if MESSAGE-FLAG was set (non-zero), then call EMSG to retrieve all messages.

Detailed Function Description

The first step in any interaction with StorHouse is to establish a session by calling CONNECT. The session is identified with an account identification code, and security is provided by requiring an associated password. After a successful (return code zero) CONNECT, other StorHouse functions can be performed.

If MESSAGE-FLAG is set (non-zero), the application must call EMSG after the session ends. The dynamic memory allocated for the session is not released until all messages have been returned; that is, EMSG receives return code 3065, which indicates that there are no more messages.

Notes

- CONNECT has no asynchronous form; there is no ASCONNECT.
- A session can be established in one task (under one TCB) and then used in another task; however, only one session-related function can be performed at one time for one session. Serialization between multiple tasks is the responsibility of the user.

  OPEN-SEQ, CREATE-OPEN, and OPEN-VRAM are considered session-related functions.
- Sessions can be shared to the same extent that DCBs can be shared. In general, sessions should not be used across multiple tasks.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, “Sample Program”:

```
PARAGRAPH 100-CONNECT-TO-SM
```

DISCONNECT

DISCONNECT concludes a session by terminating the connection with StorHouse that was established by CONNECT. Standard StorHouse features are required.

Statement Format for COBOL

TSO/Batch/IMS Environment
`CALL ‘LSMCALL’ USING DISCONNECT,C-TOKEN,R-CODE.`

CICS Environment
`CALL ‘LSMCICS’ USING DFHEIBLK,COMMAREA,` `DISCONNECT,C-TOKEN,R-CODE.`

Working Storage Section for COBOL Program

```
01 DISCONNECT     PIC X(16)   VALUE 'DISCONNECT'.
01 C-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
```

Parameter Overview

C-TOKEN The session identifier returned by CONNECT.

R-CODE Final status from the requested operation; see the following section “Return Codes.”

Return Codes

Any non-zero value indicates that the session was not concluded successfully. If DISCONNECT fails, resources allocated by StorHouse support routines, both in the user address space and in the StorHouse Subsystem address space, may not be released.

Detailed Function Description

The final step in any interaction with StorHouse is to conclude a session by calling DISCONNECT. The session is identified by the C-TOKEN variable returned from CONNECT. A successful (return code zero) DISCONNECT concludes the session and releases all resources allocated by StorHouse support functions.

Notes

- A session can be established in one task (under one TCB) and then used in another task; however, DISCONNECT must be issued from the same task (TCB) that issued the CONNECT.
- File operations should be explicitly closed before signing off; otherwise, the data transfer ends with an abort status, and DISCONNECT returns an error (2957).
- If MESSAGE-FLAG was set when the session was established (see CONNECT), then EMSG should always be called following DISCONNECT.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, “Sample Program:”

```
PARAGRAPH 700-DISCONNECT
```

File Operation Functions

The five file operation functions are:

- OPEN-SEQ – opens a non-VRAM file on StorHouse. Non-VRAM files are processed sequentially.
- CREATE-OPEN – creates a new VRAM file on StorHouse and then establishes a data transfer link for writing data to that file.
- OPEN-VRAM – opens a StorHouse VRAM file. VRAM files can be processed sequentially, or individual records can be accessed by record number or by key value, depending on the file access method.
- CHECKPOINT – synchronizes file transfer by ensuring that all previously written records have been received and processed by StorHouse.
- CLOSE – terminates the file operation.

These functions are described in the following sections.

OPEN

OPEN is an obsolete function that is retained only for compatibility with prior releases. New applications that process sequential files should use OPEN-SEQ. For information about OPEN, refer to a prior version of this document.

OPEN-SEQ

OPEN-SEQ establishes a data transfer link between the user program and StorHouse, sets the direction of the data flow, and identifies the file that will be referenced. This function allows sequential transfer of complete files, using the read or write functions. OPEN-SEQ requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING OPEN-SEQ,C-TOKEN,R-CODE,MESSAGE-FLAG,
                     O-TOKEN,MODE,FILE-NAME,VERSION,
                     FILE-PASSWORDS,GROUP-NAME,
                     GROUP-PASSWORDS,FILE-LOCATION,
                     FILE-ATTRIB,FILE-OPTIONS.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     OPEN-SEQ,C-TOKEN,R-CODE,MESSAGE-FLAG,
                     O-TOKEN,MODE,FILE-NAME,VERSION,
                     FILE-PASSWORDS,GROUP-NAME,
                     GROUP-PASSWORDS,FILE-LOCATION,
                     FILE-ATTRIB,FILE-OPTIONS.
```

Working Storage Section for COBOL Program

```
01 OPEN-SEQ                    PIC X(16)  VALUE 'OPEN-SEQ'.
01 C-TOKEN                     PIC S9(8)  COMP SYNC.
01 R-CODE                      PIC S9(8)  COMP SYNC.
01 MESSAGE-FLAG                PIC S9(8)  COMP SYNC.
01 O-TOKEN                     PIC S9(8)  COMP SYNC.
01 MODE                        PIC X(6).
01 FILE-NAME                   PIC X(56).
01 VERSION                     PIC S9(8)  COMP SYNC.
01 FILE-PASSWORDS.
   05 FILE-READ-PASSWORD       PIC X(8)   VALUE SPACES.
   05 FILE-WRITE-PASSWORD      PIC X(8)   VALUE SPACES.
   05 FILE-DELETE-PASSWORD     PIC X(8)   VALUE SPACES.
01 GROUP-NAME                  PIC X(8).
```

```
01 GROUP-PASSWORDS.
   05 GROUP-READ-PASSWORD     PIC X(8)   VALUE SPACES.
   05 GROUP-WRITE-PASSWORD    PIC X(8)   VALUE SPACES.
   05 GROUP-DELETE-PASSWORD   PIC X(8)   VALUE SPACES.
01 FILE-LOCATION.
   05 VOLUMESET-NAME          PIC X(8).
   05 FILESET-NAME            PIC X(8).
01 FILE-ATTRIB.
   05 FATTR-LIST-SIZE         PIC S9(8)  COMP SYNC VALUE 9.
   05 FATTR-FILE-SIZE         PIC S9(8)  COMP SYNC.
   05 FATTR-MAX-RECORD-LEN    PIC S9(8)  COMP SYNC.
   05 FATTR-TRANSPORT-FLAG    PIC S9(8)  COMP SYNC.
   05 FATTR-DATA-XLATE-FLAG   PIC S9(8)  COMP SYNC.
   05 FATTR-FIXED-RECORD-FL   PIC S9(8)  COMP SYNC.
   05 FATTR-CC-ANSI-FLAG      PIC S9(8)  COMP SYNC.
   05 FATTR-CC-MACH-FLAG      PIC S9(8)  COMP SYNC.
   05 FATTR-BLOCK-SIZE        PIC S9(8)  COMP SYNC.
   05 FATTR-RETENTION-INTERVAL PIC S9(8) COMP SYNC.
01 FILE-OPTIONS.
   05 FOPTS-LIST-SIZE         PIC S9(8)  COMP SYNC VALUE 8.
   05 FOPTS-LOCK              PIC S9(8)  COMP SYNC.
   05 FOPTS-WAIT              PIC S9(8)  COMP SYNC.
   05 FOPTS-ATF               PIC S9(8)  COMP SYNC.
   05 FOPTS-EDC               PIC S9(8)  COMP SYNC.
   05 FOPTS-LIMIT             PIC S9(8)  COMP SYNC.
   05 FOPTS-NEW               PIC S9(8)  COMP SYNC.
   05 FOPTS-UNLOCK            PIC S9(8)  COMP SYNC.
   05 FOPTS-VTF               PIC S9(8)  COMP SYNC.
```

Parameter Overview

C-TOKEN — Session identifier (connect token).

R-CODE — Final status from the requested operation; see the following section “Return Codes.”

MESSAGE-FLAG — An integer set to zero or non-zero. If non-zero, MESSAGE-FLAG indicates that the caller requires indicative text messages from all data transfer operation errors including CLOSE errors. If zero, messages may not be retrievable if the data transfer has terminated.

O-TOKEN — A variable that is set to the data transfer operation identifier (open token). The application program should not manipulate (in particular, not cause arithmetic conversion to) the result; it should only be used as the O-TOKEN parameter to other function calls for this file.

MODE — A 6-byte character string that identifies the file reference mode. Valid MODE values are READ and WRITE.

FILE-NAME — A 56-byte character string that contains either the StorHouse file name or the DDname to be referenced. If the DDname is specified, the string must begin with the

characters "DD=". In this case, the file name used will be the DSNAME specified on the named DD statement. File names shorter than 56 characters must be padded on the right with blanks.

VERSION — File version number, which applies only to READ operations. Zero is the default (most current) version. A negative value indicates a relative version number. Positive values are not supported.

FILE-PASSWORDS — An array of three 8-character variables containing the read, write, and delete passwords associated with the file name. The array entry for a password that is not supplied must be all blanks.

GROUP-NAME — An 8-byte character string that identifies the file access group. If the file is stored under the user's default group, this parameter need not be supplied; that is, its value must be all blanks.

GROUP-PASSWORDS — An array of three 8-character variables containing the read, write, and delete passwords for the group. The GROUP-PASSWORDS array has the same format as the FILE-PASSWORDS array.

FILE-LOCATION — An array of two 8-character variables containing the file's destination volume set name and file set name, respectively. If a default is used, FILE-LOCATION should contain all blanks. This parameter applies to WRITE operations only.

FILE-ATTRIB — An array of 32-bit integers that provides file attributes. The caller specifies values for the first entry in the array, FATTR-LIST-SIZE, and for FATTR-BLOCK-SIZE. FATTR-LIST-SIZE contains the number of the other elements in the file attributes array. To supply or retrieve all available file attributes, set FATTR-LIST-SIZE to 9.

For MODE=WRITE, the caller specifies file attributes. FATTR-FILE-SIZE is required. All other attributes are optional. The caller must supply a value for all attributes included in the array. Attributes not included in the array assume a value of 0, which indicates use of the default. (The actual default value may not equal zero.)

For MODE=READ, all file attributes, except for FATTR-BLOCK-SIZE, are returned to the caller.

For flag values, a negative value implies the opposite of the positive value; zero indicates that the default is used.

The elements in the FILE-ATTRIB array must be listed in the following order:

- FATTR-LIST-SIZE – number of other elements in the array.
- FATTR-FILE-SIZE – total file size in bytes. This estimate must be larger than the actual number of bytes that will be transferred.
- FATTR-MAX-RECORD-LEN – maximum length for any record in the file.

- FATTR-TRANSPORT-FLAG – flag value; if positive, the file is in a transportable format that can be retrieved by dissimilar host systems.
- FATTR-DATA-XLATE-FLAG – flag value; if positive, data will be stored as ASCII characters. The data is translated from EBCDIC to ASCII when the file is stored on StorHouse and translated from ASCII to EBCDIC when retrieved by the host.
- FATTR-FIXED-RECORD-FL – flag value; if positive, the records are fixed length.
- FATTR-CC-ANSI-FLAG – flag value; if positive, the first character of each record is a print carriage control character of the FORTRAN (or ANSI) type.
- FATTR-CC-MACH-FLAG – flag value; if positive, the first character of each record is a print carriage control character of "machine" type.
- FATTR-BLOCK-SIZE – size in bytes of a buffer area used by the StorHouse software to block user records prior to moving data to or from the StorHouse Subsystem. The caller does not have to reserve this area because it is GETMAINed and FREEMAINed by the StorHouse software.
- FATTR-RETENTION-INTERVAL – the retention period for the file. Valid values are:
    - Non-zero positive integer (for example, 60), which indicates the number of days to retain the file.
    - -2, which indicates to retain the file forever.
    - -1, which indicates zero days, or no retention.
    - 0, which indicates the retention period is not specified at the file level and assumes the default value. StorHouse determines the default value as follows:

      If the file's resident file set has a retention attribute equal to -2, -1, or a specified number of days, the file set retention attribute determines the file retention attribute.

      If the file's resident file set has a retention attribute of 0, the RETENTION_MODE system parameter determines the file retention attribute. If RETENTION_MODE is set to BASIC, the file retention is ZERO. If RETENTION_MODE is set to STRICT, the file retention is FOREVER.

The caller specifies the value for block size. A value of zero defaults to the site-selected value for block size. A value of 1 to 256 causes buffering to be bypassed.

The recommended block size is between 32,000 and 100,000 bytes and should contain 2 or more records plus 4 bytes.

FILE-OPTIONS

An array of 32-bit integers that provide file options. These options correspond to the StorHouse GET and PUT command modifiers. For information about GET and PUT, refer to the *Command Language Reference Manual*.

The caller sets the first entry in the FILE-OPTIONS array, FOPTS-LIST-SIZE, to the number of the other elements in the array. To access all options, set FOPTS-LIST-SIZE to 8.

Other entries are either integer or flag values.

- Integers are either positive or 0. Zero indicates that the default value is used.

  Note: The actual default value may not equal zero.

- Flags are any positive value (indicates "true" and the option is selected), any negative value (indicates the opposite of "true"), or zero (indicates use of the default).

The caller must supply a value for all attributes included in the array. Attributes not included in the array assume a value of 0, which indicates use of the default.

The elements in the FILE-OPTIONS array are:

- FOPTS-LIST-SIZE – number of other elements in the array.
- FOPTS-LOCK – lock flag. A positive value indicates that the file is to be explicitly locked; it will remain locked after the file operation completes.
- FOPTS-WAIT – wait for file lock flag.
  - For READ operations, a positive value indicates that the data transfer operation should wait for a locked file to be unlocked.
  - For WRITE operations, this field is no longer used. It is not necessary to change existing code. For new programs, set this field to 0.
- FOPTS-ATF – Access Time Factor (ATF). ATF can be 1, 2, or 3. This field is used for WRITE operations only.
- FOPTS-EDC – error detection code identifier. FOPTS-EDC can be a positive integer equal to 1 or 2; zero to indicate use of the default (which is recommended); or negative to indicate that no EDC will be generated for data in the file. This field is used for WRITE operations only.
- FOPTS-LIMIT – file version LIMIT value. FOPTS-LIMIT can be a positive integer between 1 and 32768. This field is used for WRITE operations only.

- FOPTS-NEW – new file flag. A positive value indicates that a previous version of the file (same group and file name) must not exist in StorHouse. This field is used for WRITE operations only.

- FOPTS-UNLOCK – unlock flag. This field is obsolete. It is not necessary to change existing code. For new programs, set FOPTS-UNLOCK to 0.

- FOPTS-VTF – Vulnerability Time Factor (VTF). VTF is an integer equal to 1, 2, 3, or 4. A value of 1 indicates /VTF=NEVER; 2 indicates /VTF=NEXT; 3 indicates /VTF=NOW; and 4 indicates /VTF=DIRECT. This field is used for WRITE operations only.

  Refer to the *Command Language Reference Manual* for information about the VTF attribute.

Return Codes

Any non-zero value indicates that the file was not opened. In this case, any other StorHouse functions relating to this file should not be issued. In particular, CLOSE will fail due to an invalid O-TOKEN.

Detailed Function Description

OPEN-SEQ is used to open a non-VRAM file on StorHouse. The StorHouse file name is identified by the value of FILE-NAME, and the type of processing is provided by the value of MODE. C-TOKEN contains the session identifier returned by CONNECT. OPEN-SEQ returns a file identifier in the O-TOKEN variable. After a successful OPEN-SEQ (that is, a return code of zero), other StorHouse functions relating to this file can be performed.

A StorHouse file opened with OPEN-SEQ can only be processed sequentially. Facilities implemented by the optional VRAM component, such as reading a record by record number, cannot be used.

If MESSAGE-FLAG is set (non-zero), the application must call EMSG after the file is closed. The dynamic memory allocated for the transfer operation is not released until all messages have been returned; that is, a 3065 return code, indicating no more messages, has been received from EMSG.

Notes

- Each OPEN-SEQ establishes a transfer link and returns a file identifier (O-TOKEN). It is the user's responsibility to maintain the integrity of the open tokens.

- A session can be established in one task (under one TCB) and then used in another task; however, only one session-related function can be performed at one time for one session. Serialization between multiple tasks is the user's responsibility.

  OPEN-SEQ must be considered a session-related function.

- If a file is opened and closed under one TCB and read or written from another TCB, the two tasks must share Subpool 0 storage. If one of these tasks is a subtask of the other, this is accomplished by the SZERO=YES operand on the ATTACH MACRO (this is the default value).

- If the return code is not zero and the associated messages (if any) are to be retrieved, EMSG should be called specifying the C-TOKEN rather than the O-TOKEN.

- Refer to Appendix C for a discussion of programming guidelines for using multiple open statements.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, "Sample Program":

```
PARAGRAPH 1000-OPEN-NONVRAM
```

CREATE-OPEN

CREATE-OPEN creates a new VRAM file on StorHouse, and then establishes a data transfer link for writing data to that file. CREATE-OPEN is equivalent to issuing a StorHouse Command Language CREATE FILE command followed by OPEN-VRAM in mode APPEND. CREATE-OPEN requires the StorHouse VRAM component.

CREATE-OPEN requires RECORD privilege. For more information about StorHouse privileges, refer to the *Command Language Reference Manual.*

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING CREATE-OPEN,C-TOKEN,R-CODE,
                     MESSAGE-FLAG,O-TOKEN,FILE-NAME
                     FILE-PASSWORD,GROUP-NAME,
                     GROUP-PASSWORD,MODEL-FILE-NAME,
                     FILE-LOCATION,FILE-ATTRIB.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     CREATE-OPEN,C-TOKEN,R-CODE,
                     MESSAGE-FLAG,O-TOKEN,FILE-NAME,
                     FILE-PASSWORD,GROUP-NAME,
                     GROUP-PASSWORD,MODEL-FILE-NAME,
                     FILE-LOCATION,FILE-ATTRIB.
```

Working Storage Section for COBOL Program

```
01 CREATE-OPEN            PIC X(16)  VALUE 'CREATE-OPEN'.
01 C-TOKEN                PIC S9(8)  COMP SYNC.
01 R-CODE                 PIC S9(8)  COMP SYNC.
01 MESSAGE-FLAG           PIC S9(8)  COMP SYNC.
01 O-TOKEN                PIC S9(8)  COMP SYNC.
01 FILE-NAME              PIC X(56).
01 FILE-PASSWORD          PIC X(8).
01 GROUP-NAME             PIC X(8).
01 GROUP-PASSWORD         PIC X(8).
01 MODEL-FILE-NAME        PIC X(56).
```

```
01 FILE-LOCATION.
   05 VOLUMESET-NAME       PIC X(8).
   05 FILESET-NAME         PIC X(8).
01 FILE-ATTRIB.
   05 FATTR-LIST-SIZE      PIC S9(8)  COMP SYNC VALUE 10.
   05 FATTR-BLOCK-SIZE     PIC S9(8)  COMP SYNC.
   05 FATTR-CHECKPOINT     PIC S9(8)  COMP SYNC.
   05 FATTR-FILE-SIZE      PIC S9(8)  COMP SYNC.
   05 FATTR-DATA-XLATE     PIC S9(8)  COMP SYNC.
   05 FATTR-ATF            PIC S9(8)  COMP SYNC.
   05 FATTR-CACHE          PIC S9(8)  COMP SYNC.
   05 FATTR-EDC            PIC S9(8)  COMP SYNC.
   05 FATTR-LIMIT          PIC S9(8)  COMP SYNC.
   05 FATTR-VTF            PIC S9(8)  COMP SYNC.
   05 FATTR-RETENTION-INTERVAL PIC S9(8)  COMP SYNC.
```

Parameter Overview

C-TOKEN The session identifier returned by CONNECT.

R-CODE Final status from the requested operation; see the following section “Return Codes.”

MESSAGE-FLAG An integer set to zero or non-zero. If non-zero, this flag indicates that the caller requires text messages from all data transfer errors including CLOSE function errors. If zero, messages may not be retrievable after CLOSE has been issued.

O-TOKEN Variable set by CREATE-OPEN to the file identifier. The application program should not manipulate (in particular, not cause arithmetic conversion to) the result; it should only be used as the O-TOKEN parameter to other function calls for this file.

FILE-NAME A 56-byte character string that contains either the StorHouse file name or the DDname to be referenced. If the DDname is specified, the string must begin with the characters DD=. In this case, the file name used will be the DSNAME specified on the named DD statement. File names shorter than 56 characters must be padded on the right with blanks.

FILE-PASSWORD An 8-character variable containing the write password for the file. This password must match the write password for the model file (see MODEL-FILE-NAME) unless the user has the privilege to bypass file passwords. If the user has the privilege to bypass file passwords, this value is not used unless there is no model file. All other passwords for the new file will be copied from the passwords defined for the model file. If no write password is defined for the model file, this variable should be set to all blanks.

If no model file name is provided, the FILE-PASSWORD value becomes the new read, write, and delete passwords for the newly created file. The file password value also supplies the write and delete passwords for any existing version of that file.

GROUP-NAME

An 8-byte character string that identifies the file access group for the new file and for the model file (see MODEL-FILE-NAME). If the file is stored under the account's default group, this parameter may be specified as all blanks. SETGROUP privilege is required to specify any group other than the user's default.

GROUP-PASSWORD

An 8-character variable containing the write password for the file access group. If no write password is defined for the group, this variable should be set to all blanks.

MODEL-FILE-NAME

A 56-byte character string that contains either the StorHouse file name or the DDname to be referenced. If the DDname is specified, the string must begin with the characters DD=. In this case, the file name used will be the DSNAME specified on the named DD statement. File names shorter than 56 characters must be padded on the right with blanks.

The model file must already exist on StorHouse. File characteristics for the new file (whose name is given by FILE-NAME) are determined by copying the characteristics of the model file. These characteristics are overridden by non-default values in the FILE-ATTRIB array.

Only RECORD type files can be created without a model file specification. If blanks are specified for the model file name, then file attributes are determined only from the FILE-ATTRIB array.

MODEL-FILE-NAME must not be the same as FILE-NAME. That is, a prior version of a file cannot be used as a model for a new version of the same file.

FILE-LOCATION

An array of two, 8-character variables containing the file's destination volume set name and file set name. If a variable contains all blanks, the default value associated with the StorHouse account is used. If the account's default value is also blank, the value is copied from the model file.

FILE-ATTRIB

An array of 32-bit integers that provide file attributes. The first entry in the array must be set to the number of other elements in the array. To provide all attributes, set FATTR-LIST-SIZE to 10.

Other entries in the array are either integers or flag values.

- Integers are either positive or 0. Zero indicates that the default is used.

  Note: The actual default value may not equal zero.

- Flags have one of three values:

  - Positive indicates true (the option is selected).
  - Negative indicates false (the option is not selected).
  - Zero indicates use of the default value.

The caller must supply a value for all attributes included in the array. Attributes not included in the array assume a value of 0. Either a file size or checkpoint must be supplied.

Non-default values override attributes determined from the model file. If no model file name is specified, non-default values override normal StorHouse file attribute defaults.

The elements of the FILE-ATTRIB array are:

- FATTR-LIST-SIZE – the number of other elements in the array.

- FATTR-BLOCK-SIZE – the size in bytes of a buffer area used by the Callable Interface to block user records prior to moving data to the StorHouse Subsystem. The caller does not have to reserve this area because it is GETMAINed and FREEMAINed by the StorHouse software.

  The caller supplies the value for block size. A value of 0 causes a site-selected value to be used for block size. A value of 1 to 256 causes buffering to be bypassed. The recommended block size is between 32,000 and 100,000 bytes and should be large enough to contain two or more records plus four bytes.

- FATTR-CHECKPOINT – a checkpoint number at which file processing should be restarted. For normal (non-restart) operations, 0 must be specified. If a non-zero checkpoint value is specified, then the remaining entries in this attribute array are ignored.

- FATTR-FILE-SIZE – the number of bytes of storage space (in units of 1000 bytes) allocated whenever a file is opened for an append operation and whenever a checkpoint is issued. The value must contain enough space for the largest extent set that is written. This extent set includes a data extent, a DF extent, and for KEYED files, a K extent. A file size must always be specified (non-zero) for file creation (in other words, FATTR-CHECKPOINT value is zero). Refer to the *Command Language Reference Manual* for more information about specifying file size.

- FATTR-DATA-XLATE – a flag value; if positive, data is stored as ASCII characters. The data is translated from EBCDIC to ASCII as it is transferred to StorHouse and is translated from ASCII to the local code for the host as it is transferred to the host (for IBM mainframes the local code is EBCDIC).

- FATTR-ATF – Access Time Factor, a positive integer equal to 1, 2, or 3. Refer to the *Command Language Reference Manual* for additional information.

- FATTR-CACHE – the number of records cached by StorHouse during read operations for files opened with a mode of READ or UPDATE and a method including RECORD or KEYED. The cache value may be specified as a negative value. A negative value turns off caching and ignores the cache specification for the model file.

- FATTR-EDC – a flag value; if positive error detection coding is enabled. If negative, EDC is disabled.

- FATTR-LIMIT – the file version limit value, a positive integer between 1 and 32768, or 0 for default.

- FATTR-VTF – Vulnerability Time Factor, an integer equal to 2, 3, or 4.

  - 2 indicates a VTF of NEXT
  - 3 indicates a VTF of NOW
  - 4 indicates a VTF of DIRECT.

  Refer to the *Command Language Reference Manual* for additional information about the VTF attribute.

- FATTR-RETENTION-INTERVAL – the retention period for the file. Valid values are:

  - Non-zero positive integer (for example, 60), which indicates the number of days to retain the file.

  - -2, which indicates to retain the file forever.

  - -1, which indicates zero days, or no retention.

  - 0, which indicates the retention period is not specified at the file level and assumes the default value. StorHouse determines the default value as follows:

    If the file's resident file set has a retention attribute equal to -2, -1, or a specified number of days, the file set retention attribute determines the file retention attribute.

    If the file's resident file set has a retention attribute of 0, the RETENTION_MODE system parameter determines the file retention attribute. If RETENTION_MODE is set to BASIC, the file retention is ZERO. If RETENTION_MODE is set to STRICT, the file retention is FOREVER.

Return Codes

2629 Indicates that the caller supplied an invalid checkpoint number.

2635 May be caused by the following errors:

- CREATE-OPEN was used to create a new version of the model file.
- The specified model file is open for write or update by another user.
- A user tried to CREATE-OPEN a file whose highest version was already in use.

Refer to the error message text retrieved by EMSG to identify the specific cause of error.

Any Other Non-Zero Code

Indicates that the file was not created and is not open. Any other StorHouse functions relating to this file should not be issued. In particular, CLOSE will fail because of an invalid O-TOKEN.

Detailed Function Description

CREATE-OPEN creates a VRAM file on StorHouse and builds an open data transfer path to allow WRITE operations to that file. The VRAM file is identified by the value of FILE-NAME. C-TOKEN is the session identifier returned by CONNECT. CREATE-OPEN returns a file identifier in the O-TOKEN variable. After a successful CREATE-OPEN (a return code of zero), operations for this file can be performed.

Notes

- Each CREATE-OPEN establishes another transfer link and returns another file identifier (O-TOKEN). It is the responsibility of the user to maintain the integrity of the open tokens.
- If the amount of space indicated by the FATTR-FILE-SIZE variable cannot be allocated, StorHouse returns an error code. Refer to the *Command Language Reference Manual* (CREATE FILE command) for information about how to estimate VRAM file sizes.
- If the return code is non-zero, there may be associated error messages. These messages can be retrieved using the EMSG function. The C-TOKEN (not the O-TOKEN) must be specified in the EMSG call.
- A session can be established in one task (under one TCB) and then used in another task; however, only one session-related function can be performed at one time for one session. Serialization between multiple tasks is the user's responsibility. CREATE-OPEN must be considered a session-related function.
- If a file is opened and closed under one TCB and written from another TCB, the two tasks must share Subpool 0 storage. If one of these tasks is a subtask of the other, this is accomplished by the SZERO=YES operand on the ATTACH MACRO (this is the default value).
- Generally, model files should be created only for use as models, not for use as data files. When a file is used as a model, it is referenced (mounted) as part of CREATE-OPEN processing. If the model is on optical storage, a physical platter mount may be required. Allocating models as empty files on level F storage prevents this extra platter mount.

- The additional technical information about programming guidelines for using multiple open statements and checkpoints supplied in Appendix C, "Checkpoint/Restart and Programming Guidelines," also applies to CREATE-OPEN.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, "Sample Program."

OPEN-VRAM

OPEN-VRAM establishes a data transfer link between the user program and StorHouse, sets the direction of the data flow, indicates the type of processing that will be performed, and identifies the file that will be referenced. OPEN-VRAM requires the StorHouse VRAM Component.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING OPEN-VRAM,C-TOKEN,R-CODE,
                     MESSAGE-FLAG,O-TOKEN,MODE,
                     ACCESS-METHOD,FILE-NAME,
                     REVISION,FILE-PASSWORDS,
                     GROUP-NAME,GROUP-PASSWORDS,
                     REL-REC-NUM,FILE-ATTRIB.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     OPEN-VRAM,C-TOKEN,R-CODE,
                     MESSAGE-FLAG,O-TOKEN,MODE,
                     ACCESS-METHOD,FILE-NAME,
                     REVISION,FILE-PASSWORDS,
                     GROUP-NAME,GROUP-PASSWORDS,
                     REL-REC-NUM,FILE-ATTRIB.
```

Working Storage Section for COBOL Program

```
01 OPEN-VRAM                 PIC X(16)  VALUE 'OPEN-VRAM'.
01 C-TOKEN                   PIC S9(8)  COMP SYNC.
01 R-CODE                    PIC S9(8)  COMP SYNC.
01 MESSAGE-FLAG              PIC S9(8)  COMP SYNC.
01 O-TOKEN                   PIC S9(8)  COMP SYNC.
01 MODE                      PIC X(6).
01 ACCESS-METHOD              PIC X(24).
01 FILE-NAME                  PIC X(56).
01 REVISION                   PIC S9(8)  COMP SYNC.
01 FILE-PASSWORDS.
   05 FILE-READ-PASSWORD      PIC X(8).
```

```
   05 FILE-WRITE-PASSWORD     PIC X(8).
01 GROUP-NAME                 PIC X(8).
01 GROUP-PASSWORDS.
   05 GROUP-READ-PASSWORD     PIC X(8).
   05 GROUP-WRITE-PASSWORD    PIC X(8).
01 REL-REC-NUM                PIC S9(8)  COMP SYNC.
01 FILE-ATTRIB.
   05 FATTR-LIST-SIZE         PIC S9(8)  COMP SYNC VALUE 8.
   05 FATTR-MAX-RECORD-LEN    PIC S9(8)  COMP SYNC.
   05 FATTR-LAST-PHY-REC-NUM  PIC S9(8)  COMP SYNC.
   05 FATTR-LAST-LOG-REC-NUM  PIC S9(8)  COMP SYNC.
   05 FATTR-FILE-REVISION-NUM PIC S9(8)  COMP SYNC.
   05 FATTR-FILE-TYPE         PIC S9(8)  COMP SYNC.
   05 FATTR-BLOCK-SIZE        PIC S9(8)  COMP SYNC.
   05 FATTR-VERSION           PIC S9(8)  COMP SYNC.
   05 FATTR-CHECKPT           PIC S9(8)  COMP SYNC.
```

Parameter Overview

C-TOKEN The session identifier returned by CONNECT.

R-CODE Final status from the requested operation; see the following section “Return Codes.”

MESSAGE-FLAG An integer set to zero or non-zero. If non-zero, this flag indicates that the caller requires text messages from all data transfer errors, including CLOSE function errors. If zero, messages may not be retrievable after CLOSE has been issued.

O-TOKEN Variable set by OPEN-VRAM to the file identifier. The application program should not manipulate (in particular, not cause arithmetic conversion to) the result; it should only be used as the O-TOKEN parameter to other function calls for this file.

MODE A 6-byte character string that identifies the file reference mode. Valid MODE values are READ, UPDATE, and APPEND.

ACCESS-METHOD A 24-byte character string that contains the type of processing to be performed on the file. The valid types are SEQUENTIAL, RECORD, KEYED, ALL, or a combination of any two or three of SEQUENTIAL, RECORD, and KEYED, separated by commas. The type ALL specifies that all methods are included. Specify ALL for KRA-type VRAM files only. If you specify ALL for RRA-type VRAM files, OPEN-VRAM will fail.

FILE-NAME A 56-byte character string that contains either the StorHouse file name or the DDname to be referenced. If the DDname is specified, the string must begin with the characters “DD=”. In this case, the file name used will be the DSNAME specified on the named DD statement. File names shorter than 56 characters must be padded on the right with blanks.

REVISION An integer set by the user to indicate the file version’s revision number.

- Zero is the default (most current) revision.
- A positive integer indicates an absolute revision number.
- A negative integer indicates a relative revision number.

It is the user's responsibility to track absolute revision numbers.

FILE-PASSWORDS — An array of two, 8-character variables containing the read and write passwords associated with the file name. The array entry for a password that is not supplied must be all blanks.

GROUP-NAME — An 8-byte character string that identifies the file access group. If the file is stored under the user's default group, this parameter need not be supplied; that is, its value must be all blanks.

GROUP-PASSWORDS — An array of two, 8-character variables containing the read and write passwords for the group. The GROUP-PASSWORDS array has the same format as the FILE-PASSWORDS array.

REL-REC-NUM — The relative record number of the first record to be read from StorHouse. This value is only meaningful when MODE=READ and ACCESS-METHOD= SEQUENTIAL.

FILE-ATTRIB — An array of 32-bit integers that provides file attributes. The caller must set the first entry in the array, FATTR-LIST-SIZE, to the number of the other elements in the array. The minimum value allowed is 1. The caller also specifies a value for FATTR-BLOCK-SIZE and may specify a value for FATTR-VERSION, and when applicable, FATTR-CHECKPT. (Refer to the descriptions of FATTR-VERSION and FATTR-CHECKPT.)

All other file attribute values, except for FATTR-CHECKPT, are returned to the caller when the file is opened. The FATTR-CHECKPT value is returned to the caller only if the caller does *both* of the following:

- Supplies a zero value
- Attempts to open a checkpointed, software-disabled file with OPEN-VRAM, MODE=APPEND.

In this case, the returned value in FATTR-CHECKPT is the file's last checkpoint number.

The elements in the file attributes array must be listed in the following order:

- FATTR-LIST-SIZE – the number of the other elements in the array
- FATTR-MAX-RECORD-LEN – the maximum length for any record in the file
- FATTR-LAST-PHY-REC-NUM – the last physical record number in the file
- FATTR-LAST-LOG-REC-NUM – the last logical record number in the file

- FATTR-FILE-REVISION-NUM – the absolute revision number of the file version

- FATTR-FILE-TYPE – the VRAM file type. A value of 0 indicates an RRA file, and a value of 1 indicates a KRA file. VRAM file type is specified when the file is created on StorHouse with the StorHouse Command Language CREATE FILE command. For information about CREATE FILE, refer to the *Command Language Reference Manual*.

- FATTR-BLOCK-SIZE – the size in bytes of a buffer area used by the Callable Interface to block user records prior to moving data to or from the StorHouse Subsystem. It is not necessary for the caller to reserve this area because it is GETMAINed and FREEMAINed by the Callable Interface.

  The caller supplies the value for block size. A value of 0 defaults to the site-selected value for block size. A value of 1 to 256 causes buffering to be bypassed.

  The recommended block size is between 32,000 and 100,000 bytes and should contain two or more records plus four bytes.

  FATTR-BLOCK-SIZE is used only when MODE=APPEND, or when MODE=READ and ACCESS-METHOD=SEQUENTIAL.

- FATTR-VERSION – a user-supplied value that indicates the version of the file to be opened

  - To open the latest version, omit the attribute or supply a zero, which is the default.

  - To open a specific version, supply its relative version number as a negative number (-1 through -32767).

  - Positive values are not supported.

- FATTR-CHECKPT – a value *supplied* by the caller at open (MODE=APPEND) to indicate the checkpoint number where file processing should be restarted. If zero or omitted, a normal (nonrestart) OPEN-VRAM occurs.

  After OPEN-VRAM is issued, the value of FATTR-CHECKPT is *returned* to the caller only if MODE=APPEND, the file being opened is checkpointed and software disabled, and the caller set FATTR-CHECKPT to 0.

Return Codes

2629 Indicates that the caller supplied an invalid checkpoint number.

2630 Indicates that the file was not opened because it is software disabled and that a valid checkpoint exists. The last checkpoint number is returned in FATTR-CHECKPT.

2636	Indicates that the caller supplied a checkpoint number but MODE was not APPEND.
2637	Indicates that the caller attempted to open a noncurrent revision of a file at a checkpoint. Only the current revision of a file can be opened at a checkpoint.
Any Other Non-Zero Code	Indicates that the file was not opened. Any other StorHouse functions relating to this file should not be issued. In particular, CLOSE will fail due to an invalid O-TOKEN.

Detailed Function Description

The OPEN-VRAM function opens a VRAM file in StorHouse. The VRAM file is identified by the value of FILE-NAME, and the type of processing to be performed is provided by FATTR-FILE-TYPE and MODE. C-TOKEN is the session identifier returned by CONNECT. OPEN-VRAM returns a file identifier in the O-TOKEN variable. After a successful OPEN-VRAM (that is, a return code of zero), other StorHouse functions relating to this file can be performed.

If MESSAGE-FLAG is set (non-zero), the application must call EMSG after the data transfer operation is closed. The dynamic memory allocated for the data transfer operation is not released until all messages have been returned; that is, EMSG receives a 3065 return code, indicating no more messages.

Notes

- Each OPEN-VRAM establishes another transfer link and returns another file identifier (O-TOKEN). It is the responsibility of the user to maintain the integrity of the open tokens.

- A VRAM file can be opened with MODE=APPEND either to write records into a newly created (empty) file or to add records to a file that already contains data. The two cases can be distinguished by checking the LAST_PHY_REC_NUM attribute after open; for a new file, this attribute is set to zero.

- By issuing OPEN-VRAM with MODE=APPEND, StorHouse attempts to allocate the amount of space that was specified as the value of the /SIZE modifier on the CREATE FILE command for the file currently being opened. If this amount of space cannot be allocated (for example, the file's destination file set is filled and cannot extend), StorHouse returns an error code. Refer to the *Command Language Reference Manual* (CREATE FILE command) for more information about how to estimate VRAM file size.

- If the caller attempts to open a checkpointed, software-disabled file and does not supply a checkpoint number, OPEN-VRAM returns 2630 as the value of R-CODE and the last checkpoint number in FATTR-CHECKPT. To open the software disabled file at the returned checkpoint, the caller can issue another

OPEN-VRAM (MODE=APPEND) and supply the previously returned checkpoint number as the current value of FATTR-CHECKPT.

Only the current (most recent) revision of a file version may be opened at a checkpoint.

- The following example illustrates how logical and physical record numbers are assigned in a file change. If the last physical record number in a file is record number 8, and record number 8 is deleted, the last physical record number remains 8. The last logical record number is 7. New records appended to the file begin at record number 9.

- If the return code is non-zero and the associated messages (if any) are to be retrieved, EMSG should be called specifying the C-TOKEN rather than the O-TOKEN.

- A session can be established in one task (under one TCB) and then used in another task; however, only one session-related function can be performed at one time for one session. Serialization between multiple tasks is the user's responsibility.

  OPEN-VRAM must be considered a session-related function.

- If a file is opened and closed under one TCB and read or written from another TCB, the two tasks must share Subpool 0 storage. If one of these tasks is a subtask of the other, this is accomplished by the SZERO=YES operand on the ATTACH MACRO (this is the default value).

- Refer to Appendix C for additional technical information about programming guidelines for using multiple open statements and for using OPEN-VRAM and CHECKPOINT.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, "Sample Program":

```
PARAGRAPH 1100-OPEN-VRAM
```

CHECKPOINT

CHECKPOINT synchronizes file transfer by ensuring that all previously written records have been received and processed by StorHouse. CHECKPOINT requires the VRAM StorHouse software component.

Statement Format for COBOL

TSO/Batch/IMS Environment
`CALL 'LSMCALL' USING CHECKPOINT,O-TOKEN,R-CODE,` `RETURN-CKPT-NUM.`

CICS Environment

`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,`
`CHECKPOINT,O-TOKEN,R-CODE,`
`RETURN-CKPT-NUM.`

Working Storage Section for COBOL Program

```
01 CHECKPOINT      PIC X(16)  VALUE 'CHECKPOINT'.
01 O-TOKEN         PIC S9(8)  COMP SYNC.
01 R-CODE          PIC S9(8)  COMP SYNC.
01 RETURN-CKPT-NUM PIC S9(8)  COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier returned by OPEN-VRAM or CREATE-OPEN.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

RETURN-CKPT-NUM — An integer set by StorHouse to the binary number associated with this checkpoint.

Return Codes

Any non-zero value indicates that the file was not successfully checkpointed. No other operation may be performed against a file that returns an error during CHECKPOINT, except for CLOSE.

Detailed Function Description

CHECKPOINT synchronizes file transfer to ensure that all record(s) have been written to StorHouse.

CHECKPOINT returns the checkpoint number (value of RETURN-CKPT-NUM) that must be used to restart the file transfer operation at this position. A data transfer operation (MODE=APPEND only) can be restarted by specifying this checkpoint number in the OPEN-VRAM function (FATTR-CHECKPT parameter) or the CREATE-OPEN function (FATTR-CHECKPOINT).

Notes

- To perform CHECKPOINT with OPEN-VRAM, the value for MODE in the OPEN-VRAM call must have been set to APPEND. ACCESS-METHOD is ignored when MODE=APPEND.
- Refer to Appendix C, “Checkpoint/Restart and Programming Guidelines,” for information about using CHECKPOINT and OPEN-VRAM.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

CLOSE

CLOSE closes the file and terminates the data transfer operation that was started by OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM. CLOSE requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING CLOSE,O-TOKEN,R-CODE,XFER-ABORT-FLAG.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     CLOSE,O-TOKEN,R-CODE,XFER-ABORT-FLAG.
```

Working Storage Section for COBOL Program

```
01 CLOSE           PIC X(16)  VALUE 'CLOSE'.
01 O-TOKEN         PIC S9(8)  COMP SYNC.
01 R-CODE          PIC S9(8)  COMP SYNC.
01 XFER-ABORT-FLAG PIC S9(8)  COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier returned by OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

XFER-ABORT-FLAG — A flag set by the user indicating either that the file transfer has completed normally or that the StorHouse transfer should be aborted.

- A zero value means that this CLOSE indicates end-of-data.
- A non-zero value indicates that StorHouse should abort the data transfer. This prevents a file from being cataloged on StorHouse and also cleans up any buffers that may be in transit.

XFER-ABORT-FLAG is set (non-zero) when the data stream to StorHouse must be terminated because of an error.

Setting XFER-ABORT-FLAG forces a return code of 3000. If CLOSE is being called because of a non-zero return code from READ/WRITE, do not set XFER-ABORT-FLAG. This may cause the return code identifying the actual cause of the failure to be lost.

XFER-ABORT-FLAG is only used for write operations.

Return Codes

Any non-zero value indicates that the file was not closed properly.

After a file write operation, a non-zero return code means that the file cannot be guaranteed to be stored in StorHouse.

Detailed Function Description

The final step in any StorHouse file processing is to close the file, using CLOSE. The file is identified by the O-TOKEN returned from OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

For a sequential write operation, CLOSE indicates end-of-file. All in-transit data buffers are written to StorHouse, and transfer end is signaled. StorHouse completes file storage and directory update operations, and honors the requested VTF level prior to returning operation status. A return code of 0 from CLOSE indicates that the file has been stored in StorHouse.

For a sequential read operation, CLOSE terminates the transfer and flushes any in-transit data buffers. A non-zero return code indicates that all data from the file has not been delivered to the application program.

For record-oriented transfers, CLOSE causes completion of all file and index updates. A return code of 0 indicates that the file state in StorHouse is synchronized with the state expected by the application program.

CLOSE always releases all StorHouse resources used for the transfer operation. If the MESSAGE-FLAG was clear (0) in the OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM function call, then CLOSE also releases all host resources used by the transfer. Otherwise, EMSG must be called to retrieve all indicative text messages before host resources are completely released.

A successful CLOSE (that is, a return code of 0), terminates the data transfer link and closes the file associated with the O-TOKEN.

Notes

- A file can be opened in one task (under one TCB) and then used in another task; however, CLOSE may be issued only from the same task (TCB) that issued OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

- When closing a VRAM file, some record pointers or data must still be posted to the file. If the allocated file size is too small, the file becomes software disabled, and data that was written to the file is lost. Refer to the *Command Language Reference Manual* (CREATE FILE command) for information about how to estimate VRAM file size.

- If MESSAGE-FLAG was set in the open function that began the transfer, then the application should call EMSG following CLOSE until a return code of 3065, indicating no more messages, is received.

- If a non-zero return code is received from the I/O operation, the programmer should:

  - Call the ESMG function

  - Call CLOSE with the XFER-ABORT-FLAG not set

  - Ensure that the return code and messages returned from the ESMG function and CLOSE are logged.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, “Sample Program”:

```
PARAGRAPH 1300-CLOSE-SM-FILE
```

Data Transfer Control Functions

Data transfer control functions can be performed once a session has been established and files have been opened. These functions are:

- READ – requests the next sequential record of a non-VRAM file from StorHouse.
- READ-SEQ – requests the next sequential record from a VRAM file.
- READ-RECORD – requests a record from a VRAM file. The record is identified by its relative record number.
- READ-KEYED – retrieves a record from a VRAM file. The record is identified by user-supplied key information.
- READ-NEXT-KEY – requests the next key entry-sequenced record from a VRAM file.
- WRITE – sends a record to StorHouse.
- WRITE-KEY – transfers an external key record and a data record to StorHouse.
- DELETE– deletes the last record read from a VRAM file.
- CHANGE – changes the last record read in a VRAM file.

The following sections describe these functions.

READ

READ requests the next sequential record of a non-VRAM file from StorHouse. READ requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING READ,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     READ,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN.
```

Working Storage Section for COBOL Program

```
01 READ           PIC X(16)   VALUE 'READ'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 BUFFER-SIZE    PIC S9(8)   COMP SYNC.
01 RETURN-REC-LEN PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN The file identifier initialized by OPEN-SEQ.

R-CODE Final status from the requested operation; see the following section "Return Codes."

BUFFER An area where the data record is placed.

BUFFER-SIZE A user-specified integer value indicating the size, in bytes, of the read BUFFER.

RETURN-REC-LEN An integer value returned by READ, containing the length of the record read from StorHouse.

Return Codes

5650	An end of file was encountered.
2188	The buffer is too small and a truncated record was returned. This is only a warning. It is possible to continue the data transfer operation.
Other Non-Zero Values	A record was not read successfully.

Detailed Function Description

READ allows a user to read the next sequential record from the StorHouse file that was previously opened with OPEN-SEQ.

The StorHouse file is identified by the O-TOKEN returned by OPEN-SEQ. The MODE in the open call must be set to READ. The record is placed into a user-supplied buffer. READ returns the length of the record read.

Notes

- READ updates a file's sequential record position. For more information about file positioning, refer to Chapter 3, "File Positioning."
- Do not use READ for VRAM files; use READ-SEQ instead.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, "Sample Program":

```
PARAGRAPH 310-READ-SM
```

READ-SEQ

READ-SEQ requests the next sequential record from a VRAM file. READ-SEQ requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING READ-SEQ,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN,
                     RETURN-REC-NUM.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     READ-SEQ,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN,
                     RETURN-REC-NUM.
```

Working Storage Section for COBOL Program

```
01 READ-SEQ       PIC X(16)   VALUE 'READ-SEQ'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 BUFFER-SIZE    PIC S9(8)   COMP SYNC.
01 RETURN-REC-LEN PIC S9(8)   COMP SYNC.
01 RETURN-REC-NUM PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier initialized by OPEN-VRAM.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

BUFFER — An area where the data record is placed.

BUFFER-SIZE — A user-specified integer value indicating the size, in bytes, of the read BUFFER.

RETURN-REC-LEN — An integer value returned by READ-SEQ, containing the length of the record read from StorHouse.

RETURN-REC-NUM — An integer value returned by READ-SEQ, containing the record number of the record read from StorHouse.

Return Codes

5650 — An end of file was encountered.

2188 — The buffer is too small and a truncated record was returned. This is only a warning. It is possible to continue the data transfer operation.

Other Non-Zero Values — A record was not read successfully.

Detailed Function Description

READ-SEQ allows a user to read the next sequential record from the VRAM file that was previously opened with OPEN-VRAM. The VRAM file is identified by the O-TOKEN returned by OPEN-VRAM. The record is placed into a user-supplied buffer. READ-SEQ returns the length of the record read and the record number.

Notes

- READ-SEQ updates a file's sequential record position. For more information about file positioning, refer to Chapter 3, "File Positioning."
- To perform this function, the MODE in the OPEN-VRAM call must be set to UPDATE or READ. The ACCESS-METHOD must include SEQUENTIAL.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, "Sample Program."

READ-RECORD

READ-RECORD requests a record from a VRAM file. The record is identified by its relative record number. READ-RECORD requires the StorHouse VRAM Component.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING READ-RECORD,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN,
                     REL-REC-NUM.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     READ-RECORD,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN,
                     REL-REC-NUM.
```

Working Storage Section for COBOL Program

```
01 READ-RECORD    PIC X(16)   VALUE 'READ-RECORD'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 BUFFER-SIZE    PIC S9(8)   COMP SYNC.
01 RETURN-REC-LEN PIC S9(8)   COMP SYNC.
01 REL-REC-NUM    PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN The file identifier initialized by OPEN-VRAM.

R-CODE Final status from the requested operation; see the following section "Return Codes."

BUFFER An area where the data record is placed.

BUFFER-SIZE A user-specified integer value indicating the size, in bytes, of the read BUFFER.

RETURN-REC-LEN — An integer value returned by READ-RECORD, containing the length of the record read from StorHouse.

REL-REC-NUM — A variable containing the relative record number of the record to be read from StorHouse.

Return Codes

2587 — The record number was out of range; the record could not be found.

2588 — The record number was deleted.

2188 — The buffer is too small and a truncated record was returned. This is only a warning. It is possible to continue the data transfer operation.

Any Other Non-Zero Value — A record was not read successfully.

Detailed Function Description

READ-RECORD allows a user to read a relative record from the VRAM file that was previously opened with OPEN-VRAM. The VRAM file is identified by the O-TOKEN returned by OPEN-VRAM. The record is identified by the relative record number. The record read from StorHouse is placed into a user-supplied buffer. READ-RECORD also returns the length of the record read.

Notes

- READ-RECORD updates a file's sequential record position. For more information about file positioning, refer to Chapter 3, "File Positioning."
- To perform this function, the MODE in the OPEN-VRAM call must be set to UPDATE or READ. The ACCESS-METHOD must include RECORD.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, "Sample Program":

```
PARAGRAPH 510-READ-AND-PRINT
```

READ-KEYED

READ-KEYED retrieves a record from a VRAM file. The record is identified by user-supplied key information. READ-KEYED requires the StorHouse VRAM component with the KRA feature.

Statement Format for COBOL

TSO/Batch/IMS Environment

`CALL 'LSMCALL' USING READ-KEYED,O-TOKEN,R-CODE,BUFFER,`
`BUFFER-SIZE,RETURN-REC-LEN,KEY-NAME,`
`KEY-VALUE,KEY-LENGTH,RETURN-REC-NUM.`

CICS Environment

`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,`
`READ-KEYED,O-TOKEN,R-CODE,BUFFER,`
`BUFFER-SIZE,RETURN-REC-LEN,KEY-NAME,`
`KEY-VALUE,KEY-LENGTH,RETURN-REC-NUM.`

Working Storage Section for COBOL Program

```
01 READ-KEYED     PIC X(16)   VALUE 'READ-KEYED'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 BUFFER-SIZE    PIC S9(8)   COMP SYNC.
01 RETURN-REC-LEN PIC S9(8)   COMP SYNC.
01 KEY-NAME       PIC X(56).
01 KEY-VALUE      PIC X(key-length).
01 KEY-LENGTH     PIC S9(8)   COMP SYNC.
01 RETURN-REC-NUM PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN The file identifier initialized by OPEN-VRAM.

R-CODE Final status from the requested operation; see the following section "Return Codes."

BUFFER	An area where the data record is placed.
BUFFER-SIZE	A user-specified integer value indicating the size, in bytes, of the read BUFFER.
RETURN-REC-LEN	An integer value returned by READ-KEYED, containing the length of the record read from StorHouse.
KEY-NAME	The user-supplied name of the key field that is used to find the record. The maximum size for KEY-NAME is 56 bytes. An example of a KEY-NAME is "LASTNAME".
KEY-VALUE	The user-supplied value of the key used to search for the record. An example of a KEY-VALUE is "Kelly".
KEY-LENGTH	A user-supplied integer value indicating the size of KEY-VALUE. The maximum length is 254 characters.
RETURN-REC-NUM	An integer that is set by StorHouse to the record number of the last record read from StorHouse.

Return Codes

2587	No record was found with the supplied key.
2588	The record was deleted.
2188	The buffer is too small and a truncated record was returned. This is only a warning. It is possible to continue the data transfer operation.
Any Other Non-Zero Value	A record was not read successfully.

Detailed Function Description

READ-KEYED reads a record from the VRAM file that was previously opened with OPEN-VRAM. The file is identified by the O-TOKEN returned by OPEN-VRAM.

The record is identified by the key parameters KEY-NAME and KEY-VALUE and is placed into the user-supplied buffer. READ-KEYED also returns the length of the record.

Notes

- READ-KEYED updates a file's sequential and key record positions. For more information about file positioning, refer to Chapter 3, "File Positioning."
- To perform READ-KEYED, the MODE in the OPEN-VRAM call must be set to UPDATE or READ. The ACCESS-METHOD must include KEYED.
- The following is true for KEYSEQUENTIAL files only. If READ-KEYED cannot locate the requested key and returns message XKBADRNO (return code 2587), READ-KEYED maintains the current key record position. A subsequent READ-NEXT-KEYED or READ-SEQUENTIAL will find the record with the next greater key value. This subsequent record may be read but not changed or deleted.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, "Sample Program":

```
PARAGRAPH 610-READ-KEYED
```

READ-NEXT-KEY

READ-NEXT-KEY requests the next key entry sequenced record from a VRAM file. READ-NEXT-KEY requires the StorHouse VRAM component with the KRA feature.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING READ-NEXT-KEY,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN,
                     RETURN-REC-NUM.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     READ-NEXT-KEY,O-TOKEN,R-CODE,BUFFER,
                     BUFFER-SIZE,RETURN-REC-LEN,
                     RETURN-REC-NUM.
```

Working Storage Section for COBOL Program

```
01 READ-NEXT-KEY  PIC X(16)   VALUE 'READ-NEXT-KEY'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 BUFFER-SIZE    PIC S9(8)   COMP SYNC.
01 RETURN-REC-LEN PIC S9(8)   COMP SYNC.
01 RETURN-REC-NUM PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN The file identifier initialized by OPEN-VRAM.

R-CODE Final status from the requested operation; see the following section "Return Codes."

BUFFER An area where the data record is placed.

BUFFER-SIZE A user-specified integer value indicating the size, in bytes, of the read BUFFER.

RETURN-REC-LEN — An integer value returned by READ-NEXT-KEY, containing the length of the record read from StorHouse.

RETURN-REC-NUM — An integer value returned by READ-NEXT-KEY, containing the record number of the record read from StorHouse.

Return Codes

5650 — An end of file was encountered.

2188 — The buffer is too small and a truncated record was returned. This is only a warning. It is possible to continue the data transfer operation.

Any Other Non-Zero Value — A record was not successfully read.

Detailed Function Description

READ-NEXT-KEY allows a user to read the next key entry-sequenced record from the VRAM file that was previously opened with OPEN-VRAM. The VRAM file is identified by the O-TOKEN returned by OPEN-VRAM. The record is placed into a user-supplied buffer. READ-NEXT-KEY returns the length of the record read and the record number.

Notes

- READ-NEXT-KEY updates a file's sequential and key record positions. For more information about file positioning, refer to Chapter 3, "File Positioning."
- To perform READ-NEXT-KEY, the MODE in the OPEN-VRAM call must be set to UPDATE or READ. The ACCESS-METHOD must include KEYED.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, "Sample Program":

```
PARAGRAPH 610-READ-KEYED
```

WRITE

WRITE sends a record to StorHouse. WRITE requires the StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING WRITE,O-TOKEN,R-CODE,BUFFER,
                     RECORD-LENGTH,RETURN-REC-NUM.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     WRITE,O-TOKEN,R-CODE,BUFFER,
                     RECORD-LENGTH,RETURN-REC-NUM.
```

Working Storage Section for COBOL Program

```
01 WRITE          PIC X(16)   VALUE 'WRITE'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 RECORD-LENGTH  PIC S9(8)   COMP SYNC.
01 RETURN-REC-NUM PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier initialized by OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

BUFFER — Buffer containing the record to be written to StorHouse.

RECORD-LENGTH — An integer value containing the length, in bytes, of the record written to StorHouse.

RETURN-REC-NUM — An integer value set by the StorHouse software containing the record number of the record written.

Return Codes

2210 — This is a warning that the record is too short. For a KEYED file, this warning is returned if the record is too short to contain all of its key fields.

Any Other Non-Zero Value — The record was not written to StorHouse.

Detailed Function Description

WRITE allows a user to send a record to a file on StorHouse previously opened with OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

For non-VRAM files, the file is identified by the O-TOKEN returned by OPEN-SEQ. For VRAM files, the file is identified by the O-TOKEN returned by OPEN-VRAM or CREATE-OPEN.

The record is sent from the user-supplied buffer to StorHouse. The record written is the next sequential record in the file.

Notes

- WRITE moves data from the user record area (BUFFER) to internal buffers controlled by the Callable Interface. WRITE may return to the caller without actually transferring all user data to StorHouse. Therefore, you can guarantee that the data is stored in StorHouse only after a successful CLOSE or CHECKPOINT.
- Any insufficient space error during a write to a VRAM file leaves the file software disabled. Any data that was written to the file is lost.
- To perform WRITE on a VRAM file, you first must call either OPEN-VRAM with MODE set to APPEND and ACCESS-METHOD set to any valid value or call CREATE-OPEN. To perform WRITE on a non-VRAM file, call OPEN-SEQ with MODE set to WRITE.
- WRITE and WRITE-KEY can be used in the same session.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, “Sample Program”:

```
PARAGRAPH 1200-WRITE-TO-SM
```

WRITE-KEY

WRITE-KEY transfers an external key record and a data record to StorHouse. WRITE-KEY requires the StorHouse VRAM component and KRA feature.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING WRITE-KEY,O-TOKEN,R-CODE,BUFFER,
                     RECORD-LENGTH,KEY,KEY-LENGTH,
                     RETURN-REC-NUM.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     WRITE-KEY,O-TOKEN,R-CODE,BUFFER,
                     RECORD-LENGTH,KEY,KEY-LENGTH,
                     RETURN-REC-NUM.
```

Working Storage Section for COBOL Program

```
01 WRITE-KEY      PIC X(16)   VALUE 'WRITE-KEY'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC S9(8).
01 RECORD-LENGTH  PIC S9(8).
01 KEY            PIC X(key-length).
01 KEY-LENGTH     PIC S9(8).
01 RETURN-REC-NUM PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier initialized by OPEN-VRAM or CREATE-OPEN.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

BUFFER — The buffer containing the data record to be written to StorHouse.

RECORD-LENGTH — An integer value containing the length, in bytes, of the data record written to StorHouse.

KEY — The external key associated with the data record.

KEY-LENGTH — The length, in bytes, of the external key.

RETURN-REC-NUM — A returned integer value set to the record number associated with the data record.

Return Codes

2210 — A warning indicating that the external key record is too short to contain all of its key fields.

Any Other Non-Zero Values — The record or external key was not written to StorHouse.

Detailed Function Description

WRITE-KEY writes an external key record and a data record to a StorHouse file that is identified by the O-TOKEN returned by OPEN-VRAM or CREATE-OPEN. The file must have been created either with the CREATE FILE command using the /EXTERNAL modifier, or with CREATE-OPEN using a model file with external keys. Refer to the *Command Language Reference Manual* for information about CREATE FILE. The data record is sent from the user-supplied buffer to StorHouse and becomes the next sequential record in the file.

Notes

- WRITE-KEY moves data from the user record area (BUFFER) to internal buffers maintained by the Callable Interface. WRITE-KEY may return to the caller without actually transferring all of the user data to StorHouse. Therefore, data can only be guaranteed to be stored in StorHouse after a successful CLOSE or CHECKPOINT.

- To perform WRITE-KEY, the file must be opened either using OPEN-VRAM with a MODE of APPEND, or using CREATE-OPEN with a model file with external keys.

- The Callable Interface considers a WRITE-KEY with a KEY-LENGTH of 0 the same as a WRITE. WRITE and WRITE-KEY can be used in the same session.

- To write a data record and no external key record, specify a value of 0 for KEY-LENGTH. For example, to write one external key record and five associated data records, issue WRITE-KEY to write the external key record and the first data

record. Then issue WRITE-KEY four times with a specified KEY-LENGTH of 0 to write the remaining four data records.

- The actual external key record cannot be accessed by an application. Key information is extracted from the record and stored in a key data base on StorHouse. Therefore, users cannot change an external key record once it is written or read an external key file to determine the keys.

- By definition, external keys are external to, or not part of, the data record. Therefore, data records associated with a given external key should contain control information that allows an application to determine when it has processed the last data record belonging to that external key.

- An insufficient space error during a write to a VRAM file leaves the file software disabled. Any data that was written to the file is lost.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

DELETE

DELETE deletes the last record read from a VRAM file. DELETE requires the StorHouse VRAM component.

Statement Format for COBOL

TSO/Batch/IMS Environment
`CALL 'LSMCALL' USING DELETE,O-TOKEN,R-CODE.`

CICS Environment
`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,` `DELETE,O-TOKEN,R-CODE.`

Working Storage Section for COBOL Program

```
01 DELETE         PIC X(16)   VALUE 'DELETE'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier initialized by OPEN-VRAM.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

Return Codes

2612 — A return code of 2612 indicates that an attempt was made to delete a record without reading the record first.

Any Non-Zero Value — Any non-zero value indicates that a record was not deleted from StorHouse.

Detailed Function Description

DELETE allows a user to delete the last record read from a StorHouse file previously opened with OPEN-VRAM. The VRAM file is identified by the O-TOKEN returned by OPEN-VRAM.

Note

To perform DELETE, the MODE in the OPEN-VRAM call must be set to UPDATE. For a more complete discussion of MODE and the associated ACCESS-METHOD parameter, refer to the OPEN-VRAM function description.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

CHANGE

CHANGE changes the last record read in a VRAM file. CHANGE requires the StorHouse VRAM component.

Statement Format for COBOL

TSO/Batch/IMS Environment
`CALL 'LSMCALL' USING CHANGE,O-TOKEN,R-CODE,BUFFER,` `RECORD-LENGTH.`

CICS Environment

`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,`
`CHANGE,O-TOKEN,R-CODE,BUFFER,`
`RECORD-LENGTH.`

Working Storage Section for COBOL Program

```
01 CHANGE         PIC X(16)   VALUE 'CHANGE'.
01 O-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 BUFFER         PIC X(buffer-size).
01 RECORD-LENGTH  PIC S9(8)   COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier initialized by OPEN-VRAM.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

BUFFER — The buffer containing the change record to be written to StorHouse.

RECORD-LENGTH — An integer value containing the length of the change record written to StorHouse.

Return Codes

2612	An attempt was made to change a record without reading the record first.
Any Non-Zero Value	A record was not changed.

Detailed Function Description

CHANGE allows a user to change the last record read from the VRAM file that was previously opened with OPEN-VRAM. The VRAM file is identified by the O-TOKEN returned by OPEN-VRAM.

Note

To perform CHANGE, the MODE in the OPEN-VRAM call must be set to UPDATE. For a more complete discussion of MODE and the associated ACCESS-METHOD parameter, refer to the OPEN-VRAM function description.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

StorHouse Command Submission

There is one StorHouse command submission function: SM-CMD-INTF. SM-CMD-INTF allows an application to:

- Send selected StorHouse Command Language commands to StorHouse and to retrieve response text from those commands.
- Direct administrative operations from an application rather than from a user at a terminal through the Interactive Interface.

SM-CMD-INTF is described in the following section.

SM-CMD-INTF

SM-CMD-INTF, the StorHouse Command Interface function, sends a text string to StorHouse to be processed as a StorHouse command. SM-CMD-INTF requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

`CALL 'LSMCALL' USING SM-CMD-INTF,C-TOKEN,R-CODE,CR-BUF,`
`CR-LEN,RESP-BUF,RESP-BUFSIZE,`
`RESP-INFO.`

CICS Environment

`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,`
`SM-CMD-INTF,C-TOKEN,R-CODE,CR-BUF,`
`CR-LEN,RESP-BUF,RESP-BUFSIZE,`
`RESP-INFO.`

Working Storage Section for COBOL Program

```
01 SM-CMD-INTF          PIC X(16)  VALUE 'SM-CMD-INTF'.
01 C-TOKEN              PIC S9(8)  COMP SYNC.
01 R-CODE               PIC S9(8)  COMP SYNC.
01 CR-BUF               PIC X(buffer-size).
01 CR-LEN                PIC S9(8)  COMP SYNC.
01 RESP-BUF              PIC X(resp-bufsize).
01 RESP-BUFSIZE          PIC S9(8)  COMP SYNC VALUE IS nn2.
01 RESP-INFO.
   05 RINFO-LIST-SIZE    PIC S9(8)  COMP SYNC VALUE 6.
   05 RINFO-LENGTH       PIC S9(8)  COMP SYNC.
   05 RINFO-STATUS       PIC S9(8)  COMP SYNC.
   05 RINFO-SEVERITY     PIC S9(8)  COMP SYNC.
   05 RINFO-CMD-ENDED    PIC S9(8)  COMP SYNC.
   05 RINFO-PROMPT       PIC S9(8)  COMP SYNC.
   05 RINFO-SUPPRESS     PIC S9(8)  COMP SYNC.
```

Parameter Overview

C-TOKEN — The session identifier set by CONNECT.

R-CODE — Final status from the requested operation; see the following section "Return Codes."

CR-BUF — A buffer containing the command/reply text that is sent to StorHouse. The maximum length of this buffer is 255 bytes.

CR-LEN — An integer value containing the length of the text in the buffer named by CR-BUF.

RESP-BUFFER — The area where the response text from the StorHouse command is placed.

RESP-BUFSIZE — An integer value giving the size of the response buffer. This value should be no smaller than 132. If the response buffer is too small to contain the response text, then the response is truncated to fit in the supplied buffer, and the return status indicates an error.

RESP-BUFSIZE may not equal zero. A zero value causes a 3022 return code, indicating that a zero buffer size was passed to a StorHouse read function.

RESP-INFO — An array of 32-bit integers that provides detailed information about the length and type of response text returned by StorHouse. The elements in the array are:

- RINFO-LIST-SIZE – the number of other elements in the RESP-INFO array. The caller must set this entry to 6.
- RINFO-LENGTH – the length of the response text.
- RINFO-STATUS – the status code associated with execution of the command. This code is returned only when *command end* is indicated. Note that RINFO-STATUS refers to the StorHouse status, while R-CODE indicates host system problems.
- RINFO-SEVERITY – the severity of the error indicated in RINFO-STATUS, expressed as a value between 0 and 20; see "Return Codes."
- RINFO-CMD-ENDED – a flag indicating (if 1) that the command has completed execution.
- RINFO-PROMPT – a flag indicating (if 1) that the response text is actually a prompt from StorHouse.
- RINFO-SUPPRESS – a flag indicating (if 1) that StorHouse suggests suppression of printing or displaying the information supplied in response to a prompt. This flag is valid only if RINFO-PROMPT is set.

Return Codes

Any Non-Zero Value A command text was not processed by StorHouse.

If R-CODE is non-zero, do not use RINFO-STATUS and RINFO-SEVERITY.

Zero Value Indicates that the command was successfully passed to StorHouse and that a response was received.

RINFO-STATUS indicates the status associated with StorHouse's execution of the command. Use RINFO-SEVERITY to examine the general condition associated with command execution without testing for specific status codes.

The severity codes are:

- 00 – normal; no errors detected.
- 04 – warning; results may not be as expected.
- 08 – error; results are probably incorrect, and corrective action may be required.
- 12 – severe errors occurred; corrective action is required.
- 16 – request could not be processed.
- 20 – hardware or system software error prevented command processing. (Partial execution may have occurred, or StorHouse may have processed the command, but responses have been lost.)

Detailed Function Description

SM-CMD-INTF allows direct user access to the StorHouse command processing facilities by sending a text string in the CR-BUF to StorHouse. StorHouse then processes the string as a StorHouse command. The session is identified by the O-TOKEN returned by CONNECT. The command text is sent from the user-supplied buffer.

SM-CMD-INTF returns the command response to the user-supplied response buffer. RINFO-CMD-ENDED is set (non-zero) when no additional response information is available for this command. If RINFO-CMD-ENDED is not set (zero), SM-CMD-INTF must be called again to retrieve additional response text.

For a description of available commands, refer to the *Command Language Reference Manual*.

SM-CMD-INTF is only intended for accessing informational commands, such as SHOW FILE. The following commands cannot be issued from SM-CMD-INT:

- GET and PUT. A file-oriented OPEN must be used.

- SET USER to change session defaults.

- Any command that creates or deletes files (for example, REMOVE FILE, CREATE FILE, DELETE).
- Any command that changes passwords (for example, SET GROUP).

For more information about StorHouse Command Language commands that can be accessed with SM-CMD-INTF, consult your StorHouse system administrator or your FileTek customer support representative.

When RINFO-PROMPT is set, StorHouse is requesting additional information to complete execution of the submitted command. The response string in the RESP-BUFFER is a prompt that indicates the type of information to be provided. Whenever this occurs, SM-CMD-INTF must be called again with the reply to the prompt in CR-BUF.

Note

Failure to call SM-CMD-INTF repeatedly until end of command is indicated causes the session link to become unusable. Once SM-CMD-INTF has been called, other functions cannot be called until end of command has been returned. The only exceptions are CHECK, ECBADDR, EMSG, and ABORT. Whenever RINFO-PROMPT is set, the next call to SM-CMD-INTF supplies a response. If CR-LEN is zero, a null response is sent to StorHouse, which implies use of a default.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

General Usage Functions

The general usage functions are:

- CHECK – waits for and tests the completion status of any asynchronous operation.
- ECBADDR – returns the address of the ECB that is POSTed when the asynchronous function last requested for the C-TOKEN or O-TOKEN completes.
- EMSG – retrieves indicative messages associated with a previous return code.
- ABORT – attempts to terminate the last asynchronous function started within a session or the last asynchronous data transfer function. ABORT can also be used to request termination of an SM-CMD-INTF sequence.

These functions are described in the following sections.

CHECK

CHECK waits for and tests the completion status of any asynchronous operation. CHECK requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment
`CALL 'LSMCALL' USING CHECK,{C-TOKEN|O-TOKEN},R-CODE.`

CICS Environment
`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,` `CHECK,{C-TOKEN|O-TOKEN},R-CODE.`

Working Storage Section for COBOL Program

```
01 CHECK          PIC X(16    )VALUE 'CHECK'.
01 x-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
```

Parameter Overview

C-TOKEN The session identifier returned by CONNECT.

O-TOKEN The file identifier returned by OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

R-CODE Final status from the requested operation; see the following section "Return Codes."

Return Codes

The return code from CHECK is the code returned by the previous asynchronous function.

2974 No asynchronous operations were outstanding at the time of CHECK for the particular session or file.

Detailed Function Description

CHECK waits for and tests the completion status of a previous asynchronous function. If a C-TOKEN is supplied, the CHECK applies to the last function started on a session link. If an O-TOKEN is supplied, then the last asynchronous data transfer function associated with that O-TOKEN is checked.

Notes

- There is no asynchronous form of CHECK; that is, ASCHECK cannot be used.
- For applications that must WAIT for multiple, concurrent, asynchronous events, the address of the ECB can be obtained. (See ECBADDR in Chapter 6, "Sample Program.") CHECK can then be called after the application has determined that the original function has completed. The ECB complete bit must not be cleared prior to calling CHECK.
- Important: CHECK must always be called. It is not sufficient just to WAIT for the ECB to be POSTed.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, "Sample Program."

ECBADDR

ECBADDR returns to the caller the address of the ECB that is POSTed when the asynchronous function last requested for the C-TOKEN or O-TOKEN completes. ECBADDR requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

```
CALL 'LSMCALL' USING ECBADDR,{C-TOKEN|O-TOKEN},R-CODE,
                     RETURN-ECB-ADDR.
```

CICS Environment

```
CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,
                     ECBADDR,{C-TOKEN|O-TOKEN},R-CODE,
                     RETURN-ECB-ADDR.
```

Working Storage Section for COBOL Program

```
01 ECBADDR        PIC X(16)   VALUE 'ECBADDR'.
01 x-TOKEN        PIC S9(8)   COMP SYNC.
01 R-CODE         PIC S9(8)   COMP SYNC.
01 RETURN-ECB-ADDRPIC S9(8)   COMP SYNC.
```

Parameter Overview

C-TOKEN The session identifier returned by CONNECT.

O-TOKEN The file identifier returned by OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

R-CODE Final status from the requested operation; see the following section "Return Codes."

RETURN-ECB-ADDR The address of the ECB that is POSTed when the asynchronous function last requested for the C-TOKEN or O-TOKEN completes.

Return Codes

The return code from ECBADDR is always 0.

Detailed Function Description

For applications that must WAIT for multiple, concurrent, asynchronous events, the address of the ECB can be obtained by using ECBADDR. If a C-TOKEN is supplied, ECBADDR applies to the last function started on a session link. If an O-TOKEN is supplied, then the last asynchronous data transfer function associated with that O-TOKEN is checked.

Note

There is no asynchronous form of ECBADDR; that is, ASECBADDR cannot be used.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

EMSG

EMSG retrieves indicative messages associated with a previous return code. EMSG requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment

`CALL 'LSMCALL' USING EMSG,{C-TOKEN|O-TOKEN},R-CODE,`
`MESSAGE-BUFFER,MESSAGE-BUFFER-SIZE,`
`RETURNED-MESSAGE-LEN.`

CICS Environment

`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,`
`EMSG,{C-TOKEN|O-TOKEN},R-CODE,`
`MESSAGE-BUFFER,MESSAGE-BUFFER-SIZE,`
`RETURNED-MESSAGE-LEN.`

Working Storage Section for COBOL Program

```
01 EMSG                   PIC X(16)  VALUE 'EMSG'.
01 x-TOKEN                PIC S9(8)  COMP SYNC.
01 R-CODE                 PIC S9(8)  COMP SYNC.
01 MESSAGE-BUFFER         PIC X(buffer-size).
01 MESSAGE-BUFFER-SIZE    PIC S9(8)  COMP SYNC VALUE IS nnn.
01 RETURNED-MESSAGE-LEN   PIC S9(8)  COMP SYNC.
```

Parameter Overview

C-TOKEN The session identifier set by CONNECT.

O-TOKEN The file identifier returned by an OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

R-CODE Final status from the requested operation; see the following section "Return Codes."

MESSAGE-BUFFER The area into which the error message text will be moved.

MESSAGE-BUFFER-SIZE — A user-specified integer value containing the size of the MESSAGE-BUFFER.

RETURNED-MESSAGE-LEN — An integer value that is returned by EMSG containing the length of the retrieved error message.

Return Codes

3065 — There are no more messages in the message buffer for the previously called function.

Detailed Function Description

EMSG allows a user to retrieve indicative text messages associated with a previous function call. If a C-TOKEN is supplied, EMSG returns the error message associated with the return code from the last session-related command. If an O-TOKEN is supplied, EMSG returns the text messages associated with the last file-related command.

EMSG returns one message in the user-supplied buffer. The length of the message is also returned.

Notes

- There is no asynchronous form of EMSG; that is, ASEMSG cannot be used.
- The maximum buffer length required to retrieve an error message is 132 bytes. If the user-supplied buffer is shorter than 132 bytes, some messages may be truncated when returned. Messages are padded with blanks to the full size of the supplied buffer.
- If MESSAGE-FLAG was set for a CONNECT, OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM, then EMSG must be called after a DISCONNECT or CLOSE, or after a failing CONNECT. EMSG must be called repeatedly until a return code of 3065, indicating no more messages, is received. Otherwise, dynamically allocated memory used by the LSMCALL routine may not be released.
- The C-TOKEN is the correct token for an EMSG call following any type of open request.

Cross-Reference to Sample Program

Refer to the sample COBOL program in Chapter 6, “Sample Program”:

```
PARAGRAPH 1510-CALL-EMSG, 1610-CALL-EMSG
```

ABORT

ABORT attempts to terminate the last asynchronous function started within a session or the last asynchronous data transfer function. ABORT can also be used to request termination of an SM-CMD-INTF sequence. ABORT requires StorHouse standard features.

Statement Format for COBOL

TSO/Batch/IMS Environment
`CALL 'LSMCALL' USING ABORT,{O-TOKEN|C-TOKEN},R-CODE.`

CICS Environment
`CALL 'LSMCICS' USING DFHEIBLK,COMMAREA,` `ABORT,{O-TOKEN|C-TOKEN},R-CODE.`

Working Storage Section for COBOL Program

```
01 ABORT          PIC X(16)  VALUE 'ABORT'.
01 x-TOKEN        PIC S9(8)  COMP SYNC.
01 R-CODE         PIC S9(8)  COMP SYNC.
```

Parameter Overview

O-TOKEN — The file identifier returned by OPEN-SEQ, CREATE-OPEN, or OPEN-VRAM.

C-TOKEN — The session identifier returned by CONNECT.

R-CODE — Status from the abort requested; see "Return Codes."

Return Codes

2989 — Neither an asynchronous function nor an SM-CMD-INTF sequence was outstanding.

2990 — The function to be aborted was CONNECT, which is not allowed.

Zero is the only other return code from ABORT. CHECK must be issued to retrieve the return code associated with the aborted function.

Detailed Function Description

ABORT unconditionally attempts to terminate the last function started on a session or a data transfer function. If a C-TOKEN is supplied, the ABORT applies to the last function started on a session link. If an O-TOKEN is supplied, then the last data transfer function associated with that O-TOKEN is aborted.

Notes

- There is no asynchronous form of ABORT; that is, ASABORT cannot be used.
- ABORT can only request termination of a function. The function may have already completed or may complete before the abort request is forwarded. The return code associated with the original function must be analyzed to determine the actual outcome of the abort attempt.
- ABORT is intended as a mechanism to terminate a pending asynchronous operation for a data transfer or session, when that transfer or session is to be subsequently terminated.
- For some operations, such as a sequential read or write, ABORT causes the entire data transfer to fail.
- When ABORT is issued during an SM-CMD-INTF sequence, termination of processing of the command by StorHouse is requested. However, the user must continue calling SM-CMD-INTF until command-ended is indicated.

Cross-Reference to Sample Program

There is no cross-reference to the sample COBOL program contained in Chapter 6, “Sample Program.”

Chapter

6

Sample Program

The sample program in this chapter illustrates the use of the standard IBM StorHouse host Callable Interface. The sample program performs the following functions:

- Establishes a session with the StorHouse.
- Transfers a sequential file from the host to a non-VRAM file on StorHouse.
- Reads and prints all records in the non-VRAM file.
- Transfers the same host file to a VRAM file on StorHouse.
- Reads the VRAM file by relative record number.
- Reads the VRAM file by key.
- Ends a session with StorHouse.

If any errors occur, the program prints an error message (using EMSG), closes the host files, and terminates.

Note This document reflects the name change from Storage Machine to StorHouse. The code will be updated in a later release.

COBOL Sample Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPCOB.

*******************************************************************************
*                                                                             *
*   THIS IS THE COBOL SAMPLE PROGRAM FOR FILETEK'S CALLABLE                   *
*   INTERFACE.  THIS PROGRAM DOES THE FOLLOWING:                              *
*                                                                             *
*   1. CONNECTS TO STORHOUSE.                                                 *
*   2. TRANSFERS A HOST SEQUENTIAL FILE TO A NON-VRAM STORHOUSE FILE.         *
*   3. READS AND PRINTS ALL RECORDS FROM THE NON-VRAM STORHOUSE FILE.         *
*   4. TRANSFERS THE SAME HOST FILE TO A VRAM STORHOUSE FILE.                 *
*   5. PROCESSES THE VRAM FILE BY RELATIVE RECORD NUMBER.                     *
*   6. PROCESSES THE VRAM FILE BY KEY.                                        *
*   7. DISCONNECTS FROM STORHOUSE.                                            *
*                                                                             *
*******************************************************************************
   ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.

      SELECT  HOST-FILE ASSIGN TO HOSTFILE
              ORGANIZATION IS SEQUENTIAL
              ACCESS IS SEQUENTIAL.

      SELECT  PRINT-FILE ASSIGN TO SYSPRINT
              ORGANIZATION IS SEQUENTIAL
              ACCESS IS SEQUENTIAL.

   DATA DIVISION.
   FILE SECTION.
*
*   THE HOST SEQUENTIAL FILE CONTAINS 80-BYTE RECORDS. A SOCIAL SECURITY
*   NUMBER IS IN POSITIONS 1 THRU 9.
*
      FD HOST-FILE
         LABEL RECORDS STANDARD
         BLOCK CONTAINS 0 RECORDS
         RECORDING MODE IS F
         DATA RECORD IS HOST-RECORD.
      01 HOST-RECORD.
         05 HOST-RECORD-SSN            PIC 9(9).
         05 HOST-RECORD-DATA           PIC X(71).

*
*   PRINT-FILE IS USED TO PRINT ERROR MESSAGES AND DATA RECORDS.
*

      FD PRINT-FILE
         LABEL RECORDS STANDARD
         BLOCK CONTAINS 0 RECORDS
         RECORDING MODE IS F
         DATA RECORD IS PRINT-RECORD.
      01 PRINT-RECORD.
```

```
          05 PRINT-RECORD-CC            PIC X.
          05 PRINT-RECORD-DATA          PIC X(132).

      WORKING-STORAGE SECTION.

*
*     DEFINE THE STORHOUSE FUNCTION CHARACTER STRING IDENTIFIERS
*

       01   SM1-FUNCTION-CONNECT        PIC X(16) VALUE 'CONNECT'.
       01   SM1-FUNCTION-DISCONNECT     PIC X(16) VALUE 'DISCONNECT'.
       01   SM1-FUNCTION-OPEN-SEQ       PIC X(16) VALUE 'OPEN-SEQ'.
       01   SM1-FUNCTION-OPEN-VRAM      PIC X(16) VALUE 'OPEN-VRAM'.
       01   SM1-FUNCTION-CLOSE          PIC X(16) VALUE 'CLOSE'.
       01   SM1-FUNCTION-EMSG           PIC X(16) VALUE 'EMSG'.
       01   SM1-FUNCTION-READ           PIC X(16) VALUE 'READ'.
       01   SM1-FUNCTION-READ-RECORD    PIC X(16) VALUE 'READ-RECORD'.
       01   SM1-FUNCTION-READ-NEXT-KEY  PIC X(16) VALUE 'READ-NEXT-KEY'.
       01   SM1-FUNCTION-WRITE          PIC X(16) VALUE 'WRITE'.

*
*     DEFINE THE STORHOUSE COMMAND AND TRANSFER LINK TOKEN IDENTIFIERS
*

       01   SM1-CONNECT-TOKEN           PIC S9(8) SYNC COMP.
       01   SM1-OPEN-TOKEN              PIC S9(8) SYNC COMP.

*
*     DEFINE THE STORHOUSE RETURN CODE AREAS
*

       01   SM1-RETURN-CODE             PIC S9(8) SYNC COMP.
       01   SM1-EMSG-RETURN-CODE        PIC S9(8) SYNC COMP.

*
*     DEFINE THE PARAMETERS SPECIFIC TO THE CONNECT FUNCTION
*

       01   SM1-CMDLINK-EMSGFLAG        PIC S9(8) SYNC COMP VALUE +1.
       01   SM1-ACCOUNT                 PIC X(12) VALUE 'YOUR_ACCT'.
       01   SM1-PASSWORD                PIC X(32) VALUE 'YOUR_PSWD'.
       01   SM1-LIBRARY-ID              PIC X(6) VALUE SPACES.
       01   SM1-SUBSYSTEM-ID            PIC X(4) VALUE SPACES.

*
*     THE DISCONNECT FUNCTION REQUIRES ONLY THE FUNCTION IDENTIFIER,
*     CONNECT-TOKEN, AND RETURN-CODE PARAMETERS.

*
*
*     DEFINE THE PARAMETERS SPECIFIC TO THE CLOSE FUNCTION
*
       01   SM1-CLOSE-ABORT-FLAG        PIC S9(8) SYNC COMP VALUE +0.

*
*     DEFINE THE PARAMETERS SPECIFIC TO THE EMESSAGE FUNCTION
*
```

```
        01   SM1-EMSG-BUFFER            PIC X(132) VALUE SPACES'.
        01   SM1-EMSG-BUFFER-LENGTH     PIC S9(8) COMP SYNC VALUE +132.
        01   SM1-EMSG-MSG-LENGTH        PIC S9(8) COMP SYNC VALUE +0.

*
*      THE PARAMETERS FOR OPEN-SEQ AND OPEN-VRAM ARE NOW DEFINED
*      IN THE FOLLOWING MANNER:
*         1)DEFINE THOSE PARAMETERS COMMON TO BOTH OPEN FUNCTIONS
*         2)DEFINE THOSE PARAMETERS SPECIFIC TO EACH FUNCTION
*
*


*      COMMON PARAMETERS USED BY BOTH OPEN-SEQ AND OPEN-VRAM
*
        01   SM1-XFRLINK-EMSGFLAG       PIC S9(8) COMP SYNC VALUE +1.
        01   SM1-FILE-MODE              PIC X(6) VALUE SPACES.
        01   SM1-FILE-PASSWORDS.
          05 F-READ-PWD                PIC X(8) VALUE SPACES.
          05 F-WRITE-PWD               PIC X(8) VALUE SPACES.
*      NOTE: THE DELETE PASSWORD IS NOT USED BY OPEN-VRAM
          05 F-DELETE-PWD              PIC X(8) VALUE SPACES.
        01   SM1-GROUP-NAME             PIC X(8) VALUE SPACES.
        01   SM1-GROUP-PASSWORDS.
          05 G-READ-PWD                PIC X(8) VALUE SPACES.
          05 G-WRITE-PWD               PIC X(8) VALUE SPACES.
*      NOTE: THE DELETE PASSWORD IS NOT USED BY OPEN-VRAM
          05 G-DELETE-PWD              PIC X(8) VALUE SPACES.

*
*      DEFINE THE FILE MODE CHARACTER STRINGS USED FOR OPENS.
*


        01   SM1-MODE-APPEND            PIC X(6)  VALUE 'APPEND'.
        01   SM1-MODE-READ              PIC X(6)  VALUE 'READ'.
        01   SM1-MODE-WRITE             PIC X(6)  VALUE 'WRITE'.

*
*      PARAMETERS SPECIFIC TO OPEN-SEQ
*

        01   SM1-FILE-VERSION           PIC S9(8) COMP SYNC VALUE +0.

*
*      THE STORHOUSE FILENAME IS SPECIFIED ON THE "//SM1NVFIL" DD STATEMENT IN
       THE JCL
*
        01   SM1-NONVRAM-FILENAME       PIC X(56) VALUE 'DD=SM1NVFIL'.

*
*      SPECIFY BLANKS FOR THE FILE LOCATION (VOLUME AND FILE SETS)
*      TO USE THE DEFAULTS

        01   SM1-FILE-LOCATION.
          05 SM1-VOLUME-SET             PIC X(8)  VALUE SPACES.
          05 SM1-FILE-SET               PIC X(8)  VALUE SPACES.
```

```
*
*      DEFINE THE FILE ATTRIBUTE LIST USED BY THE OPEN-SEQ FUNCTION
*

       01   SM1-NONVRAM-FILE-ATTRIBUTES.

*      THERE ARE 8 ITEMS IN THIS LIST
          05 FATTR-LIST-SIZE            PIC S9(8) COMP SYNC VALUE +8.

*      THE FILE SIZE MUST BE SPECIFIED AND IS CALCULATED AS FOLLOWS:
*         100 80-BYTE RECORDS IN THE HOST FILE = 8000 BYTES
*         A 5% OVERHEAD IS ADDED, THUS FILESIZE = 8400

          05 FATTR-FILE-SIZE            PIC S9(8) COMP SYNC VALUE +8400.

*      THE RECORD LENGTH IS A MAXIMUM OF 80 BYTES
          05 FATTR-MAX-RECORD-LEN       PIC S9(8) COMP SYNC VALUE +80.

*      THE STORHOUSE FILE IS TO BE TRANSPORTABLE ACROSS HOST SYSTEMS
          05 FATTR-TRANSPORT-FL         PIC S9(8) COMP SYNC VALUE +1.

*      THE FILE IS TO BE STORED IN STORHOUSE IN ASCII FORMAT
          05 FATTR-DATA-XLATE-FL        PIC S9(8) COMP SYNC VALUE +1.

*      THE FILE IS FIXED RECORD FORMAT
          05 FATTR-FIXED-RECORD-FL      PIC S9(8) COMP SYNC VALUE +1.

*      THE FILE DOES NOT CONTAIN ANSI CONTROL CHARACTERS
          05 FATTR-CC-ANSI-FL           PIC S9(8) COMP SYNC VALUE +0.

*      THE FILE DOES NOT CONTAIN MACHINE CONTROL CHARACTERS
          05 FATTR-CC-MACH-FL           PIC S9(8) COMP SYNC VALUE +0.

*      A CROSS MEMORY BUFFER SIZE OF 64K IS TO BE USED
          05 FATTR-BLOCK-SIZE           PIC S9(8) COMP SYNC VALUE +65536.

*
*      DEFINE THE FILE OPTIONS LIST USED BY THE OPEN-SEQ FUNCTION
*

       01   SM1-NONVRAM-FILE-OPTIONS.

*      THERE ARE 8 ITEMS IN THIS LIST
          05 FOPTS-LIST-SIZE            PIC S9(8) COMP SYNC VALUE +8.

*      LOCK THE FILE ON WRITE
          05 FOPTS-LOCK                 PIC S9(8) COMP SYNC VALUE +1.

*      IF THE FILE IS LOCKED WHEN THIS PROGRAM ATTEMPTS ACCESS,
*      DO NOT WAIT FOR THE LOCK TO BE RELEASED.  INSTEAD,
*      TERMINATE PROCESSING.
          05 FOPTS-WAIT                 PIC S9(8) COMP SYNC VALUE +0.

*      SPECIFY /ATF=2
          05 FOPTS-ATF                  PIC S9(8) COMP SYNC VALUE +2.
```

```
*      SPECIFY THE DEFAULT EDC
          05 FOPTS-EDC                  PIC S9(8) COMP SYNC VALUE +0.

*      SPECIFY THE MAXIMUM VERSION LIMIT VALUE
          05 FOPTS-LIMIT                PIC S9(8) COMP SYNC VALUE +128.

*      CREATE A NEW VERSION OF THE FILE BY SPECIFYING THAT THIS
*      IS NOT A NEW FILE
          05 FOPTS-NEW                  PIC S9(8) COMP SYNC VALUE +0.

*      UNLOCK THE FILE AFTER WRITING
          05 FOPTS-UNLOCK               PIC S9(8) COMP SYNC VALUE +1.

*      SPECIFY /VTF=NEXT
          05 FOPTS-VTF                  PIC S9(8) COMP SYNC VALUE +2.

*
*      PARAMETERS SPECIFIC TO THE OPEN-VRAM FUNCTION
*

       01    SM1-FILE-REVISION          PIC S9(8) COMP SYNC VALUE +0.

*      THE STORHOUSE FILENAME IS SPECIFIED ON THE "//SM1VFIL" DD
*      STATEMENT IN THE JCL

       01    SM1-VRAM-FILENAME          PIC X(56) VALUE 'DD=SM1VFIL'.

*      THE VRAM FILE WILL BE PROCESSED BY RECORD NUMBER AND KEY

       01    SM1-OPEN-METHOD            PIC X(24) VALUE 'RECORD,KEYED'.

*
*      DEFINE THE VRAM FILE ATTRIBUTE LIST
*

       01    SM1-VRAM-FILE-ATTRIBUTES.

*      THERE ARE 8 ITEMS IN A FULL-LENGTH ATTRIBUTE LIST.  THIS PROGRAM
*      DOES NOT USE THE VERSION OR THE CHECKPOINT FIELDS, SO THEY ARE
*      OMITTED.

          05 LIST-SIZE                  PIC S9(8) COMP SYNC VALUE +6.

*      THE MAXIMUM RECORD LENGTH IS 80
          05 MAX-RCD-LENGTH             PIC S9(8) COMP SYNC VALUE +80.

*      THE LAST PHYSICAL RECORD NUMBER IS RETURNED HERE BY OPEN
          05 LAST-P-REC-NUM             PIC S9(8) COMP SYNC VALUE +0.

*      THE LAST LOGICAL RECORD NUMBER IS RETURNED HERE BY OPEN
          05 LAST-L-REC-NUM             PIC S9(8) COMP SYNC VALUE +0.

*      THE CURRENT REVISION OF THE FILE
          05 FILE-REVISION-NUMBER       PIC S9(8) COMP SYNC VALUE +0.

*      THE FILE TYPE IS RETURNED HERE
          05 FILE-TYPE                  PIC S9(8) COMP SYNC VALUE +0.
```

```
*      A CROSS MEMORY BUFFER SIZE OF 64K IS TO BE USED
          05 BLOCK-SIZE                 PIC S9(8) COMP SYNC VALUE +65536.

*
*      THE PARAMETERS FOR THE READS AND WRITES ARE NOW DEFINED
*      IN THE FOLLOWING MANNER:
*         1)DEFINE THOSE PARAMETERS COMMON TO ALL FUNCTIONS
*         2)DEFINE THOSE PARAMETERS SPECIFIC TO EACH FUNCTION
*
*
*      PARAMETERS USED FOR ALL READ AND WRITE OPERATIONS
*

*      ALL DATA IS READ INTO OR WRITTEN FROM THIS AREA
        01   SM1-FILE-BUFFER.
          05 SM1-FILE-KEY               PIC 9(9).
          05 SM1-FILE-DATA              PIC X(71).

*      THE BUFFER IS 80 BYTES IN LENGTH
        01   SM1-FILE-BUFFER-LENGTH     PIC S9(8) COMP SYNC VALUE +80.

*      THE RECORD LENGTH (SPECIFIED FOR WRITE, RETURNED BY READ)
        01   SM1-RECORD-LENGTH          PIC S9(8) COMP SYNC VALUE +80.

*      THE RECORD NUMBER JUST WRITTEN (RETURNED BY WRITE)
        01   SM1-RECORD-NUMBER          PIC S9(8) COMP SYNC.

*
*      PARAMETERS SPECIFIC TO THE VRAM READ FUNCTIONS
*

        01   SM1-RECORD-KEY             PIC 9(9).
        01   SM1-KEY-NAME               PIC X(56) VALUE 'SOCIAL-SECURITY-NUMBER'.
        01   SM1-KEY-LENGTH             PIC S9(8) COMP SYNC VALUE +9.
        01   SM1-RELATIVE-RECORD        PIC S9(8) COMP SYNC VALUE +0.

*
*      MISCELLANEOUS STORAGE AREAS
*

       01    COMP-WORK-AREAS            COMP SYNC.
          05 HOST-FILE-EOF              PIC S9(8).
          05 I                          PIC S9(8)   VALUE +0.
          05 K                          PIC S9(8)   VALUE +0.
       01    KEY-TABLE                  PIC 9(9)    OCCURS 4.
       01    PRINT-MESSAGES.
          05 ERROR-RETURNED-FROM-SM.
             10 FILLER                  PIC X(13) VALUE 'SM1 FUNCTION'.
             10 ERROR-SM-FUNCTION       PIC X(16).
             10 FILLER                  PIC X(19) VALUE ' FAILED...RETURN ='.
             10 ERROR-SM-RETURN         PIC Z(4).


    PROCEDURE DIVISION.

    INITIAL-PROCESSING.
```

```
      OPEN OUTPUT PRINT-FILE.
      MOVE SPACES TO PRINT-RECORD.

   MAIN-PROCESSING.

*  CONNECT TO STORHOUSE.

      PERFORM 100-CONNECT-TO-SM THRU 100-EXIT.

*  TRANSFER THE HOST FILE TO A NONVRAM STORHOUSE FILE.

      PERFORM 200-TRANSFER-NOVRAM-FILE THRU 200-EXIT.

*  RETRIEVE/PRINT ALL RECORDS FROM THE NEWLY CREATED STORHOUSE FILE.

      PERFORM 300-RETRIEVE-NOVRAM-FILE THRU 300-EXIT.

*  TRANSFER THE HOST FILE TO A VRAM STORHOUSE FILE.

      PERFORM  400-TRANSFER-VRAM-FILE THRU 400-EXIT.

*  READ AND PRINT RELATIVE RECORDS 2, 4, 6, AND 8.

      PERFORM  500-PROCESS-RELATIVE-RECORDS THRU 500-EXIT.

*  READ-KEYED AND PRINT RECORDS 2, 4, 6, AND 8;
*  READ-NEXT-KEY AND PRINT RECORDS 3, 5, 7, AND 9.

      PERFORM  600-PROCESS-KEYED-RECORDS THRU 600-EXIT.

*  DISCONNECT FROM STORHOUSE.

      PERFORM 700-DISCONNECT THRU 700-EXIT.

   FINAL-PROCESSING.

      CLOSE PRINT-FILE.
      GOBACK.

   100-CONNECT-TO-SM.
*
*  CONNECT TO STORHOUSE.
*

      CALL 'LSMCALL' USING SM1-FUNCTION-CONNECT
                           SM1-CONNECT-TOKEN
                           SM1-RETURN-CODE
                           SM1-CMDLINK-EMSGFLAG
                           SM1-ACCOUNT
                           SM1-PASSWORD
                           SM1-LIBRARY-ID
                           SM1-SUBSYSTEM-ID.

*
*  USE THE EMESSAGE FUNCTION TO RETRIEVE ANY STORHOUSE MESSAGES.
*
```

```
   PERFORM 1500-RETRIEVE-CMDLINK-EMSGS THRU 1500-EXIT.
*
*  INSURE A SUCCESSFUL CONNECT TO STORHOUSE. TERMINATE PROCESSING
*  IF THE CONNECT FAILED.
*
      IF SM1-RETURN-CODE EQUAL ZERO  NEXT SENTENCE
         ELSE
            MOVE SM1-FUNCTION-CONNECT TO ERROR-SM-FUNCTION
            PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
            PERFORM FINAL-PROCESSING.

   100-EXIT.
      EXIT.


   200-TRANSFER-NONVRAM-FILE.

*  THIS ROUTINE TRANSFERS A HOST FILE TO STORHOUSE.  THE STORHOUSE FILE
*  IS CREATED IN TRANSPORTABLE, ASCII FORMAT.

*  OPEN THE HOST FILE

      OPEN INPUT HOST-FILE.

*  SET THE STORHOUSE FILE OPTIONS TO LOCK AT OPEN, UNLOCK AT CLOSE.

      MOVE +1 TO  FOPTS-LOCK  FOPTS-UNLOCK.

*  SET THE STORHOUSE FILE MODE TO WRITE.

      MOVE SM1-MODE-WRITE TO SM1-FILE-MODE.

*  OPEN THE STORHOUSE FILE.  IF THE OPEN FAILS, ALL PROCESSING IS
*  TERMINATED BY THE 1000-OPEN-NONVRAM ROUTINE.

      PERFORM 1000-OPEN-NONVRAM THRU 1000-EXIT.

*  READ THE HOST FILE, AND WRITE TO STORHOUSE

      MOVE +0 TO HOST-FILE-EOF.

      PERFORM 210-READ-WRITE THRU 210-EXIT
         UNTIL HOST-FILE-EOF NOT EQUAL ZERO.

*  CLOSE THE HOST FILE AND THE STORHOUSE FILE.

      CLOSE HOST-FILE

      PERFORM 1300-CLOSE-SM-FILE THRU 1300-EXIT.


*  TRANSFER IS COMPLETE.

   200-EXIT.
      EXIT.
```

```
   210-READ-WRITE.

*  READ A RECORD FROM THE HOST FILE.

      READ HOST-FILE INTO SM1-FILE-BUFFER
         AT END MOVE +1 TO HOST-FILE-EOF.

*  WRITE THE RECORD TO STORHOUSE.  IF THE WRITE FAILS, ALL PROCESSING
*  IS TERMINATED BY THE 1200-WRITE-TO-SM ROUTINE.

      IF  HOST-FILE-EOF EQUAL ZERO
         PERFORM 1200-WRITE-TO-SM THRU 1200-EXIT.

   210-EXIT.
      EXIT.


   300-RETRIEVE-NONVRAM-FILE.

*  THIS ROUTINE RETRIEVES AND PRINTS EACH RECORD FROM STORHOUSE
*  NONVRAM FILE JUST CREATED.

*  SET THE STORHOUSE FILE OPTIONS FOR NO LOCK.

      MOVE +0 TO FOPTS-LOCK.

*  SET THE STORHOUSE FILE MODE TO READ.

      MOVE SM1-MODE-READ TO SM1-FILE-MODE.

*  OPEN THE STORHOUSE FILE.  IF THE OPEN FAILS, ALL PROCESSING IS
*  TERMINATED BY THE 1000-OPEN-NONVRAM ROUTINE.

      PERFORM 1000-OPEN-NONVRAM THRU 1000-EXIT.

*  READ THE STORHOUSE FILE UNTIL END OF FILE.

      PERFORM 310-READ-SM THRU 310-EXIT
         UNTIL SM1-RETURN-CODE NOT EQUAL ZERO.

*  CLOSE THE STORHOUSE FILE.

      PERFORM 1300-CLOSE-SM-FILE THRU 1300-EXIT.

*  RETRIEVAL IS COMPLETE.

   300-EXIT.
      EXIT.

   310-READ-SM.

*  READ A RECORD FROM STORHOUSE AND PRINT THE RECORD.

      CALL 'LSMCALL' USING SM1-FUNCTION-READ
                           SM1-OPEN-TOKEN
                           SM1-RETURN-CODE
                           SM1-FILE-BUFFER
```

```
                              SM1-FILE-BUFFER-LENGTH
                              SM1-RECORD-LENGTH.

*   SM1-RETURN-CODE = 5650, THEN END-OF-FILE WAS ENCOUNTERED.
*   ANY OTHER NON-ZERO RETURN IS AN ERROR CONDITION.

       IF SM1-RETURN-CODE NOT EQUAL ZERO
          IF SM1-RETURN-CODE NOT EQUAL +5650
             MOVE SM1-FUNCTION-READ TO ERROR-SM-FUNCTION
             PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
             PERFORM 1600-RETRIEVE-XFRLINK-EMSGS THRU 1600-EXIT
          ELSE
             NEXT SENTENCE
          ELSE
             PERFORM 2200-PRINT-SM-RECORD THRU 2200-EXIT.

   310-EXIT.
      EXIT.

   400-TRANSFER-VRAM-FILE.

*   THIS ROUTINE TRANSFERS A HOST FILE TO A STORHOUSE VRAM FILE.

*   OPEN THE HOST FILE

       OPEN INPUT HOST-FILE.

*   SET THE STORHOUSE FILE MODE TO APPEND.

       MOVE SM1-MODE-APPEND TO SM1-FILE-MODE.

*   OPEN THE STORHOUSE FILE.  IF THE OPEN FAILS, ALL PROCESSING IS
*   TERMINATED BY THE 1100-OPEN-VRAM ROUTINE.

       PERFORM 1100-OPEN-VRAM THRU 1100-EXIT.

*   READ THE HOST FILE, AND WRITE TO STORHOUSE

       MOVE +0 TO HOST-FILE-EOF.

       PERFORM 410-READ-WRITE THRU 410-EXIT
          UNTIL HOST-FILE-EOF NOT EQUAL ZERO.

*   CLOSE THE HOST FILE AND THE STORHOUSE FILE.

       CLOSE HOST-FILE.

       PERFORM 1300-CLOSE-SM-FILE THRU 1300-EXIT.


*   TRANSFER IS COMPLETE.

   400-EXIT.
      EXIT

   410-READ-WRITE.
```

```
*   READ A RECORD FROM THE HOST FILE.

       READ HOST-FILE INTO SM1-FILE-BUFFER
          AT END MOVE +1 TO HOST-FILE-EOF.

*   WRITE THE RECORD TO STORHOUSE.  IF THE WRITE FAILS, ALL PROCESSING
*   IS TERMINATED BY THE 1200-WRITE-TO-SM ROUTINE.

       IF HOST-FILE-EOF EQUAL ZERO
          PERFORM 1200-WRITE-TO-SM THRU 1200-EXIT.

   410-EXIT.
       EXIT.

   500-PROCESS-RELATIVE-RECORDS.
*
*      THIS ROUTINE WILL:
*      OPEN THE STORHOUSE VRAM FILE JUST CREATED FOR READ PROCESSING
*      READ BY RELATIVE RECORD NUMBER AND PRINT RECORDS 2, 4, 6, 8
*      SAVE THE KEYS FOR THESE RECORDS FOR SUBSEQUENT READ-KEYED
*      PROCESSING.

*      OPEN THE VRAM FILE FOR READ PROCESSING.

       MOVE SM-MODE-READ TO SM1-FILE-MODE.

       PERFORM 1100-OPEN-VRAM THRU 1100-EXIT.

*      READ AND PRINT RECORDS 2, 4, 6, AND 8.  ALSO, SAVE THEIR KEYS.

       PERFORM 510-READ-PRINT THRU 510-EXIT
          VARYING SM1-RELATIVE-RECORD FROM +2 BY +2
             UNTIL SM1-RELATIVE-RECORD > +8.

*      RELATIVE RECORD PROCESSING COMPLETE.

   500-EXIT.
       EXIT.


   510-READ-AND-PRINT.

       CALL 'LSMCALL' USING SM1-FUNCTION-READ-RECORD
                            SM1-OPEN-TOKEN
                            SM1-RETURN-CODE
                            SM1-FILE-BUFFER
                            SM1-FILE-BUFFER-LENGTH
                            SM1-RECORD-LENGTH
                            SM1-RELATIVE-RECORD.

*   IF THE RETURN CODE FROM THE READ-RECORD IS NOT 0,
*   FORMAT AN ERROR MESSAGE, AND RETRIEVE AND PRINT ANY
*   STORHOUSE PROVIDED EMESSAGES.

       IF SM1-RETURN-CODE NOT EQUAL ZERO
          MOVE SM1-FUNCTION-READ-RECORD TO ERROR-SM-FUNCTION
          PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
```

```
         PERFORM 1600-RETRIEVE-XFRLINK-EMSGS THRU 1600-EXIT

*  OTHERWISE, PRINT THE RECORD JUST READ AND SAVE ITS KEY.

         ELSE
            PERFORM 2200-PRINT-SM-RECORD THRU 2200-EXIT
            ADD +1 TO I
            MOVE SM1-FILE-KEY TO KEY-TABLE(I).

   510-EXIT.
      EXIT.

   600-PROCESS-KEYED-RECORDS.
*
*  THIS ROUTINE WILL:
*     READ THE STORHOUSE VRAM FILE BY KEY, USING THE KEYS SAVED BY THE PREVIOUS
*     RELATIVE RECORD PROCESSING, READ-NEXT-KEY FOR THE INTERVENING RECORDS,
*     PRINT ALL RECORDS READ, AND CLOSE THE VRAM FILE.
*
      PERFORM 610-READ-KEYED THRU 610-EXIT
         VARYING K FROM +1 BY +1 UNTIL K > I.


*  CLOSE THE VRAM FILE.
      PERFORM 1300-CLOSE-SM-FILE THRU 1300-EXIT.

*  KEYED PROCESSING COMPLETE.

   600-EXIT.
      EXIT.

   610-READ-KEYED.
*
*  READ THE STORHOUSE VRAM FILE BY KEY.  KEYS WERE STORED IN KEY-TABLE.
*
      MOVE KEY-TABLE(I) TO SM1-RECORD-KEY.

      CALL 'LSMCALL' USING SM1-FUNCTION-READ-KEYED
                           SM1-OPEN-TOKEN
                           SM1-RETURN-CODE
                           SM1-FILE-BUFFER
                           SM1-FILE-BUFFER-LENGTH
                           SM1-RECORD-LENGTH
                           SM1-KEY-NAME
                           SM1-RECORD-KEY
                           SM1-KEY-LENGTH
                           SM1-RELATIVE-RECORD.

*  IF THE RETURN CODE FROM THE READ-KEYED IS NOT 0, FORMAT AN ERROR MESSAGE
*  AND RETRIEVE AND PRINT ANY STORHOUSE PROVIDED EMESSAGES.
*
      IF SM1-RETURN-CODE NOT EQUAL ZERO
         MOVE SM1-FUNCTION-READ-KEYED TO ERROR-SM-FUNCTION
         PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
         PERFORM 1600-RETRIEVE-XFRLINK-EMSGS THRU 1600-EXIT

*  IF THE RETURN CODE FROM THE READ-KEYED WAS 0,
```

```
*   PRINT THE RECORD JUST READ

       ELSE

          PERFORM 2200-PRINT-SM-RECORD THRU 2200-EXIT.

*   NOW, EXECUTE A READ-NEXT-KEY FUNCTION

       CALL 'LSMCALL' USING SM1-FUNCTION-READ-NEXT-KEY
                            SM1-OPEN-TOKEN
                            SM1-RETURN-CODE
                            SM1-FILE-BUFFER
                            SM1-FILE-BUFFER-LENGTH
                            SM1-RECORD-LENGTH
                            SM1-RELATIVE-RECORD.

*   IF THE RETURN CODE FROM THE READ-NEXT-KEY IS NOT 0, FORMAT AN ERROR
*   MESSAGE, AND RETRIEVE AND PRINT ANY STORHOUSE PROVIDED EMESSAGES.
*
       IF SM1-RETURN-CODE NOT EQUAL ZERO
          MOVE SM1-FUNCTION-READ-NEXT-KEY TO ERROR-SM-FUNCTION
          PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
          PERFORM 1600-RETRIEVE-XFRLINK-EMSGS THRU 1600-EXIT

*   OTHERWISE, PRINT THE RECORD JUST READ.

       ELSE
          PERFORM 2200-PRINT-SM-RECORD THRU 2200-EXIT.

   610-EXIT.
       EXIT.


   700-DISCONNECT.
*
*   DISCONNECT FROM STORHOUSE.
*
*    NOTE:
*
*   SINCE THE CONNECT FUNCTION SET THE EMSG FLAG TO 1, THIS ROUTINE WILL
*   CALL THE EMSG FUNCTION TO ALLOW THE STORHOUSE INTERFACE MODULES TO PERFORM
*   STORAGE CLEANUP.
*


       CALL 'LSMCALL' USING SM1-FUNCTION-DISCONNECT
                            SM1-CONNECT-TOKEN
                            SM1-RETURN-CODE.

       PERFORM 1500-RETRIEVE-CMDLINK-EMSGS THRU 1500-EXIT.


700-EXIT.
   EXIT.

   1000-OPEN-NONVRAM.
*
```

```
*   OPEN THE STORHOUSE NONVRAM FILE.
*
       CALL 'LSMCALL' USING SM1-FUNCTION-OPEN-SEQ
                            SM1-CONNECT-TOKEN
                            SM1-RETURN-CODE
                            SM1-XFRLINK-EMSGFLAG
                            SM1-OPEN-TOKEN
                            SM1-FILE-MODE
                            SM1-NONVRAM-FILENAME
                            SM1-FILE-VERSION
                            SM1-FILE-PASSWORDS
                            SM1-GROUP-NAME
                            SM1-GROUP-PASSWORDS
                            SM1-FILE-LOCATION
                            SM1-NONVRAM-FILE-ATTRIBUTES
                            SM1-NONVRAM-FILE-OPTIONS.
*
*   CHECK FOR A SUCCESSFUL OPEN.  IF THE OPEN FAILED, DISCONNECT
*   FROM STORHOUSE AND TERMINATE PROCESSING.
*
*   NOTE: DISCONNECT PROCESSING WILL RETRIEVE ANY STORHOUSE EMESSAGES.
*
       IF SM1-RETURN-CODE EQUAL ZERO NEXT SENTENCE
          ELSE
             MOVE SM1-FUNCTION-OPEN-SEQ TO ERROR-SM-FUNCTION
             PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
             PERFORM 700-DISCONNECT THRU 700-EXIT
             PERFORM FINAL-PROCESSING.


   1000-EXIT.
       EXIT.


   1100-OPEN-VRAM.
*
*   OPEN THE STORHOUSE VRAM FILE.
*
       CALL 'LSMCALL' USING SM1-FUNCTION-OPEN-VRAM
                            SM1-CONNECT-TOKEN
                            SM1-RETURN-CODE
                            SM1-XFRLINK-EMSGFLAG
                            SM1-OPEN-TOKEN
                            SM1-FILE-MODE
                            SM1-OPEN-METHOD
                            SM1-VRAM-FILENAME
                            SM1-FILE-REVISION
                            SM1-FILE-PASSWORDS
                            SM1-GROUP-NAME
                            SM1-GROUP-PASSWORDS
                            SM1-RELATIVE-RECORD
                            SM1-VRAM-FILE-ATTRIBUTES.

*
*   CHECK FOR A SUCCESSFUL OPEN.  IF THE OPEN FAILED, DISCONNECT
*   FROM STORHOUSE AND TERMINATE PROCESSING.
*
```

```
*   NOTE:  DISCONNECT PROCESSING WILL RETRIEVE ANY STORHOUSE EMESSAGES.
*
       IF SM1-RETURN-CODE EQUAL ZERO NEXT SENTENCE
          ELSE
             MOVE SM1-FUNCTION-OPEN-VRAM TO ERROR-SM-FUNCTION
             PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
             PERFORM 700-DISCONNECT THRU 700-EXIT
             PERFORM FINAL-PROCESSING.


    1100-EXIT.
       EXIT.


    1200-WRITE-TO-SM.
*
*   THIS ROUTINE WILL USE STORHOUSE WRITE FUNCTION TO TRANSFER
*   RECORDS TO A STORHOUSE FILE.

*
       CALL 'LSMCALL' USING SM1-FUNCTION-WRITE
                            SM1-OPEN-TOKEN
                            SM1-RETURN-CODE
                            SM1-FILE-BUFFER
                            SM1-FILE-BUFFER-LENGTH
                            SM1-RECORD-NUMBER.

*
*   CHECK FOR A SUCCESSFUL WRITE.  IF IT FAILED, CLOSE THE FILE
*   WITH THE ABORT FLAG SET TO 1, DISCONNECT FROM STORHOUSE, AND
*   END ALL PROCESSING.
*
*   NOTE:  CLOSE PROCESSING WILL RETRIEVE ANY EMESSAGES.
*
       IF SM1-RETURN-CODE EQUAL ZERO NEXT SENTENCE
          ELSE
             MOVE SM1-FUNCTION-WRITE TO ERROR-SM-FUNCTION
             PERFORM 2100-PRINT-SM-ERROR THRU 2100-EXIT
             MOVE +1 TO SM1-CLOSE-ABORT-FLAG
             PERFORM 1300-CLOSE-SM-FILE THRU 1300-EXIT
             PERFORM 700-DISCONNECT THRU 700-EXIT
             PERFORM FINAL-PROCESSING.

    1200-EXIT.
       EXIT.


    1300-CLOSE-SM-FILE.
*
*   CLOSE THE STORHOUSE FILE.
*
*   NOTE:
*
*   SINCE THE OPEN FUNCTION SET THE EMSG FLAG TO 1, THIS ROUTINE WILL CALL
*   EMSG TO ALLOW THE STORHOUSE INTERFACE MODULES TO PERFORM STORAGE CLEANUP.
*
       CALL 'LSMCALL' USING SM1-FUNCTION-CLOSE
```

```
                          SM1-OPEN-TOKEN
                          SM1-RETURN-CODE
                          SM1-CLOSE-ABORT-FLAG.

      PERFORM 1600-RETRIEVE-XFRLINK-EMSGS THRU 1600-EXIT.

   1300-EXIT.
      EXIT.


   1500-RETRIEVE-CMDLINK-EMSGS.
*
*  THIS ROUTINE WILL EXTRACT ALL MESSAGES RETURNED FROM STORHOUSE
*  AT THE COMMAND LINK (SESSION) LEVEL.  EACH RETURNED MESSAGE IS THEN
*  PRINTED.
*
      MOVE ZERO TO SM1-EMSG-RETURN-CODE.
      PERFORM 1510-CALL-EMSG THRU 1510-EXIT
         UNTIL SM1-EMSG-RETURN-CODE NOT EQUAL ZERO.

   1500-EXIT.
      EXIT.

   1510-CALL-EMSG.
*
*  USE THE EMESSAGE FUNCTION TO RETRIEVE ANY STORHOUSE MESSAGES.
*
      CALL 'LSMCALL' USING SM1-FUNCTION-EMSG
                           SM1-CONNECT-TOKEN
                           SM1-EMSG-RETURN-CODE
                           SM1-EMSG-BUFFER
                           SM1-EMSG-BUFFER-LENGTH
                           SM1-EMSG-MSG-LENGTH.

*  NOW PRINT THE RETRIEVED MESSAGE (IF EMSG RETURN CODE IS 0).

      IF SM1-EMSG-RETURN-CODE EQUAL ZERO
      PERFORM 2000-PRINT-EMSG THRU 2000-EXIT.

   1510-EXIT.
      EXIT.

   1600-RETRIEVE-XFRLINK-EMSGS.
*
*  THIS ROUTINE WILL EXTRACT ALL MESSAGES RETURNED FROM
*  STORHOUSE AT THE TRANSFER LINK (DATA) LEVEL.
*  EACH RETURNED MESSAGE IS THEN PRINTED.
*

      MOVE ZERO TO SM1-EMSG-RETURN-CODE.
      PERFORM 1610-CALL-EMSG THRU 1610-EXIT
         UNTIL SM1-EMSG-RETURN-CODE NOT EQUAL ZERO.

   1600-EXIT.
      EXIT.

   1610-CALL-EMSG.
```

```
*
*  USE THE EMESSAGE FUNCTION TO RETRIEVE ANY STORHOUSE MESSAGES.
*
      CALL 'LSMCALL' USING SM1-FUNCTION-EMSG
                           SM1-OPEN-TOKEN
                           SM1-EMSG-RETURN-CODE
                           SM1-EMSG-BUFFER
                           SM1-EMSG-BUFFER-LENGTH
                           SM1-EMSG-MSG-LENGTH.

*  NOW PRINT THE RETRIEVED MESSAGE (IF EMSG RETURN CODE IS 0).

      IF SM1-EMSG-RETURN-CODE EQUAL ZERO
         PERFORM 2000-PRINT-EMSG THRU 2000-EXIT.

   1610-EXIT.
      EXIT.


   2000-PRINT-EMSG.

*  THIS ROUTINE PRINTS STORHOUSE EMESSAGES.

      MOVE SM1-EMSG-BUFFER TO PRINT-RECORD-DATA.
      MOVE '0' TO PRINT-RECORD-CC.
      WRITE PRINT-RECORD.
      MOVE SPACES TO PRINT-RECORD.

   2000-EXIT.
      EXIT.


   2100-PRINT-SM-ERROR.

*  THIS ROUTINE PRINTS AN ERROR MESSAGE IF A STORHOUSE FUNCTION FAILS

      MOVE SM1-RETURN-CODE TO ERROR-SM-RETURN.
      MOVE ERROR-RETURNED-FROM-SM TO PRINT-RECORD-DATA.
      MOVE '0' TO PRINT-RECORD-CC.
      WRITE PRINT-RECORD.
      MOVE SPACES TO PRINT-RECORD.

   2100-EXIT.
      EXIT.


   2200-PRINT-SM-RECORD.

*  THIS ROUTINE PRINTS A RECORD READ FROM STORHOUSE
      MOVE SM1-FILE-BUFFER TO PRINT-RECORD-DATA.
      MOVE ' ' TO PRINT-RECORD-CC.
      WRITE PRINT-RECORD.
      MOVE SPACES TO PRINT-RECORD.

   2200-EXIT.
      EXIT.
```

Appendix

A

Pass-Through Functions

Pass-through functions allow application programs direct access to StorHouse capabilities by sending and receiving StorHouse ASCII messages. In this mode, messages are passed directly from StorHouse to the application program without any manipulation by the StorHouse Subsystem. This mode is intended primarily for use by FileTek-provided host utility software, such as the (TSO) Interactive Interface.

To use these functions, the application program must have access to message structure definitions that are generally not distributed. The C language implementation is available by special order only.

Installations can limit the use of pass-through functions to authorized programs.

The four pass-through functions are:

- PTOPEN
- PTWRTOSM
- PTRDFRSM
- CONFIG

These functions are described in the following sections.

PTOPEN

PTOPEN establishes a data transfer link between the user program and StorHouse and sets the direction of the data flow. This function is used only for transfers of complete files.

PTOPEN requires StorHouse standard features.

Statement Format for C

```
/*
 *Data Areas...
 */
char  *ctoken ;      /* C-Token returned by CONNECT */
char  *otoken ;      /* O-Token, set by PTOPEN */

long  returncd ;     /* Return-code */
long  msgflag ;      /* Messages Flag */
long  direct[11] ;   /* File Info list */

char  sysid[8] ;     /* Host-Id for XP for Data Link */
char  link[8] ;      /* Link-Id for Data Link */
char  vcptopn[16] ;  /* For function name */
char  mode[6] ;      /* Open Mode string */
char  file_name[56];
char  group_name[8];
struct
{
   char   volumeset_name[8];
   char   fileset_name[8];
}file_location;

longLSMCALL( ) ;

/*
 *Setup data areas prior to PTOPEN call
 */
strncpy( vcptopn, "PTOPEN          ", (sizeof vcptopn) ) ;
strncpy( mode, "READ  ", (sizeof mode) ) ;
msgflag = 1 ;/* Return Messages*/

/*
 * "sysid," "link" must be set up from data received from
 * the Storage Machine.  Most of the direct list must also
 * be set from data received from the Storage Machine; only
 * the first and last entries are shown here.
 */
direct[0]  = 10 ;/* Ten entries in list */
direct[10] = 0 ;/* Default buffering */

/*
```

```
 * The file name, group name, and file location information
 * must be set from the MIVGASG/MIVPASG message
 * "hs" structure members.  This setup is not shown here.
 */
/*
 *PTOPEN Call
 */
LSMCALL( vcptopn, &ctoken, &returncd, &msgflag,
         &otoken, mode, file_name, group_name,
         &file_location, sysid, link, direct ) ;

if ( returncd ! = 0 )
{
   /*
    *Error Handling
    */
}
```

Parameter Overview

CTOKEN The session identifier returned by CONNECT.

RETURNCD Final status from the requested operation; see “Return Codes.”

MSGFLAG An integer set to zero or non-zero. If non-zero, MSGFLAG indicates that the caller requires text messages from all session errors, including PTOPEN/CLOSE function errors. If set to zero, messages may not be retrievable if the session has terminated.

OTOKEN A variable set by PTOPEN to the file identifier. The application program should not manipulate (in particular, not cause arithmetic conversion to) the result; it should only be used as the OTOKEN parameter to other function calls for this file.

MODE A 6-byte character string that identifies the file reference mode. Valid modes are READ and WRITE.

FILE_NAME A 56-byte character string that contains the StorHouse file name. This name is only used to identify the file in SMF records.

GROUP_NAME An 8-byte character string that contains the group identifier. This field is only used to identify the file in SMF records.

FILE_LOCATION An array of two 8-character strings that contain the volume set and file set names. These fields are only used to identify the file in SMF records.

SYSID An 8-byte character string that provides the network identifier for the StorHouse system being used. This information must be extracted from the MIVxFILE message by the caller (x is P or G).

LINK An 8-byte character string that identifies the network OFFER name. This information must be extracted from the MIVxFILE message by the caller.

DIRECT An array of variables that supply file characteristics that are kept in the StorHouse directory entry for the file. The caller must supply these values for both read and write operations. This information is used by the StorHouse Subsystem to assemble and disassemble data frames.

Return Codes

Any Non-Zero Value The data link session was not established.

Detailed Function Description

PTOPEN directs the StorHouse Subsystem to establish a data link to StorHouse over the network. The Subsystem manages the data link and exchanges blocks (frames) with StorHouse. The transfer between the Subsystem and the user program is in records, which may be any arbitrary unit of data that the user chooses.

The caller is responsible for initiating the transfer request to StorHouse and processing the MIVxASG and MIVxFILE (x is P or G) messages from StorHouse. The caller must have sent the MIVxASG response message to StorHouse. Network link and file characteristics information in these messages are supplied to the Subsystem through parameter values passed to PTOPEN. PTOPEN completes as soon as the network data link is established. Standard sequential read and write can then be used to transfer records. CLOSE should be called after all data has been transferred.

Notes

- Only one data transfer can be active at a time when PTOPEN is used.
- A session can be established in one task (under one TCB) and then used in another task; however, only one session-related function can be performed at one time for one session. Serialization between multiple tasks is the user's responsibility.

  PTOPEN must be considered a session-related function.
- If a file is opened and closed under one TCB and read or written from another TCB, the two tasks must share Subpool 0 storage. If one of these tasks is a subtask of the other, this is accomplished by the SZERO=YES operand on the ATTACH MACRO (this is the default value).
- Files transferred using PTOPEN can only be processed sequentially.

PTWRTOSM

PTWRTOSM sends an ASCII message to StorHouse.

PTWRTOSM requires StorHouse standard features.

Statement Format for C

```
/*
 *Data Areas...
 */
char  *ctoken ;      /* C-Token returned by CONNECT */
long  returncd ;     /* Return-code */
long  smsglen ;      /* Message length */
char  ptwrtosm[16] ; /* For Function name */
char  smessage[768] ;/* Message text string */

long  LSMCALL( ) ;

/*
 * Setup data areas prior to PTWRTOSM call
 */
strncpy( ptwrtosm, "PTWRTOSM       ", (sizeof ptwrtosm) ) ;
smsglen = xxxx ;     /* Set to actual msg length*/

/*
 *PTWRTOSM Call
 */
LSMCALL( ptwrtosm, &ctoken, &returncd, smessage, &smsglen );

if ( returncd != 0 )
{
   /*
    *Error Handling
    */
}
```

Parameter Overview

CTOKEN — The session identifier returned by CONNECT.

RETURNCD — Final status from the requested operation; see “Return Codes.”

MESSAGE — The buffer containing the message to be sent to StorHouse.

SMSGLEN — The length of the message in the MESSAGE buffer.

Return Codes

Any Non-Zero Value — PTWRTOSM was not processed by StorHouse.

Detailed Function Description

PTWRTOSM (WRite-TO-Storage-Machine) allows the application program to send a StorHouse ASCII message (MIZxxx structure) to StorHouse. The actual code used in the message must be EBCDIC; translation is performed by the Subsystem or network so that the message received by StorHouse is in ASCII.

This function completes as soon as the Subsystem has queued the message for delivery to StorHouse. To receive response messages from StorHouse, PTRDFRSM must be used. (Refer to the function description section for PTRDFRSM.)

Notes

- The maximum length required to retrieve a response is 768 bytes. Any message longer than this is rejected.
- This function is essentially an immediate operation and therefore has no asynchronous form (ASPTWRTOSM).
- The formats of the messages are available as special order information. Ask your system administrator to contact your FileTek customer support representative.

PTRDFRSM

PTRDFRSM queues a request to receive the next ASCII message (structure) sent to this session by StorHouse.

PTRDFRSM requires StorHouse standard features.

Statement Format for C

```
/*
 *  Data Areas...
 */
char  *ctoken ;      /* C-Token returned by CONNECT */

long  returncd ;     /* Return-code */
long  resbufsz ;     /* Message length */
long  ecb ;          /* Event Control Block (IBM/MVS) */
char  ptrdfrsm[16] ; /* For Function name */
char  resbuf[772] ;  /* Message input string */

long  LSMCALL( ) ;

/*
 *Setup data areas prior to PTWRTOSM call
 */
strncpy( ptrdfrsm, "PTRDFRSM       ", (sizeof ptrdfrsm) ) ;
resbufsz = 772 ;     /* Set to area size*/

/*
 * PTRDFRSM Call
 */
LSMCALL( ptrdfrsm, &ctoken, &returncd,
         resbuff, &resbufsz, &ecb ) ;

if ( returncd != 0 )
{
   /*
    *Error Handling
    */
}
```

Parameter Overview

CTOKEN — The session identifier returned by CONNECT.

RETURNCD — Final status from the requested operation; see “Return Codes.”

RESBUFF — The buffer into which the message from StorHouse is copied. The first four bytes of this area are the actual length of the message. The message follows at the fifth byte.

RESBUFSZ — The size of the RESBUFF buffer.

ECB — The Event Control Block that is POSTed when a message has been copied to RESBUFF.

Return Codes

Any Non-Zero Value — PTRDFRSM was not processed by the Subsystem.

Detailed Function Description

PTRDFRSM (ReaD-FRom-Storage-Machine) allows the application program to receive an ASCII message from StorHouse. The code used in the message is EBCDIC. Translation from ASCII is performed by the Subsystem or network prior to delivery of the message to the caller's buffer.

This function completes as soon as the Subsystem has queued the request. The ECB supplied is POSTed when the message has been copied. The caller should not modify or reference the buffer area until after this POST.

Notes

- The maximum length required to retrieve a response is 772 bytes. The longest message is 768 bytes. The first four bytes are required for the length. PTRDFRSM will fail if a smaller buffer is supplied. However, a larger buffer can be supplied.
- PTRDFRSM is essentially an immediate operation. There is no asynchronous form (ASPTRDFRSM). Also, because the function completes immediately, ABORT cannot be used to cancel this operation. If the session is signed-off while a read request is still active, the request is canceled, and the caller can free the buffer area as soon as DISCONNECT completes.
- The formats of the messages are available as special order information. Ask your system administrator to contact your FileTek customer support representative.

CONFIG

CONFIG allows retrieval of the configuration parameters used by the StorHouse Subsystem.

CONFIG requires StorHouse standard features.

Statement Format for C

```
/*
 * Data Areas...
 */
char  *ctoken;/* C-Token returned by CONNECT */

long  returncd;      /* Return-code */
char  config[16];    /* For Function name */
char  conval[1024];  /* Configuration Table area */

long  LSMCALL( ) ;

/*
 *Setup data areas prior to CONFIG call
 */
strncpy( config, "CONFIG          ", (sizeof config) ) ;

/*
 * CONFIG Call
 */
LSMCALL( config, &ctoken, &returncd, conval ) ;

if ( returncd != 0 )
{
   /*
    *Error Handling
    */
}
```

Parameter Overview

CTOKEN — The session identifier returned by CONNECT.

RETURNCD — Final status from the requested operation; see “Return Codes.”

CONVAL — The buffer into which the configuration structure is copied.

Return Codes

Any Non-Zero Value — CONFIG was not processed by the StorHouse Subsystem.

Detailed Function Description

CONFIG copies the StorHouse Subsystem host configuration parameters to a caller-defined area.

Note

The format of the configuration area is available in the form of a C language "typedef" statement. This is special order information. Ask your system administrator to contact your FileTek customer support representative.

Appendix

B

ALC Macro Definition

This appendix provides documentation for accessing the Callable Interface from programs coded in IBM Assembler language. The assumed operating system environment is MVS, either SP1.3 or SP2.X.

This appendix is divided into two sections. The first section documents the Assembler MACRO LSMCALL, which facilitates construction of parameter lists and calling sequences. The second section provides an example of a specific call, both with the MACRO and with the standard assembler CALL statement.

LSMCALL – Call the Callable Interface Program

LSMCALL builds the parameter list required for a call to the LSMCALL program, which is the single entry point for all function requests to the StorHouse Callable Interface. LSMCALL also generates the instructions to set up register 1 (pointer to the parameter list) and register 15 (LSMCALL entry point address) and to issue the branch to LSMCALL.

The three forms of the LSMCALL macro instruction (standard, list, and execute) are written as follows in Table B-1.

Table B-1: LSMCALL Macro Instruction

name	symbol. Begin name in column 1.
b	One or more blanks must precede LSMCALL
LSMCALL	
b	One or more blanks must follow LSMCALL
function	any valid function code, see Chapter 5, "Callable Interface Functions."
,CTOKEN=c-token ,OTOKEN=o-token	x-token: RX-type address, or register (2) - (12)
,RC=return-code addr	return-code addr: RX-type address, or register (2) - (12)
,TYPE=SYNC ,TYPE=ASYNC	DEFAULT: TYPE=SYNC
Additional keyword operand requirements are dictated by the value of the function parameter. For information about these requirements, refer to Chapter 5, "Callable Interface Functions."	
,keyword=value addr	value addr: RX-type address, or register (2) - (12)
...	additional keyword/value entries
,MF=L ,MF=(E,ctrl prog) ,MF=S	ctrl prog: RX-type address, or register (2) - (12) DEFAULT: MF=S

Required Parameters

The first three parameters are required for all uses of LSMCALL. These parameters are:

- Function code
- C-TOKEN or O-TOKEN
- Return code.

These parameters are described below.

function
Specifies an identifier that is one of the function names documented in Chapter 5, "Callable Interface Functions." The identifier is a string written in uppercase (but including any characters). For example, the function value to open a VRAM file is OPEN-VRAM.

CTOKEN=c-token
Specifies the location of a fullword to be used as the connect token. CONNECT places the C-TOKEN value in this fullword; all other session-type calls require the address of a fullword containing the value returned by CONNECT.

OTOKEN=o-token
Specifies the location of a fullword to be used as the open token. OPEN-SEQ, OPEN-VRAM, or PTOPEN places the O-TOKEN value in this fullword; all other data transfer-type calls require the address of a fullword containing the value returned by OPEN.

RC=return-code addr
Specifies the location of a fullword to be used to receive the return code from the requested function.

Optional Parameters

TYPE
TYPE=SYNC
TYPE=ASYNC
Specifies that the function is to be performed synchronously (TYPE=SYNC) with control returned to the caller only when the operation has been completed or that the function is to be performed asynchronously (TYPE=ASYNC) with control returned to the caller as soon as the request is queued. (See Chapter 5, "Callable Interface Functions.")

TYPE is used for all function codes but can assume its default value. Before coding TYPE=ASYNC, check the function description in Chapter 5 to ensure that the particular function supports asynchronous requests. Note that the macro builds the function name string with AS prefixed, so the function parameter should not include the leading AS designation.

MF
MF=S
MF=L
MF=(E,ctrl addr)
Specifies the type of parameter list to be generated by this execution of the macro.

- MF=S builds a non-reentrant parameter list in-line.
- MF=L defines the data area to be used as a parameter list by the execute (MF=E) form of the macro.

- MF=E builds a re-entrant parameter list using the data area built by the list (MF=L) form of the macro. The name of the area built by the list form of the macro must be specified as the second subparameter of this operand. If MF=L is coded, any parameters specified are ignored, and all required parameters must be specified on the MF=E form of the macro.

  The Macro Form parameter allows standard in-line call and parameter list generation (which is not re-entrant) or allows remote parameter list generation to build re-entrant parameter lists. MF can be used for all functions.

Remaining Keywords

The remaining keywords are listed in alphabetical order below. The specific keyword/value specifications required for a function call can be determined from the function definition in Chapter 5, "Callable Interface Functions."

ABORTFL=xfer-abort-flag addr
Specifies the location of a fullword whose contents are used to indicate whether the file transfer associated with the CLOSE is to be aborted. The caller must preset that fullword to zero for a normal CLOSE or to any non-zero value to indicate a transfer-abort condition.

ACCT=account addr
Specifies the location of a 12-byte character string containing the StorHouse account identifier for the session to be connected.

ACCTPW= password add
Specifies the location of a 32-byte character string containing the password associated with the StorHouse account identifier.

ATTRIBS=file-attrib addr
Specifies the location of an array of fullwords that specify the attributes for the file being opened.

BUFFER=buffer addr
Specifies the location of the data buffer to be used in a StorHouse data transfer operation.

BUFFERL=buffer-size addr
Specifies the location of a fullword that contains the length in bytes of the data buffer specified by the BUFFER operand.

CHECKPOINT= checkpoint addr
Specifies the location of a fullword that contains the checkpoint number where OPEN-VRAM will restart the file append operation.

CHKPT=checkpoint addr
Specifies the location of a fullword to receive the checkpoint number.

CMND=command-string addr
Specifies the location of a character string, a maximum of 255 bytes long, containing a StorHouse command or a response to a StorHouse Prompt and Read, to be executed by SM-CMD-INTF.

CMNDL=string-length addr
Specifies the location of a fullword containing the length of the data in the command buffer identified by the CMND=operand.

ECBA=return-ecb-addr	Specifies the location of a fullword to receive the address of the Event Control Block (ECB) returned by ECBADDR.
FILE=file-name addr	Specifies the location of a 56-byte character string that contains the StorHouse file name.
FILEPW=file-passwords addr	Specifies the location of an array containing three (OPEN or OPEN-SEQ), or two (OPEN-VRAM) 8-byte entries, where each entry contains a read, write, or delete password for the file specified by the FILE operand.
GROUP=group-name addr	Specifies the location of an 8-byte character string that contains the StorHouse group name.
GROUPPW=group-passwords addr	Specifies the location of a array containing three (OPEN or OPEN-SEQ), or two (OPEN-VRAM) 8-byte entries, where each entry contains a read, write, or delete password for the group specified by the GROUP operand.
KEY=index-name addr	Specifies the location of the 56-byte character string that contains the name of the key index to be used.
KEYL=key-length addr	Specifies the location of a fullword that contains the length of the key value specified by the KEY operand.
KEYV=key-value addr	Specifies the location of a character string that contains the value of the key to be used for the record search. The length of the key value is given by the KEYL operand (the maximum length is 254 bytes).
METHOD=access-method addr	Specifies the location of a 24-byte character string that contains the processing method specification for the data transfer operation being opened.
MODE=mode addr	Specifies the location of a 6-byte character string that contains the data transfer mode specification.
MSGFLAG= message-flag addr	Specifies the location of a fullword. The contents of that fullword are used to indicate whether messages will be retrieved (using EMSG) following a CLOSE or an OPEN that returns a failure code. The caller must preset that fullword to zero if messages need not be retained after CLOSE/abnormal OPEN or to any non-zero value to indicate that messages must be retained for subsequent retrieval.
MSGL=returned-message-length addr	Specifies the location of a fullword to receive the length of the message returned by EMSG.
OPTS=command-options addr (when using function: OPEN)	Specifies the location of a 255-byte character string that contains options to be supplied to OPEN.

OPTSL=command-opt-length addr (when using function: OPEN)	Specifies the location of a fullword that contains the length of the string specified by the OPTS operand.
OPTS=file-options array addr (when using function: OPEN-SEQ)	Specifies the location of an array of 32-bit integers providing file options corresponding with modifiers available for the StorHouse GET and PUT file operations.
RCHKPT= checkpoint-num addr	Specifies the location of a fullword that contains the number of the checkpoint at which the restart operation should begin.
RECORDL=return-rec-len addr	Specifies the location of a fullword to receive the length of the record placed in the buffer specified by the BUFFER operand.
RECORDN=rel-rec-num addr	Specifies the location of a fullword that contains the relative record number for the requested record.
REVISION=revision addr	Specifies the location of a fullword that contains the revision number of the file version.
SMID=sm-identifier addr	Specifies the location of a 6-byte character string that contains the StorHouse identifier.
SSN=subsystem-identifier addr	Specifies the location of a 4-byte character string that contains the StorHouse Subsystem name.
VERSION=version addr	Specifies the location of a fullword that contains the version number of the file identified by the FILE operand.

Assembly Language Standard Call

Note the following when using the standard CALL macro to build the calling sequence for the LSMCALL routine or when writing open code to accomplish this call:

- Parameter list entries are always an address, never a value.
- The parameter list address must be passed in register 1.
- The parameter list is an array of fullwords.
- The areas addressed by the addresses in the parameter list are either character strings or fullword integers.

- The last fullword in the parameter does not have to be flagged (by setting the high-order bit) but can be, if desired.
- The entry point address for LSMCALL must be in register 15.
- The return address must be in register 14.

Because the Interface is usually called from high-level languages that generate parameter lists and calling sequences that always conform to the above, the contents of register 1 and the parameter list are not checked to see if they contain reasonable addresses. If errors are made in these areas, the usual result will be an S0C4 ABEND.

Example: CALL Macro

The following example in Table B-2 shows the CALL macro and data areas for OPEN-VRAM. The file is opened for append processing with keyed records. Note that the file name, group name, and passwords are explicitly coded. Usually, these values are set from information supplied to the program.

Table B-2: CALL Macro

CALL Macro

`CALL  LSMCALL,(OPENVRAM,CTOKEN,RETURNCD,  X`
`  MSGFLAG,OTOKEN,MODE,METHOD,FILE,  X`
`  REVISION,FILEPWS,GROUP,GROUPWS,  X`
`  RELREC.ATTRIB),VL`

The data areas for the CALL macro are:

```
*
*     DATA AREAS...
*
OPENVRAM    DC    CL16'OPEN-VRAM'
CTOKEN      DS    F        BUILT BY CONNECT
RETURNCD    DS    F        RETURNED VALUE
MSGFLAG     DC    F'1'     GET MESSAGES
OTOKEN      DS    F        RETURNED VALUE
MODE        DC    CL6'APPEND'
METHOD      DC    CL24'SEQUENTIAL,KEYED'
REVISION    DC    F'0'     USING CURRENT REVISION
FILEPWS     DC    CL8'READ-PW'
            DC    CL8'WRITE-PW'
GROUP       DC    CL8' '   DEFAULT GROUP
GROUPPWS    DC    CL8' '
            DC    CL8' '
RELREC      DC    F'0'     DON'T-CARE FOR WRITE
ATTRIB      DC    F'8'     EIGHT ENTRIES IN LIST
            DS    F        RETURNED MAXIMUM RECORD LENGTH
            DS    F        RETURNED LAST PHYSICAL RECORD NUM
```

```
DS    F        RETURNED LAST LOGICAL RECORD NUM
DS    F        RETURNED FILE REVISION NUMBER
DS    F        RETURNED FILE TYPE
DC    F'1'     NO XM BLOCKING
DC    F'0'     USING CURRENT VERSION
DC    F'0'     USING NO CHECKPOINT NUMBER
```

Example: LSMCALL Macro

The LSMCALL macro can be used instead of the standard CALL macro. The data areas used above remain the same (except OPEN-VRAM, which is not required). Table B-3 shows the executable code.

Table B-3: LSMCALL Macro

LSMCALL Macro		
LSMCALL	OPEN-VRAM,CTOKEN=CTOKEN,RC=RETURNCD,	X
	MSGFLAG=MSGFLAG,OTOKEN=OTOKEN,	X
	MODE=MODE,METHOD=METHOD,	X
	FILE=FILE,REVISION=REVISION,	X
	FILEPW=FILEPWS,GROUP=GROUP,	X
	GROUPPW=GROUPPWS,RECORDN=RELREC,	X
	ATTRIBS=ATTRIB,MF=S	

The LSMCALL macro can be used to obtain a help listing of the parameters that are required for each function. You can do this by coding 'LSMCALL HELP'.

Appendix

C

Checkpoint/Restart and Programming Guidelines

Appendix C contains additional technical information about programming with the Callable Interface. Information is presented in two sections:

- Checkpoint/Restart
- Programming Guidelines.

The intent of these sections is to provide additional examples and programming tips.

Checkpoint/Restart

CHECKPOINT can only be issued during VRAM, MODE=APPEND data transfer operations. A successful CHECKPOINT guarantees that all data written up to the time of the checkpoint has been received and processed by StorHouse.

Only the current (most recent) revision of a file version, either accessible or software disabled, can be opened at a checkpoint. Opening a file at a checkpoint is referred to as a *restart*.

Examples

This section contains four examples that use OPEN-VRAM and CHECKPOINT. The examples assume that the current version of the VRAM file DATAFILE has three revisions. Revisions 1 and 2 contain no checkpoints. Revision 3 contains three checkpoints, which are referenced here as checkpoints `a`, `b`, and `c`.

Note Actual checkpoints are binary numbers, not alphanumeric characters. The caller should keep track of checkpoint numbers and make no assumptions about their value.) Refer to Table C-1.

Table C-1: DATAFILE Revisions

Revision Number	Checkpoint	Open
1	None	OPEN-VRAM, any MODE
2	None	OPEN-VRAM, any MODE
3	a,b,c	OPEN-VRAM, any MODE or OPEN-VRAM, MODE=APPEND at any checkpoint

Revisions 1 and 2 can be opened in any MODE. Revision 3 can be opened without supplying a checkpoint in any MODE or in MODE=APPEND at checkpoint a, b, or c.

Example 1

In Example 1, the caller opens Revision 3 shown above with MODE=APPEND at checkpoint a and issues CHECKPOINT and CLOSE. The resulting revisions and their checkpoints are:

	Revision Number	Checkpoint
OPEN-VRAM MODE=APPEND	1	None
CHECKPOINT	2	None
CLOSE	3	a,d

Checkpoints b and c in the original revision 3 are no longer accessible. The last checkpoint in the current revision 3 is checkpoint d.

Example 2

In Example 2, the caller opens revision 3, generated in Example 1, with MODE=UPDATE and issues CHANGE, DELETE, and CLOSE. The resulting revisions are:

	Revision Number	Checkpoint
OPEN-VRAM MODE=UPDATE	1	None
CHANGE	2	None

	Revision Number	Checkpoint
DELETE	3	None
CLOSE	4	None

There are now four revisions. Any previous checkpoints are no longer accessible. Checkpoints are only accessible in the current revision.

Example 3

In Example 3, the caller opens revision 3, generated in Example 2, using MODE=APPEND and issues WRITE, CHECKPOINT, WRITE, CHECKPOINT, and ABORT. The resulting revisions are:

	Revision Number	Checkpoint
OPEN-VRAM MODE=APPEND	1	None
WRITE CHECKPOINT	2	None
WRITE CHECKPOINT	3	None
ABORT	4	None
	5	a, b (software disabled)

There are now five revisions. Revision 5 has two checkpoints, a and b, and is marked as software disabled because of the ABORT.

If the caller opens revision 5 in MODE=APPEND and supplies a checkpoint of 0 or no checkpoint number, StorHouse returns a status code of 2630 and the last checkpoint number, in this case checkpoint b.

Example 4

In Example 4, the caller opens revision 4 (from Example 3) with MODE=APPEND and issues WRITE and CLOSE. The resulting revisions are:

	Revision Number	Checkpoint
OPEN-VRAM MODE=APPEND	1	None
WRITE	2	None
CLOSE	3	None

Revision Number	Checkpoint
4	None
5	None

In this example, the user opened an older, accessible revision of the file to *roll back* the current revision, which was software disabled. A new revision 5 containing the appended data now supersedes the software disabled revision 5 from Example 3.

Programming Guidelines

The guidelines in this section apply to programs that use:

- OPEN-SEQ
- OPEN-VRAM with the StorHouse system parameter VRAM_FILE_OPEN set to true, and any mode and access method
- OPEN-VRAM with VRAM_FILE_OPEN set to false, a mode of READ, and an access method of SEQUENTIAL
- OPEN-VRAM with VRAM_FILE_OPEN set to false and a mode of APPEND or UPDATE
- CREATE-OPEN.

A program using one or more of the types of access listed above will never run to completion if the program attempts to have open at the same time files that require use of the same resource.

Defining Resources

Resources include:

- Optical volumes (for write)
- Optical disk drives (ODU)
- Transfer Manager processes.

The system parameter XFR_COUNT limits the number of Transfer Manager processes.

The following situations require use of the same resource:

- Attempting to have files open on more level L volumes than available level L drives
- Attempting to have open for write two or more files that are on the same optical volume
- Attempting to have open more files than the value of XFR_COUNT.

Examples

The two examples in Table C-2 illustrate what can happen when open statements require the use of the same resource. Both examples assume that:

- There are two available optical disk drives.
- All files reside on different optical disks.
- Files are opened using OPEN-SEQ with a mode of READ or OPEN-VRAM with a mode of READ and an access method of SEQUENTIAL:

Table C-2: Example of Open Statements Requiring the Same Resource

Example 1	Example 2
OPEN FILE1	OPEN FILE1
READ FILE1	OPEN FILE2
CLOSE FILE1	OPEN FILE3
OPEN FILE2	READ FILE1
READ FILE2	READ FILE2
CLOSE FILE2	READ FILE3
OPEN FILE3	CLOSE FILE1
READ FILE3	CLOSE FILE2
CLOSE FILE3	CLOSE FILE3

Example 1 executes successfully because an ODU is always available to satisfy each OPEN-SEQ request. Because the close statement for each file releases an ODU, there are no conflicts for shared resources.

In contrast, Example 2 will not run to completion. It attempts to have three level L files open at the same time when there are only two available optical disk drives.

Example 2 will wait indefinitely for an available ODU to satisfy the OPEN FILE3 request. In Example 2, the optical drive is the resource causing the problem.

User Guidelines

Applications and files should be set up to avoid resource conflicts.

- Do not plan to write to files that are in the same volume set at the same time.
- If you must read files concurrently, ensure that there are enough optical drives configured in the system to handle the read requests. If there are enough drives, understand that your application may not run if a drive goes down.
- To prevent problems resulting from an insufficient number of Transfer Manager processes, use the interactive SHOW SYSTEM command to display the value of XFR_COUNT. If more files must be open at the same time than the value of XFR_COUNT, refer the problem to your system administrator.

Permanent Fixes

The following suggestions are *permanent fixes* to a resource conflict involving optical drives. They should not be used as a *temporary solution* for a resource conflict caused by a drive that goes down.

- To prevent problems resulting from an insufficient number of optical drives when level L files that must be open at the same time reside on different optical volumes, verify that there are at least as many optical drives available as level L files. If there are not enough available optical drives, RELOCATE, or move, some of the level L files to a level F file set.

Note RELOCATE is a permanent move that deletes the source. Do not RELOCATE to level F unless you are willing to lose your original level L copy.

- To prevent problems resulting from writing to files residing on the same level L volume, ensure that all level L files that must be open for write at the same time are in different volume sets. If files belong to the same volume set(s):
  - RELOCATE one or more files to a different volume set(s).
  - Write one or more files to the performance buffer rather than directly to a level L volume set. In other words, do not use VTF=DIRECT.

Index

A

ABORT general usage function
- C-TOKEN parameter 5-71
- description 5-72
- O-TOKEN parameter 5-71
- overview 5-71
- R-CODE parameter 5-71
- return codes 5-71

ABORTFL keyword B-4

access privilege 2-2

ACCESS-METHOD parameter 5-28

account 2-1

account identification code (AID) 2-2

ACCOUNT parameter 5-7

account password 2-2

ACCT keyword B-4

ACCTPW keyword B-4

AID (account identification code) 2-2

ALC (assembly language) 1-1

ASCII
- character stream 2-6
- characters, printable 2-3

assembly language (ALC) 1-1

Assembly Language standard call B-6

asynchronous form for functions 5-4

ATTRIBS keyword B-4

B

binary file data representation 2-6

BUFFER keyword B-4

BUFFER parameter
- CHANGE data transfer control function 5-57
- READ data transfer control function 5-39
- READ-KEYED data transfer control function 5-46
- READ-NEXT-KEY data transfer control function 5-48
- READ-RECORD data transfer control function 5-43
- READ-SEQ data transfer control function 5-41
- WRITE-KEY data transfer control function 5-52

BUFFERL keyword B-4

BUFFER-SIZE parameter
- READ data transfer control function 5-39
- READ-KEYED data transfer control function 5-46
- READ-NEXT-KEY data transfer control function 5-48
- READ-RECORD data transfer control function 5-43
- READ-SEQ data transfer control function 5-41

C

C language 1-1

CALL macro, example B-7

Callable Interface entry points
- LSMCALL 5-1
- LSMCICS 5-1

Callable Interface for StorHouse
- function 1-1
- function hierarchy 1-2
- languages invoked from 1-1

operating environment 1-1

CHANGE data transfer control function
- BUFFER parameter 5-57
- description 5-58
- O-TOKEN parameter 5-57
- overview 5-57
- R-CODE parameter 5-57
- RECORD-LENGTH parameter 5-57
- return codes 5-58

character strings, requirements 4-1

CHECK general usage function
- C-TOKEN parameter 5-65
- description 5-66
- O-TOKEN parameter 5-65
- overview 5-65
- R-CODE parameter 5-65
- return codes 5-65

CHECKPOINT file operation function
- description 5-34
- O-TOKEN parameter 5-33
- overview 1-3, 5-33
- R-CODE parameter 5-33
- return codes 5-34
- RETURN-CKPT-NUM parameter 5-33

CHECKPOINT keyword B-4

checkpoint/restart
- description C-1
- example C-2, C-3

CHKPT keyword B-4

CICS
- Interface programs 5-2
- restrictions 5-3

CLOSE file operation function
- description 5-36
- O-TOKEN parameter 5-35
- overview 1-3, 5-35
- R-CODE parameter 5-35
- return codes 5-36
- XFER-ABORT-FLAG parameter 5-35

CMND keyword B-4

CMNDL keyword B-4

COBOL
- function statement format 5-5
- language 1-1

command privilege, StorHouse 2-3

CONFIG pass-through function
- CONVAL parameter A-9
- CTOKEN parameter A-9
- description A-10
- overview A-9
- return codes A-10

CONNECT session control function
- ACCOUNT parameter 5-7
- C-TOKEN parameter 5-6
- description 5-7
- MESSAGE-FLAG parameter 5-7
- overview 1-3, 5-6
- PASSWORD parameter 5-7
- R-CODE parameter 5-6
- return codes 5-7
- SM-IDENTIFIER parameter 5-7
- SUBSYSTEM-IDENTIFIER parameter 5-7

CONVAL parameter A-9

CR-BUF parameter 5-61

CREATE-OPEN file operation function
- C-TOKEN parameter 5-21
- description 5-25
- FILE-ATTRIB parameter 5-22
- FILE-LOCATION parameter 5-22
- FILE-NAME parameter 5-21
- FILE-PASSWORD parameter 5-21
- GROUP-NAME parameter 5-22
- GROUP-PASSWORD parameter 5-22
- MESSAGE-FLAG parameter 5-21
- MODEL-FILE-NAME parameter 5-22
- O-TOKEN parameter 5-21
- overview 1-3, 5-20
- programming guidelines C-4
- R-CODE parameter 5-21
- return codes 5-24

CR-LEN parameter 5-61

C-TOKEN parameter
- ABORT general usage function 5-71
- CHECK general usage function 5-65
- CONNECT session control function 5-6
- CREATE-OPEN file operation function 5-21
- DISCONNECT session control function 5-9
- ECBADDR general usage function 5-67
- EMSG general usage function 5-69
- OPEN-SEQ file operation function 5-14

OPEN-VRAM file operation function 5-28
SM-CMD-INTF StorHouse command submission function 5-61

CTOKEN parameter
CONFIG pass-through function A-9
LSMCALL Assembler MACRO B-3
PTOPEN pass-through function A-3
PTRDFRSM pass-through function A-7
PTWRTOSM pass-through function A-5

current record position
key 3-2
sequential 3-2

D

data transfer control functions
CHANGE 5-57
DELETE 5-55
READ 5-39
READ-KEYED 5-45
READ-NEXT-KEY 5-48
READ-RECORD 5-43
READ-SEQ 5-41
WRITE 5-50
WRITE-KEY 5-52

data transfer link identifier 2-1

default access group 2-2

default access rights 2-2

definitions
account 2-1
account identification code (AID) 2-2
account password 2-2
asynchronous form for functions 5-4
default access group 2-2
default access rights 2-2
entry sequence 3-1
file 2-3
file access group 2-4
file name 2-3
file passwords 2-5
file revision 2-6
file version 2-5
group passwords 2-4
key sequence 3-1
return code 4-2
synchronous form for functions 5-4

DELETE data transfer control function
description 5-56
O-TOKEN parameter 5-55
overview 5-55
R-CODE parameter 5-55
return codes 5-55

DIRECT parameter A-4

DISCONNECT session control function
C-TOKEN parameter 5-9
description 5-10
overview 1-3, 5-9
R-CODE parameter 5-9
return codes 5-9

E

ECB parameter A-8

ECBA keyword B-5

ECBADDR general usage function
C-TOKEN parameter 5-67
description 5-68
O-TOKEN parameter 5-67
overview 5-67
R-CODE parameter 5-67
return codes 5-68
RETURN-ECB-ADDR parameter 5-67

EMSG general usage function
C-TOKEN parameter 5-69
description 5-70
MESSAGE-BUFFER parameter 5-69
MESSAGE-BUFFER-SIZE parameter 5-70
O-TOKEN parameter 5-69
overview 5-69
R-CODE parameter 5-69
return codes 5-70
RETURNED-MESSAGE-LEN parameter 5-70

entry points for Callable Interface
LSMCALL 5-1
LSMCICS 5-1

entry sequenced records 3-1

error handling 5-3

examples
CALL macro B-7
checkpoint/restart C-2, C-3

LSMCALL macro B-8

externally specified parameters, requirements 4-1

F

file 2-3

file access group 2-4

file data representations
- ASCII character stream 2-6
- binary 2-6

FILE keyword B-5

file name 2-3

file operation functions
- CHECKPOINT 1-3, 5-33
- CLOSE 1-3, 5-35
- CREATE-OPEN 1-3, 5-20
- OPEN-SEQ 1-3, 5-13
- OPEN-VRAM 1-3, 5-27

file passwords 2-5

file revision 2-6

file version 2-5

FILE_LOCATION parameter A-3

FILE_NAME parameter A-3

FILE-ATTRIB parameter
- CREATE-OPEN file operation function 5-22
- elements 5-15, 5-23, 5-29
- OPEN-SEQ file operation function 5-15
- OPEN-VRAM file operation function 5-29

FILE-LOCATION parameter
- CREATE-OPEN file operation function 5-22
- OPEN-SEQ file operation function 5-15

FILE-NAME parameter
- CREATE-OPEN file operation function 5-21
- OPEN-SEQ file operation function 5-14
- OPEN-VRAM file operation function 5-28

FILE-OPTIONS parameter
- elements 5-17
- OPEN-SEQ file operation function 5-17

FILE-PASSWORD parameter 5-21

FILE-PASSWORDS parameter
- OPEN-SEQ file operation function 5-15
- OPEN-VRAM file operation function 5-29

FILEPW keyword B-5

forms for functions
- asynchronous 5-4
- synchronous 5-4

FORTRAN language 1-1

function parameter B-3

function statement format for COBOL 5-5

functions
- data transfer control 1-3, 5-38
- file operation 1-3, 5-11
- general usage 5-64
- session control 1-2, 5-5
- StorHouse command 1-4
- StorHouse command submission 5-59

G

general usage functions
- ABORT 5-71
- CHECK 5-65
- ECBADDR 5-67
- EMSG 5-69

GROUP keyword B-5

group passwords 2-4

GROUP_NAME parameter A-3

GROUP-NAME parameter
- CREATE-OPEN file operation function 5-22
- OPEN-SEQ file operation function 5-15
- OPEN-VRAM file operation function 5-29

GROUP-PASSWORD parameter 5-22

GROUP-PASSWORDS parameter
- OPEN-SEQ file operation function 5-15
- OPEN-VRAM file operation function 5-29

GROUPPW keyword B-5

I

indicative text messages 4-2

K

KEY
- keyword B-5
- parameter 5-53

key record position 3-2

key sequenced records 3-1

KEYL keyword B-5

KEY-LENGTH parameter
- READ-KEYED data transfer control function 5-46
- WRITE-KEY data transfer control function 5-53

KEY-NAME parameter 5-46

KEYV keyword B-5

KEY-VALUE parameter 5-46

keywords
- ABORTFL B-4
- ACCT B-4
- ACCTPW B-4
- ATTRIBS B-4
- BUFFER B-4
- BUFFERL B-4
- CHECKPOINT B-4
- CHKPT B-4
- CMND B-4
- CMNDL B-4
- ECBA B-5
- FILE B-5
- FILEPW B-5
- GROUP B-5
- GROUPPW B-5
- KEY B-5
- KEYL B-5
- KEYV B-5
- METHOD B-5
- MODE B-5
- MSGFLAG B-5
- MSGL B-5
- OPTS (command) B-5
- OPTS (file) B-6
- OPTSL B-6
- RCHKPT B-6
- RECORDL B-6
- RECORDN B-6
- REVISION B-6
- SMID B-6
- SSN B-6
- VERSION B-6

L

link identifiers
- data transfer 2-1
- session 2-1

LINK parameter A-4

LSMCALL
- function, Callable Interface entry point 5-1
- macro, example B-8

LSMCALL Assembler MACRO
- ABORTFL keyword B-4
- ACCT keyword B-4
- ACCTPW keyword B-4
- ATTRIBS keyword B-4
- BUFFER keyword B-4
- BUFFERL keyword B-4
- CHECKPOINT keyword B-4
- CHKPT keyword B-4
- CMND keyword B-4
- CMNDL keyword B-4
- CTOKEN parameter B-3
- description B-1
- ECBA keyword B-5
- FILE keyword B-5
- FILEPW keyword B-5
- function parameter B-3
- GROUP keyword B-5
- GROUPPW keyword B-5
- KEY keyword B-5
- KEYL keyword B-5
- KEYV keyword B-5
- METHOD keyword B-5
- MF parameter B-3
- MODE keyword B-5
- MSGFLAG keyword B-5
- MSGL keyword B-5
- OPTS keyword (command) B-5
- OPTS keyword (file) B-6
- OPTSL keyword B-6
- OTOKEN parameter B-3
- RC parameter B-3
- RCHKPT keyword B-6
- RECORDL keyword B-6
- RECORDN keyword B-6

REVISION keyword B-6
SMID keyword B-6
SSN keyword B-6
TYPE parameter B-3
VERSION keyword B-6

LSMCICS function, Callable Interface entry point 5-1

M

MESSAGE parameter A-5

MESSAGE-BUFFER parameter 5-69

MESSAGE-BUFFER-SIZE parameter 5-70

MESSAGE-FLAG parameter
CONNECT session control function 5-7
CREATE-OPEN file operation function 5-21
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28

METHOD keyword B-5

MF parameter B-3

MODE keyword B-5

MODE parameter
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
PTOPEN pass-through function A-3

MODEL-FILE-NAME parameter 5-22

MSGFLAG
keyword B-5
parameter A-3

MSGL keyword B-5

multitasking in sessions 1-4

MVS/SP environment for Callable Interface 1-1

MVS/XA environment for Callable Interface 1-1

O

OPEN file operation function (obsolete) 5-12

OPEN-SEQ file operation function
C-TOKEN parameter 5-14
description 5-18
FILE-ATTRIB parameter 5-15
FILE-LOCATION parameter 5-15
FILE-NAME parameter 5-15
FILE-OPTIONS parameter 5-17
FILE-PASSWORDS parameter 5-15
GROUP-NAME parameter 5-15
GROUP-PASSWORDS parameter 5-15
MESSAGE-FLAG parameter 5-14
MODE parameter 5-14
O-TOKEN parameter 5-14
overview 1-3, 5-13
programming guidelines C-4
R-CODE parameter 5-14
return codes 5-18
VERSION parameter 5-15

OPEN-VRAM file operation function
ACCESS-METHOD parameter 5-28
C-TOKEN parameter 5-28
description 5-31
FILE-ATTRIB parameter 5-29
FILE-NAME parameter 5-28
FILE-PASSWORDS parameter 5-29
GROUP-NAME parameter 5-29
GROUP-PASSWORDS parameter 5-29
MESSAGE-FLAG parameter 5-28
MODE parameter 5-28
O-TOKEN parameter 5-28
overview 1-3, 5-27
programming guidelines C-4
R-CODE parameter 5-28
REL-REC-NUM parameter 5-29
return codes 5-30
REVISION parameter 5-28

OPTS keyword (command) B-5

OPTS keyword (file) B-6

OPTSL keyword B-6

O-TOKEN parameter
ABORT general usage function 5-71
BUFFER data transfer control function 5-50
CHANGE data transfer control function 5-57
CHECK general usage function 5-65
CHECKPOINT file operation function 5-33
CLOSE file operation function 5-35
CREATE-OPEN file operation function 5-21
DELETE data transfer control function 5-55
ECBADDR general usage function 5-67
EMSG general usage function 5-69
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28

R-CODE data transfer control function 5-50
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-45
READ-NEXT-KEY data transfer control function 5-48
READ-RECORD data transfer control function 5-43
READ-SEQ data transfer control function 5-41
RECORD-LENGTH data transfer control function 5-50
RETURN-REC-NUM data transfer control function 5-51
WRITE data transfer control function 5-50
WRITE-KEY data transfer control function 5-52

OTOKEN parameter
LSMCALL Assembler MACRO B-3
PTOPEN pass-through function A-3

P

parameter values, specified by
character strings 4-1
JCL statements 4-1

parameters
ACCESS-METHOD 5-28
ACCOUNT 5-7
BUFFER
CHANGE data transfer control function 5-57
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-46
READ-NEXT-KEY data transfer control function 5-48
READ-RECORD data transfer control function 5-43
READ-SEQ data transfer control function 5-41
WRITE data transfer control function 5-50
WRITE-KEY data transfer control function 5-52
BUFFER-SIZE
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-46
READ-NEXT-KEY data transfer control function 5-48
READ-RECORD data transfer control function 5-43
READ-SEQ data transfer control function 5-41
CONVAL A-9
CR-BUF 5-61
CR-LEN 5-61
C-TOKEN
ABORT general usage function 5-71
CHECK general usage function 5-65
CONNECT session control function 5-6
CREATE-OPEN file operation function 5-21
DISCONNECT session control function 5-9
ECBADDR general usage function 5-67
EMSG general usage function 5-69
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
SM-CMD-INTF StorHouse command submission function 5-61
CTOKEN
CONFIG pass-through function A-9
LSMCALL parameter B-3
PTOPEN pass-through function A-3
PTRDFRSM pass-through function A-7
PTWRTOSM pass-through function A-5
DIRECT A-4
ECB A-8
FILE_LOCATION A-3
FILE_NAME A-3
FILE-ATTRIB
CREATE-OPEN file operation function 5-22
OPEN-SEQ file operation function 5-15
OPEN-VRAM file operation function 5-29
FILE-LOCATION
CREATE-OPEN file operation function 5-22
OPEN-SEQ file operation function 5-15
FILE-NAME
CREATE-OPEN file operation function 5-21
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
FILE-OPTIONS 5-17
FILE-PASSWORD 5-21
FILE-PASSWORDS
OPEN-SEQ file operation function 5-15
OPEN-VRAM file operation function 5-29
function B-3
GROUP_NAME A-3
GROUP-NAME
CREATE-OPEN file operation function 5-22
OPEN-SEQ file operation function 5-15
OPEN-VRAM file operation function 5-29
GROUP-PASSWORD 5-22
GROUP-PASSWORDS

OPEN-SEQ file operation function 5-15
OPEN-VRAM file operation function 5-29
KEY 5-53
KEY-LENGTH
READ-KEYED data transfer control function 5-46
WRITE-KEY data transfer control function 5-53
KEY-NAME 5-46
KEY-VALUE 5-46
LINK A-4
MESSAGE
PTOPEN pass-through function A-3
PTWRTOSM pass-through function A-5
MESSAGE-BUFFER 5-69
MESSAGE-BUFFER-SIZE 5-70
MESSAGE-FLAG
CONNECT session control function 5-7
CREATE-OPEN file operation function 5-21
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
MF B-3
MODE
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
PTOPEN pass-through function A-3
MODEL-FILE-NAME 5-22
O-TOKEN
ABORT general usage function 5-71
CHANGE data transfer control function 5-57
CHECK general usage function 5-65
CHECKPOINT file operation function 5-33
CLOSE file operation function 5-35
CREATE-OPEN file operation function 5-21
DELETE data transfer control function 5-55
ECBADDR general usage function 5-67
EMSG general usage function 5-69
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-45
READ-NEXT-KEY data transfer control function 5-48
READ-RECORD data transfer control function 5-43
READ-SEQ data transfer control function 5-41
WRITE data transfer control function 5-50
WRITE-KEY data transfer control function 5-52
OTOKEN A-3, B-3
PASSWORD 5-7
RC B-3
R-CODE
ABORT general usage function 5-71
CHANGE data transfer control function 5-57
CHECK general usage function 5-65
CHECKPOINT file operation function 5-33
CLOSE file operation function 5-35
CONNECT session control function 5-6
CREATE-OPEN file operation function 5-21
DELETE data transfer control function 5-55
DISCONNECT session control function 5-9
ECBADDR general usage function 5-67
EMSG general usage function 5-69
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-45
READ-NEXT-KEY data transfer control function 5-48
READ-RECORD data transfer control function 5-43
READ-SEQ data transfer control function 5-41
SM-CMD-INTF StorHouse command submission function 5-61
WRITE data transfer control function 5-50
WRITE-KEY data transfer control function 5-52
RECORD-LENGTH
CHANGE data transfer control function 5-57
WRITE data transfer control function 5-50
WRITE-KEY data transfer control function 5-53
REL-REC-NUM
OPEN-VRAM file operation function 5-29
READ-RECORD data transfer control function 5-44
RESBUFF A-8
RESBUFSZ A-8
RESP-BUFFER 5-61
RESP-BUFSIZE 5-61
RESP-INFO 5-61
RETURNCD
CONFIG pass-through function A-9
PTOPEN pass-through function A-3
PTRDFRSM pass-through function A-7

PTWRTOSM pass-through function A-5
RETURN-CKPT-NUM 5-33
RETURN-ECB-ADDR 5-67
RETURNED-MESSAGE-LEN 5-70
RETURN-REC-LEN
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-46
READ-NEXT-KEY data transfer control function 5-49
READ-RECORD data transfer control function 5-44
READ-SEQ data transfer control function 5-42
RETURN-REC-NUM
READ-KEYED data transfer control function 5-46
READ-NEXT-KEY data transfer control function 5-49
READ-SEQ data transfer control function 5-42
WRITE data transfer control function 5-51
WRITE-KEY data transfer control function 5-53
REVISION 5-28
SM-IDENTIFIER 5-7
SMSGLEN A-5
SUBSYSTEM-IDENTIFIER 5-7
SYSID A-3
TYPE B-3
VERSION 5-15
XFER-ABORT-FLAG 5-35

pass-through functions
CONFIG A-9
description A-1
PTOPEN A-2
PTRDFRSM A-7
PTWRTOSM A-5

PASSWORD parameter 5-7

passwords, StorHouse
file 2-5
group 2-4

PL/1 language 1-1

printable ASCII characters 2-3

privileges, StorHouse
access 2-2
command 2-3

programming guidelines
CREATE-OPEN C-4
OPEN-SEQ C-4
OPEN-VRAM C-4

programs for CICS Interface 5-2

PTOPEN pass-through function
CTOKEN parameter A-3
description A-4
DIRECT parameter A-4
FILE_LOCATION parameter A-3
FILE_NAME parameter A-3
GROUP_NAME parameter A-3
LINK parameter A-4
MODE parameter A-3
MSGFLAG parameter A-3
OTOKEN parameter A-3
overview A-2
return codes A-4
RETURNCD parameter A-3
SYSID parameter A-3

PTRDFRSM pass-through function
CTOKEN parameter A-7
description A-8
ECB parameter A-8
overview A-7
RESBUFF parameter A-8
RESBUFSZ parameter A-8
return codes A-8
RETURNCD parameter A-7

PTWRTOSM pass-through function
CTOKEN parameter A-5
description A-6
MESSAGE parameter A-5
overview A-5
return codes A-6
RETURNCD parameter A-5
SMSGLEN parameter A-5

R

RC parameter B-3

RCHKPT keyword B-6

R-CODE parameter
ABORT general usage function 5-71
CHANGE data transfer control function 5-57
CHECK general usage function 5-65
CHECKPOINT file operation function 5-33

CLOSE file operation function 5-35
CONNECT session control function 5-6
CREATE-OPEN file operation function 5-21
DELETE data transfer control function 5-55
DISCONNECT session control function 5-9
ECBADDR general usage function 5-67
EMSG general usage function 5-69
OPEN-SEQ file operation function 5-14
OPEN-VRAM file operation function 5-28
READ data transfer control function 5-39
READ-KEYED data transfer control function 5-45
READ-NEXT-KEY data transfer control function 5-48
READ-RECORD data transfer control function 5-43
READ-SEQ data transfer control function 5-41
SM-CMD-INTF StorHouse command submission function 5-61
WRITE-KEY data transfer control function 5-52

READ data transfer control function
BUFFER parameter 5-39
BUFFER-SIZE parameter 5-39
description 5-40
O-TOKEN parameter 5-39
overview 5-39
R-CODE parameter 5-39
return codes 5-40
RETURN-REC-LEN parameter 5-39

read functions for VRAM files
READ-KEYED 3-2
READ-NEXT-KEY 3-2
READ-RECORD 3-2
READ-SEQ 3-2

READ-KEYED data transfer control function
BUFFER parameter 5-46
BUFFER-SIZE parameter 5-46
description 5-46
KEY-LENGTH parameter 5-46
KEY-NAME parameter 5-46
KEY-VALUE parameter 5-46
O-TOKEN parameter 5-45
overview 5-45
R-CODE parameter 5-45
return codes 5-46
RETURN-REC-LEN parameter 5-46
RETURN-REC-NUM parameter 5-46

READ-KEYED read function 3-2

READ-NEXT-KEY data transfer control function
BUFFER parameter 5-48
BUFFER-SIZE parameter 5-48
description 5-49
O-TOKEN parameter 5-48
overview 5-48
R-CODE parameter 5-48
return codes 5-49
RETURN-REC-LEN parameter 5-49
RETURN-REC-NUM parameter 5-49

READ-NEXT-KEY read function 3-2

READ-RECORD data transfer control function
BUFFER parameter 5-43
BUFFER-SIZE parameter 5-43
description 5-44
O-TOKEN parameter 5-43
overview 5-43
R-CODE parameter 5-43
REL-REC-NUM parameter 5-44
return codes 5-44
RETURN-REC-LEN parameter 5-44

READ-RECORD read function 3-2

READ-SEQ data transfer control function
BUFFER parameter 5-41
BUFFER-SIZE parameter 5-41
description 5-42
O-TOKEN parameter 5-41
overview 5-41
R-CODE parameter 5-41
return codes 5-42
RETURN-REC-LEN parameter 5-42
RETURN-REC-NUM parameter 5-42

READ-SEQ read function 3-2

record sequencing
by entry 3-1
by key 3-1
example 3-2

RECORDL keyword B-6

RECORD-LENGTH parameter
CHANGE data transfer control function 5-57
WRITE-KEY data transfer control function 5-53

RECORDN keyword B-6

REL-REC-NUM parameter
OPEN-VRAM file operation function 5-29
READ-RECORD data transfer control function 5-44

RESBUFF parameter A-8

RESBUFSZ parameter A-8

resource conflicts with optical drives C-6

RESP-BUFFER parameter 5-61

RESP-BUFSIZE parameter 5-61

RESP-INFO parameter
- elements 5-61
- SM-CMD-INTF StorHouse command submission function 5-61

return code, definition 4-2

return codes
- ABORT general usage function 5-71
- CHANGE data transfer control function 5-58
- CHECK general usage function 5-65
- CHECKPOINT file operation function 5-34
- CLOSE file operation function 5-36
- CONFIG pass-through function A-10
- CONNECT session control function 5-7
- CREATE-OPEN file operation function 5-24
- DELETE data transfer control function 5-55
- DISCONNECT session control function 5-9
- ECBADDR general usage function 5-68
- EMSG general usage function 5-70
- OPEN-SEQ file operation function 5-18
- OPEN-VRAM file operation function 5-30
- PTOPEN pass-through function A-4
- PTRDFRSM pass-through function A-8
- PTWRTOSM pass-through function A-6
- READ data transfer control function 5-40
- READ-KEYED data transfer control function 5-46
- READ-NEXT-KEY data transfer control function 5-49
- READ-RECORD data transfer control function 5-44
- READ-SEQ data transfer control function 5-42
- SM-CMD-INTF StorHouse command submission function 5-62
- WRITE data transfer control function 5-51
- WRITE-KEY data transfer control function 5-53

RETURNCD parameter
- CONFIG pass-through function A-9
- PTOPEN pass-through function A-3
- PTRDFRSM pass-through function A-7
- PTWRTOSM pass-through function A-5

RETURN-CKPT-NUM parameter 5-33

RETURN-ECB-ADDR parameter 5-67

RETURNED-MESSAGE-LEN parameter 5-70

RETURN-REC-LEN parameter
- READ data transfer control function 5-39
- READ-KEYED data transfer control function 5-46
- READ-NEXT-KEY data transfer control function 5-49
- READ-RECORD data transfer control function 5-44
- READ-SEQ data transfer control function 5-42

RETURN-REC-NUM parameter
- READ-KEYED data transfer control function 5-46
- READ-NEXT-KEY data transfer control function 5-49
- READ-SEQ data transfer control function 5-42
- WRITE-KEY data transfer control function 5-53

REVISION
- keyword B-6
- parameter 5-28

S

sequential record position 3-2

session control functions
- CONNECT 1-3, 5-6
- DISCONNECT 1-3, 5-9

session link identifier 2-1

sessions and multitasking 1-4

SM-CMD-INTF StorHouse command submission function
- CR-BUF parameter 5-61
- CR-LEN parameter 5-61
- C-TOKEN parameter 5-61
- description 5-62
- overview 5-60
- R-CODE parameter 5-61
- RESP-BUFFER parameter 5-61
- RESP-BUFSIZE parameter 5-61
- RESP-INFO parameter 5-61
- return codes 5-62

SMID keyword B-6

SM-IDENTIFIER parameter 5-7

SMSGLEN parameter A-5

SSN keyword B-6

StorHouse
- account 2-1
- account identification code (AID) 2-2

account password 2-2
default access group 2-2
default access rights 2-2
file 2-3
file access group 2-4
file name 2-3
file passwords 2-5
file revision 2-6
file version 2-5
group passwords 2-4
privileges 2-2

StorHouse Callable Interface
function 1-1
function hierarchy 1-2
languages invoked from 1-1
operating environment 1-1

StorHouse command submission function, SM-CMD-INTF 5-60

SUBSYSTEM-IDENTIFIER parameter 5-7

synchronous form for functions 5-4

SYSID parameter A-3

T

Task Control Block (TCB) 1-4

text messages 4-2

TYPE parameter B-3

V

VERSION
keyword B-6
parameter 5-15

W

WRITE data transfer control function
BUFFER parameter 5-50
description 5-51
O-TOKEN parameter 5-50
overview 5-50
R-CODE parameter 5-50
RECORD-LENGTH parameter 5-50
return codes 5-51
RETURN-REC-NUM parameter 5-51

WRITE-KEY data transfer control function
BUFFER parameter 5-52
description 5-53
KEY parameter 5-53
KEY-LENGTH parameter 5-53
O-TOKEN parameter 5-52
overview 5-52
R-CODE parameter 5-52
RECORD-LENGTH parameter 5-53
return codes 5-53
RETURN-REC-NUM parameter 5-53

X

XFER-ABORT-FLAG parameter 5-35