FUJITSU

Fujitsu Software NetCOBOL V13.0

COBOL File Access Subroutines User's Guide

Linux(64)

J2UL-2881-01ENZ0(00)
July 2023

Preface

What are the NetCOBOL File Access Routines?

The NetCOBOL File Access Routines are Application Programming Interface (API) functions used to access COBOL files from C language programs.

The API provides a common interface to COBOL files. The C application developer is not required to develop low-level file handling code or understand the intricacies of the physical file storage system.

By using the function calls outlined in this API, C applications can safely access COBOL files in a manner consistent with the COBOL Runtime System. Additionally, the API's serve to isolate the C application from changes in the COBOL file behavior that may change with newer versions of NetCOBOL.

Purpose of This Manual

This manual explains the COBOL File Access Routines only.

The intended audience of this manual should be familiar with the concept of COBOL file structures and I-O techniques.

This software supports the development and operation of C applications that access COBOL files. The reader is assumed to have sufficient understanding of the C programming language.

This manual does not explain how COBOL file I-O works. For this information, refer to "NetCOBOL Language Reference", "NetCOBOL User's Guide" and "NetCOBOL Messages"

Abbreviations

The names of products described in this manual are abbreviated as follows:

Product Name	Abbreviation
Red Hat(R) Enterprise Linux(R) 9 (for Intel64) Red Hat(R) Enterprise Linux(R) 8 (for Intel64)	Linux or Linux(64)
Microsoft(R) Windows Server(R) 2022 Datacenter Microsoft(R) Windows Server(R) 2022 Standard	Windows Server 2022
Microsoft(R) Windows Server(R) 2019 Datacenter Microsoft(R) Windows Server(R) 2019 Standard	Windows Server 2019
Microsoft(R) Windows Server(R) 2016 Datacenter Microsoft(R) Windows Server(R) 2016 Standard	Windows Server 2016
Microsoft(R) Windows Server(R) 2012 R2 Datacenter Microsoft(R) Windows Server(R) 2012 R2 Standard Microsoft(R) Windows Server(R) 2012 R2 Foundation	Windows Server 2012 R2
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012
Windows(R) 11 Home Windows(R) 11 Pro Windows(R) 11 Enterprise Windows(R) 11 Education	Windows 11
Windows(R) 10 Education Windows(R) 10 Home	Windows 10

Product Name	Abbreviation
Windows(R) 10 Pro Windows(R) 10 Enterprise	

Microsoft Windows products listed in the table above are referred to in this manual as "Windows".

Trademarks

- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Red Hat and Red Hat Enterprise Linux are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.
- Intel and Itanium are trademarks of Intel Corporation or its subsidiaries.
- UNIX is a registered trademark of The Open Group.
- Microsoft, Windows, and Windows Server are trademarks of the Microsoft group of companies.
- All other trademarks are the property of their respective owners.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

July 2023

Contents

Chapter 1 Before Using this Software

1.1 About COBOL File Access Routines

The COBOL File Access Routines are Application Programming Interface (API) functions used to access COBOL files from C language programs.

When a COBOL file access routine is called, a file access module provided by the COBOL Runtime System is called by the COBOL file access routine, and the file is accessed.

The COBOL file access routines enable the following operations:

- File I-O to or from the existing resources, such as reading or rewriting a file created by a COBOL application
- Creation of COBOL files of any organization type
- Sharing or exclusive use of files with COBOL applications
- Analysis of file attributes/record key list of existing indexed files

Note

Locking control of file is enabled for access using the COBOL file access routines. For more details, refer to "Locking Control of Files" in "NetCOBOL User's Guide."

1.2 Requirements

It is necessary to prepare C compiler as a language for development to develop the application software by using the COBOL file access routine.

1.3 Environment Setup

When NetCOBOL is installed, the COBOL file access routine support is copied to the same directory as NetCOBOL.

Please confirm COBOL installation directory is set to environment variable LD_LIBRARY_PATH after NetCOBOL is installed.

If it is not set, please add the installation directory (/opt/FJSVcbl64/lib) of COBOL to this environment variable.

Chapter 2 Usage

2.1 Making C Source Programs

C programs that use this facility may be created with the development tools or with any text editor.

Put the following description in the top of C source program to explicitly include the header file:

```
#include "cobfa.h"
```

2.2 Compilation of C Source Programs

Compile the C source programs.

When compiling the C source programs, be sure to specify the following compiler options:

- Option used to search the include file from the directory

```
-I <directory>
```

Specify the directory where NetCOBOL was installed in <*directory*>.

Example

```
-I /opt/FJSVcbl64/include
```

When a C program that operates in the Multithreading environment is compiled, it is necessary to specify the Multithreading compilation option of the C compiler. Refer to your manual of C compiler for details.

2.3 Linking Objects

Link the objects to make an execution program.

When linking the objects, be sure to link one of the following object files:

- librcobfa.so

2.4 Application Program Execution

There are no precautions at this time. I-O to or from a file is carried out under the same considerations of file sharing or exclusive use with COBOL application programs.

Chapter 3 API Functions

This chapter explains the File Access functions, calling parameters and returned values. Using this application programming interface (API) allows applications to access the features of the COBOL file system.

The File Access API consists of the following functional categories:

Category	Function	Description	Access
File Oriented	3.1 cobfa_close()	Close a file	-
	3.6 cobfa_indexinfo()	Get the file attributes or key structure of an indexed file	-
	3.7 cobfa_open()	Open a file	-
	3.11 cobfa_reclen()	Get the length of the current record	-
	3.12 cobfa_recnum()	Get the relative record number of the positioned record	-
	3.13 cobfa_release()	Release all record locks	-
	3.23 LOCK_cobfa()	Set an exclusive lock	-
	3.24 UNLOCK_cobfa()	Release an exclusive lock	-
File Positioning	3.18 cobfa_stkey()	Position to the record indicated by the value of an arbitrary key	SD
	3.19 cobfa_strec()	Position to the record indicated by the relative record number	SD
Record Oriented	3.2 cobfa_delcurr()	Delete the current record after a sequential read	S
	3.3 cobfa_delkey()	Delete the record indicated by the value of the primary record key	RD
	3.4 cobfa_delrec()	Delete the record indicated by a relative record number	RD
	3.8 cobfa_rdkey()	Reads the record indicated by the value of an arbitrary key	RD
	3.9 cobfa_rdnext()	Reads a record sequentially	S
	3.10 cobfa_rdrec()	Read the record indicated by the relative record number	RD
	3.14 cobfa_rewcurr()	Rewrite the current record after a sequential read	SD
	3.15 cobfa_rewkey()	Rewrite the record indicated by the value of the primary key	RD
	3.16 cobfa_rewrec()	Rewrite the record indicated by a relative record number	RD
	3.20 cobfa_wrkey()	Write a record indicated by the value of the primary record key	RD
	3.21 cobfa_wrnext()	Write a record sequentially	S
	3.22 cobfa_wrrec()	Write the record indicated by the relative record number	RD
Status Oriented	3.5 cobfa_errno()	Get the last error number	-
	3.17 cobfa_stat()	Get the last I-O status	-

Access Restrictions

- RD : function that is usable when the open mode is random access or dynamic access
- SD : function that is usable when the open mode is sequential access or dynamic access
- S : function that is usable only when the open mode is sequential access

3.1 cobfa_close()

Close a file.

```
long  cobfa_close  (
    long  fd            /* file descriptor */
);
```

Description

Closes the file indicated by the file descriptor <*fd*>.

Execution Conditions

File Organization	Sequential file	Executable
	Record sequential file	Executable
	Relative file	Executable
	Indexed file	Executable
Open Mode	INPUT mode	Executable
	OUTPUT mode	Executable
	I-O mode	Executable
	EXTEND mode	Executable
Access Mode	Sequential	Executable
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (*)	FA_ENOTOPEN	42	An invalid file descriptor is specified

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.2 cobfa_delcurr()

Delete the current record after a sequentially read (sequential delete).

```
long  cobfa_delcurr  (
    long  fd            /* file descriptor */
);
```

Description

The current record is deleted from the file indicated by the file descriptor <*fd*>.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	Executable (*)
	Indexed file	Executable (*)
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable (*)
	EXTEND mode	-
Access Mode	Sequential	Executable (*)
	Random	-
	Dynamic	-

* : The record must have been positioned by a sequential read.

Other conditions:

The record must have been positioned by a sequential read.

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	49	The specified file is opened in a mode other than I-O mode. Alternatively, an invalid file descriptor is specified.
	FA_EBADACC	90	The file is opened with a file organization or access mode that does not allow this function to be executed.
	FA_ELOCKED	99	The record positioned by sequential read is locked
	FA_ENOCURR	43	Sequential read in the specified file had failed

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.3 cobfa_delkey()

Delete the record indicated by the value of the primary record key (random delete).

```
long  cobfa_delkey  (
    long          fd,        /* file descriptor */
    const  char  *recarea    /* record area     */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record indicated by the value of the primary record key retained in the record area <*recarea*> is deleted.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	49	The specified file is opened in a mode other than I-O mode. Alternatively, an invalid file descriptor is specified.
	FA_EBADACC	90	The file is opened with a file organization or access mode that does not allow this function to be executed.
	FA_ELOCKED	99	The record specified by the value of the primary record key is locked
	FA_ENOREC	23	The record specified by the value of the primary record key does not exist

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.4 cobfa_delrec()

Delete the record indicated by a relative record number (random delete).

```
long  cobfa_delrec  (
    long            fd,      /* file descriptor        */
    unsigned  long  recnum   /* relative record number */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record indicated by a relative record number <*recnum*> is deleted.

Execution Conditions

File Organization	Sequential file	-

	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	49	The specified file is opened in a mode other than I-O mode. Alternatively, an invalid file descriptor is specified.
	FA_EBADACC	90	The file is opened with a file organization or access mode that does not allow this function to be executed.
	FA_ELOCKED	99	The record specified by the value of the relative record key is locked
	FA_ENOREC	23	The record specified by the relative record number does not exist

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.5 cobfa_errno()

Get the last error number.

```
long  cobfa_errno  (
    void            /* no arguments */
);
```

Description

Returns the error number from the most recent API function call.

Execution Conditions

Always accessible.

Return Values

FA_ENOERR	Most recent function call executed successfully
Other than FA_ENOERR	Most recent function call failed. For details on this value, see "5.1 Error Number".

3.6 cobfa_indexinfo()

Get the file attributes or key structure of an indexed file.

```
long  cobfa_indexinfo  (
    long                fd,         /* file descriptor                */
    struct  fa_keydesc  *buffer,    /* acquisition result storage area */
    long                funccode    /* function code                  */
);
```

Description

Information on the indexed file indicated by the file descriptor <*fd*> is set in the acquisition result storage area <*buffer*>. The file be open prior to execution of this function.

Select the type of information to be acquired by specifying it in the function code <*funccode*>.

Retrieving File Attributes

To retrieve the attributes of an indexed file, specify 0 for *funccode*. The *struct fa_dictinfo* type file attribute is stored in the acquisition result storage area. For the *struct fa_dictinfo* type, see "4.3 struct fa_dictinfo".

Retrieving Key Structures

To retrieve the key structure of an indexed file, specify 1 or greater for *funccode*. Keys are numbered sequentially, starting with 1 for the primary key. The *struct fa_keydesc* type is stored in the area pointed to by the acquisition result storage area.

If 1 is specified as the funccode value, the primary record key list is acquired. If 2 or greater is specified, the alternate record key list is acquired. This list order corresponds to the order in which alternate record keys were declared when the indexed file was created. For the *struct fa_keydesc* type, see "4.1 struct fa_keydesc".

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	Executable
	OUTPUT mode	Executable
	I-O mode	Executable
	EXTEND mode	Executable
Access Mode	Sequential	Executable
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all)	FA_ENOTOPEN	90	An invalid file descriptor is specified
	FA_EBADACC	90	The specified file is not an indexed file

3.7 cobfa_open()

Open a file.

```
long  cobfa_open  (
    const  char               *fname,          /* filename        */
    long                       openflgs,       /* open attribute  */
    const  struct  fa_keylist  *keylist,       /* record key list */
    long                       reclen          /* record length   */
);
```

Description

Opens the file indicated by filename <*fname*> based on the information of the open attribute <*openflgs*>, record length <*reclen*>, and record key list <*keylist*>.

The following functions can be used by specifying file name <*fname*>.

- To open a dummy file, append the character string ",DUMMY" to the end of the file name or specify ",DUMMY" as the file name. Refer to "3.25 Dummy File" for details.
- Record sequential files and line sequential files can be accessed faster by specifying an available range. To use High-speed File Processing, append the character string ",BSAM" to the end of the file name. Refer to "3.26 High-speed File Processing" for details.
- A named pipe can be used as a record sequential file and line sequential file. The named pipe is created before the application is executed. The named pipe is specified as cobfa_open file name. Refer to "3.27 Named pipe" for details.

The maximum length of a file name <fname> is specified as a character string of no more than 1021 bytes.

The values of the open attribute *openflgs* have the following eight categories, which are associated by a logical OR. The Open Mode category (a) must be specified. The remaining categories (b) to (h) can be omitted (default: *).

a. Open mode (COBOL syntax)

	FA_INPUT	INPUT mode (OPEN INPUT)
	FA_OUTPUT	OUTPUT mode (OPEN OUTPUT)
	FA_INOUT	I-O mode (OPEN I-O)
	FA_EXTEND	EXTEND mode (OPEN EXTEND)

- If the file is a line sequential file (FA_LSEQFILE), I-O mode (FA_INOUT) cannot be specified as the open mode.

b. File organization (COBOL syntax)

*	FA_SEQFILE	Record sequential file (ORGANIZATION IS SEQUENTIAL)
	FA_LSEQFILE	Line sequential file (ORGANIZATION IS LINE SEQUENTIAL)
	FA_RELFILE	Relative file (ORGANIZATION IS RELATIVE)
	FA_IDXFILE	Indexed file (ORGANIZATION IS INDEXED)

c. Record format (COBOL syntax)

*	FA_FIXLEN	Fixed-length format (RECORD CONTAINS integer CHARACTERS)
	FA_VARLEN	Variable-length format (RECORD IS VARYING IN SIZE)

d. Access mode (COBOL syntax)

*	FA_SEQACC	Sequential access (ACCESS MODE IS SEQUENTIAL)
	FA_RNDACC	Random access (ACCESS MODE IS RANDOM)
	FA_DYNACC	Dynamic access (ACCESS MODE IS DYNAMIC)

- If the file is a line sequential file (FA_LSEQFILE) or record sequential file (FA_SEQFILE), an access mode other than sequential access (FA_SEQACC) cannot be specified.

e. Lock mode (COBOL syntax)

	FA_AUTOLOCK	Automatic lock (LOCK MODE IS AUTOMATIC)
	FA_MANULOCK	Manual lock (LOCK MODE IS MANUAL)
	FA_EXCLLOCK	Exclusive lock (OPEN WITH LOCK / LOCK MODE IS EXCLUSIVE)

- If the open mode is OUTPUT mode (FA_OUTPUT), it is assumed that exclusive lock (FA_EXCLLOCK) is specified as the lock mode.
- If the open mode is INPUT mode (FA_INPUT), the file is opened in shared mode using lock mode as the default. At read, the record lock specification is ignored.
- If the open mode is a mode other than INPUT mode (FA_INPUT), the default lock mode is exclusive lock (FA_EXCLLOCK).
- If the file is a line sequential file (FA_LSEQFILE) or record sequential file (FA_SEQFILE), manual lock (FA_MANULOCK) cannot be specified as the lock mode.

Information

The following describes the relationship between the execution of each API function and exclusive file control.

Exclusive control of files

- Exclusive mode

  When a file is opened in exclusive mode, other users cannot access the file.

- Shared mode

  A file opened in shared mode can be accessed by other users. However, a file that has already been opened in exclusive mode cannot be opened by other users.

The table below shows the state of the file when opened under various combinations of Open mode and Lock mode.

		Open mode			
		FA_INPUT	FA_OUTPUT	FA_INOUT	FA_EXTEND
Lock mode	**None**	shared	exclusive	exclusive	exclusive
	FA_AUTOLOCK			shared	shared
	FA_MANULOCK				
	FA_EXCLLOCK	exclusive		exclusive	exclusive

Exclusive control of records

If a record of a file opened with FA_INOUT in shared mode becomes exclusive via a record lock specification at read time, other users cannot access the record.

Refer to "3.8 cobfa_rdkey()", "3.9 cobfa_rdnext()" and "3.10 cobfa_rdrec()" for record lock specifications at read time.

In the following cases, the exclusive lock on the record is released:

- All record locks released (cobfa_release)
- A file is closed (cobfa_close)

When FA_AUTOLOCK is in effect and one the following processes is executed, the exclusive lock on the record is released:

- Read a record(cobfa_rdkey, cobfa_rdnext, cobfa_rdrec)
- Write a record(cobfa_wrkey, cobfa_wrnext, cobfa_wrrec)
- Delete a record(cobfa_delcurr, cobfa_delkey, cobfa_delrec)
- Rewrite a record(cobfa_rewcurr, cobfa_rewkey, cobfa_rewrec)
- Position to the record(cobfa_stkey, cobfa_strec)

f. Optional file (COBOL syntax)

*	FA_NOTOPT	Non-optional file (SELECT filename)
	FA_OPTIONAL	Optional file (SELECT OPTIONAL filename)

g. Character encoding type

*	FA_ASCII	The file records have character data encoded in ASCII.
	FA_UCS2	The file records have character data encoded in UCS-2. Since the endian mode depends on the system, it is set to little-endian.
	FA_UCS2BE	The file records have character data encoded in UCS-2 (big-endian).
	FA_UCS2LE	The file records have character data encoded in UCS-2 (little-endian).
	FA_UTF8	The file records have character data encoded in UTF-8.
	FA_UTF32	The file records have character data encoded in UTF-32. Since the endian mode depends on the system, it is set to little-endian.
	FA_UTF32BE	The file records have character data encoded in UTF-32 (big-endian).
	FA_UTF32LE	The file records have character data encoded in UTF-32 (little-endian).

- The character encoding type specifies the encoding type of the character data of line sequential file records. For line sequential files, the file structure depends on the encoding type. Therefore, the character encoding type is significant when using line sequential files.
- The character encoding type can be specified only when the file organization is line sequential file (FA_LSEQFILE). For other file organizations, the file structure does not depend on the encoding type. Therefore, the character encoding type cannot be specified.
- For line sequential files to be handled by COBOL applications operating in Unicode mode, if the record data items are national items, specify FA_UCS2, FA_UCS2BE, or FA_UCS2LE. If the record data items are not national items, specify FA_UTF8.
- For line sequential files to be handled by COBOL applications operating in a mode other than Unicode, specify FA_ASCII.

h. Key part flag usage specification

	FA_USEKPFLAGS	Enables the specification value of the kp_flags member of struct fa_keypart type structure.

- Specify key part flag usage to enable the specification value of member kp_flags of struct fa_keypart type structure included in record key list keylist. See "4.1 struct fa_keydesc" for details about the struct fa_keypart type.
- Key part flag usage can be specified only when the file organization is indexed file (FA_IDXFILE).

- Key part flag usage must be specified to use indexed files handled by COBOL applications whose operating mode is Unicode.
- If the key part flag usage specification is omitted, the specification value of member kp_flags will be ignored.

The record length <*reclen*> is treated as the fixed-record length when the record format is the fixed-length format (FA_FIXLEN). When the record format is the variable-length format (FA_VARLEN), the record length is treated as the maximum record length. The record length must not exceed FA_NRECSIZE (typically 32760).

The record key list <*keylist*> is significant only if the file is an indexed file (FA_IDXFILE). In this case, <*keylist*> is valid as the list of the primary record key and alternate record key(s) of the file being opened. For the *struct fa_keylist* type, see "4.2 struct fa_keylist".

If the NULL pointer is specified for <*keylist*> to open an indexed file, this function automatically identifies the key list structure, record format, and record length of an existing file, and opens the file. In this case, the specified record format and record length are ignored.

Execution Conditions

File Organization	Sequential file	Executable
	Record sequential file	Executable
	Relative file	Executable
	Indexed file	Executable
Open Mode	INPUT mode	Executable
	OUTPUT mode	Executable
	I-O mode	Executable
	EXTEND mode	Executable
Access Mode	Sequential	Executable
	Random	Executable
	Dynamic	Executable

Return Values

1 or greater	Successful. Function executed successfully. A code indicating the status may have been set in the I-O status This return value is the value of the file descriptor of the file that has been opened successfully. Note, however, that this file descriptor is not the value of the file handle returned by the operating system when the file is opened.
-1	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions.

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
	FA_ENOERR	5	No optional file exists but the file is virtually opened. Alternatively, a new file is created when the open mode is not INPUT mode (FA_INPUT).
Failure (not all) (*)	FA_EBADACC	90	The specified combination of file organization, access mode, and open mode cannot be executed
	FA_EFNAME	35	The file does not exist
	FA_EFLOCKED	93	The file has been opened exclusively
	FA_EFNAME	90	The filename is incorrect or file access failed
	FA_EFNAME	91	The filename is not specified

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
	FA_EBADFLAG	39	The specified attribute such as file organization or record format is different from the organization of the existing file
	FA_EBADKEY	39	Information on the specified record key is different from the key list of the existing indexed file
	FA_EBADKEY	90	Information on the specified record key is incorrect
	FA_EBADLENG	39	The specified record length is different from the record length of the existing file
	FA_EBADFILE	90	The specified file is unreadable

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.8 cobfa_rdkey()

Reads the record indicated by the value of an arbitrary key (random read).

```
long  cobfa_rdkey  (
    long                        fd,        /* file descriptor      */
    long                        readflgs,  /* read attribute       */
    char                       *recarea,   /* record area          */
    const  struct  fa_keydesc  *keydesc,   /* Record key structure */
    long                        keynum     /* record key number    */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record indicated by the value of the arbitrary key retained in the record area <*recarea*> is read. The record is stored in the record area <*recarea*>.

The read attribute <*readflgs*> includes the following two categories, which are associated by a logical OR. If omitted, default values are assigned (default: *).

- Read mode

*	FA_EQUAL	Reads the record corresponding to the specified record key value.

- Record lock flag (COBOL syntax)

	FA_LOCK	Reads with lock. (READ WITH LOCK)
	FA_NOLOCK	Reads with no lock. (READ WITH NO LOCK)

  - The default of record lock flag is affected by lock mode specification when the file is opened. If the lock mode is automatic lock (FA_AUTOLOCK), the default is READ WITH LOCK (FA_LOCK). Otherwise, the default is READ WITH NO LOCK (FA_NOLOCK).

An arbitrary record key is specified with record key structure specification <*keydesc*>. For the *struct fa_keydesc* type, see "4.1 struct fa_keydesc".

If NULL is specified for this record key structure specification, record key number specification <*keynum*> is valid as an arbitrary record key. To use the primary record key, specify 1 for the record key number. To use an alternate record key, specify 2 or greater for the record key number. This value corresponds to the list of the order in which alternate record keys were declared when the indexed file was created. For the first alternate record key, specify 2. For the second alternate record key, specify 3. For the following alternate record keys, specify the corresponding numbers in the same way.

Execution Conditions

File Organization	Sequential file	-

	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	Executable
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully. A code indicating the status may be set in the I-O status.
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
	FA_ENOERR	2	The value of the reference key in the record that is read is equivalent to the value of the reference key in the next record
Failure (not all) (*)	FA_ENOTOPEN	47	The specified file is opened in a mode other than INPUT mode and I-O mode. Alternatively, an invalid file descriptor is specified
	FA_EBADACC	90	The file is opened with the file organization or access mode that does not allow this function to be executed
	FA_EBADFLAG	90	The read mode not allowing execution of this function is specified. Alternatively, another flag specification is incorrect
	FA_ENOREC	23	The record indicated by an arbitrary record key value does not exist
	FA_EBADKEY	90	The specified record key structure or record key number is incorrect or does not exist
	FA_ELOCKED	99	The record specified with the primary record key value has been locked

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.9 cobfa_rdnext()

Reads a record sequentially (sequential read).

```
long  cobfa_rdnext  (
    long   fd,              /* file descriptor */
    long   readflgs,        /* read attribute  */
    char  *recarea          /* record area     */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record immediately after (or before) the current record is read and stored in the record area <*recarea*>.

The read attribute <*readflgs*> includes the following two categories, which are associated by an OR. If omitted, default values are assigned (default: *).

- Read mode (COBOL syntax)

*	FA_NEXT	Reads the next record. (READ NEXT RECORD)
	FA_PREV	Reads the previous record. (READ PREVIOUS RECORD)

  - If the file is a line sequential file (FA_LSEQFILE) or record sequential file (FA_SEQFILE), previous record (FA_PREV) cannot be specified for the read mode.

- Record lock flag (COBOL syntax)

	FA_LOCK	Reads with lock. (READ WITH LOCK)
	FA_NOLOCK	Reads with no lock. (READ WITH NO LOCK)

  - The default of the record lock flag depends on the lock mode specified when the file is opened. If the lock mode is automatic lock (FA_AUTOLOCK), the default is READ WITH LOCK (FA_LOCK). Otherwise, the default is READ WITH NO LOCK (FA_NOLOCK).

Execution Conditions

File Organization	Sequential file	Executable (*)
	Record sequential file	Executable (*)
	Relative file	Executable (*)
	Indexed file	Executable (*)
Open Mode	INPUT mode	Executable (*)
	OUTPUT mode	-
	I-O mode	Executable (*)
	EXTEND mode	-
Access Mode	Sequential	Executable (*)
	Random	-
	Dynamic	Executable (*)

* : File positioning must not be optional.

Return Values

0: Successful	Function executed successfully. A code indicating the status may be set in the I-O status.
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
	FA_ENOERR	2	The value of the reference key in the record that is read is equivalent to the value of the reference key in the next record

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Failure (not all) (*)	FA_ENOTOPEN	47	The specified file is opened in a mode other than INPUT mode and I-O mode. Alternatively, an invalid file descriptor is specified
	FA_EBADACC	90	The file is opened with the access mode that does not allow this function to be executed
	FA_EENDFILE	10	A file ending condition occurred.
	FA_EBADFLAG	90	A read mode that does not allow execution of this function is specified. Alternatively, another flag specification is incorrect
	FA_ENOCURR	46	Positioning to the record was undefined
	FA_ELOCKED	99	The record to be positioned by sequential read has been locked

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.10 cobfa_rdrec()

Reads the record indicated by a relative record number (random read).

```
long  cobfa_rdrec  (
    long            fd,        /* file descriptor        */
    long            readflgs,  /* read attribute         */
    char           *recarea,   /* record area            */
    unsigned  long  recnum     /* relative record number */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record specified by the relative record number <*recnum*> is read and stored in the record area <*recarea*>.

The read attribute <*readflgs*> includes the following two categories, which are associated by an OR. If omitted, default values are assigned (default: *).

- Read mode (COBOL syntax)

*	FA_EQUAL	Reads the record with the specified relative record number.

- Record lock flag (COBOL syntax)

	FA_LOCK	Reads with lock. (READ WITH LOCK)
	FA_NOLOCK	Reads with no lock. (READ WITH NO LOCK)

  - The record lock flag default is affected by the lock mode specified when the file is opened. If the lock mode is automatic lock (FA_LOCK), the default is READ WITH LOCK (FA_LOCK). Otherwise, the default is READ WITH NO LOCK (FA_NOLOCK).

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	Executable
	Indexed file	-

Open Mode	INPUT mode	Executable
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	47	The specified file is opened in a mode other than INPUT mode and I-O mode. Alternatively, an invalid file descriptor is specified
	FA_EBADACC	90	The file is opened with the file organization or access mode that does not allow this function to be executed
	FA_EBADFLAG	90	A read mode not allowing execution of this function is specified. Alternatively, another flag specification is incorrect
	FA_ENOREC	23	The record indicated by the relative record number does not exist
	FA_ELOCKED	99	The record specified by the relative record number has been locked

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.11 cobfa_reclen()

Returns the length of the current record.

```
long  cobfa_reclen  (
    void                /* no arguments */
);
```

Description

When handling files with variable length records, returns the length of the record just read. Unpredictable values are returned if the read function failed or there have been intervening function calls. This function is valid after calls to:

- cobfa_rdnext ()
- cobfa_rdrec ()
- cobfa_rdkey ()

Execution Conditions

Always accessible.

Return Values

Length (in bytes) of the current record.

3.12 cobfa_recnum()

Returns the relative record number of the current record.

```
unsigned  long  cobfa_recnum  (
    void                /* no arguments */
);
```

Description

When handling relative files, returns the record number of the current record. Unpredictable values are returned if the read function failed or there have been intervening function calls.

This function is valid after calls to:

- cobfa_rdnext ()
- cobfa_rdrec ()

Execution Conditions

Always accessible.

Return Values

Relative record number of the record currently positioned in a relative file.

3.13 cobfa_release()

Releases all record locks of the specified file.

```
long  cobfa_release  (
    long  fd                /* file descriptor */
);
```

Description

In the file indicated by the file descriptor <*fd*>, all record locks are released.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	Executable
	Relative file	Executable
	Indexed file	Executable
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	Executable

	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	90	An invalid file descriptor that was not acquired when the file was opened is specified

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.14 cobfa_rewcurr()

Rewrite the current record after a sequentially read (sequential rewrite).

```
long  cobfa_rewcurr  (
    long          fd,          /* file descriptor         */
    const  char  *recarea,     /* record area             */
    long          reclen       /* rewritten record length */
);
```

Description

The current record is rewritten with the contents of the record area <*recarea*> in the file indicated by the file descriptor <*fd*>.

The rewritten record length <*reclen*> specified is valid only if the file is in the variable-length format. If the file is a record sequential file (FA_SEQFILE), the original record length cannot be changed.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	Executable (*)
	Relative file	Executable (*)
	Indexed file	Executable (*)
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable (*)
	EXTEND mode	-
Access Mode	Sequential	Executable (*)
	Random	-
	Dynamic	-

* : The record must have been positioned by a sequential read.

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_EDUPL	22	Although the value of the primary record key or alternate record key of the record to be rewritten already exists in the file, duplication of the primary record key or alternate record key is not permitted
	FA_EBADLENG	44	The specified value of the rewritten record length is out of the permissible range

* : See "3.2 cobfa_delcurr()"for additional values.

3.15 cobfa_rewkey()

Rewrite the record indicated by the value of the primary record key (random rewrite).

```
long  cobfa_rewkey  (
    long          fd,          /* file descriptor         */
    const  char  *recarea,     /* record area             */
    long          reclen       /* rewritten record length */
);
```

Description

The record specified by the value of the primary record key held in the record area <*recarea*> in the file indicated by the file descriptor <*fd*> is rewritten with the contents of the record area <*recarea*>.

The rewritten record length <*reclen*> specified is valid only if the file is in the variable-length format.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_EDUPL	22	An attempt was made to rewrite with the record contents having an existing record key value to the record key not permitting duplication
	FA_EBADLENG	44	The specified value of the rewritten record length is out of the permissible range

* : See "3.3 cobfa_delkey()"for additional values.

3.16 cobfa_rewrec()

Rewrite the record indicated by a relative record number (random rewrite).

```
long  cobfa_rewrec  (
    long            fd,         /* file descriptor         */
    const  char    *recarea,    /* record area             */
    long            reclen,     /* rewritten record length */
    unsigned  long  recnum      /* relative record number  */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record indicated by a relative record number <*recnum*> is rewritten by the contents of the record area <*recarea*>.

The rewritten record length <*reclen*> specified is valid only if the file is in the variable-length format.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	Executable
	Indexed file	-
Open Mode	INPUT mode	-
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully

-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_EBADLENG	44	The specified value of the rewritten record length is out of the permissible range

* : See "3.4 cobfa_delrec()" for additional values.

3.17 cobfa_stat()

Get the last I-O status.

```
long  cobfa_stat  (
    void               /* no arguments */
);
```

Description

Returns the I-O status from the most recent API function call.

Execution Conditions

Always accessible.

Return Values

Refer to "I-O Status List" in "NetCOBOL User's Guide."

3.18 cobfa_stkey()

Positions to the record indicated by the value of an arbitrary key.

```
long  cobfa_stkey  (
    long                        fd,       /* file descriptor                    */
    long                        stflgs,   /* positioning attribute              */
    const  char                *recarea,  /* record area                        */
    const  struct  fa_keydesc  *keydesc,  /* record key structure specification */
    long                        keynum,   /* record key number specification    */
    long                        keyleng   /* valid key length                   */
);
```

Description

In the file indicated by the file descriptor <*fd*>, positions to the record matching the value of an arbitrary key held in the record area <*recarea*> and the requested positioning mode <*stflgs*>. Successful repositioning establishes the reference key for subsequent sequential read.

The positioning mode <*stflgs*> includes the following two categories, which are associated by a logical OR. If omitted, default values are assigned (default: *).

- Positioning mode (COBOL syntax)

	FA_FIRST	First record (START FIRST RECORD)
*	FA_EQUAL	Record equivalent to the record key value. (START KEY IS =)

	FA_GREAT	Record greater than the record key value. (START KEY IS >)
	FA_GTEQ	Record equivalent to or greater than the record key value. (START KEY IS >=)
	FA_LESS	Record smaller than the record key value. (START KEY IS <)
	FA_LTEQ	Record equivalent to or smaller than the record key value. (START KEY IS <=)

- Reversed-order read flag (COBOL syntax)

*	-	Reads in logically order upon sequential read.
	FA_REVORD	Reads in logically reversed order upon sequential read. (START WITH REVERSED ORDER)

  - The default of the reversed-order read flag is disabled.
  - If the positioning mode exceeds the record key value (FA_GREAT) or is equivalent to or greater than the record key value (FA_GTEQ), the reversed-order read flag (FA_REVORD) cannot be specified.
  - The reversed-order flag specified is disabled by any random read, positioning, file close, or end of file ending condition.

An arbitrary record key is specified by using record key structure specification <keydesc>. For the struct fa_keydesc type, see "4.1 struct fa_keydesc".

If NULL is specified for this record key structure specification, record key number specification <keynum> is valid as an arbitrary record key specified. To specify the primary record key, specify 1 for the record key number. To specify the alternate record key, specify 2 or greater for the record key number. This value corresponds to the list of the order in which alternate record keys were declared when the indexed file was created. For the first alternate record key, specify 2. For the second alternate record key, specify 3. For the following alternate record keys, specify the corresponding numbers in the same way.

The valid key length <keyleng> is used to shorten the length of the valid reference key. In most cases, in which the key length is not shortened, 0 is specified for <keyleng>. The entire arbitrary record key is valid as a reference key. To shorten the valid key length, specify 1 or a greater value. This value indicates key length from the top of the key.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	Executable
	OUTPUT mode	-
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	Executable
	Random	-
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	47	The specified file is opened in a mode other than INPUT mode and I-O mode. Alternatively, an invalid file descriptor is specified
	FA_EBADACC	90	The file is opened with the file organization or access mode that does not allow this function to be executed
	FA_EBADFLAG	90	The positioning mode not allowing execution of this function is specified. Alternatively, another flag specification is incorrect
	FA_ENOREC	23	The record corresponding to the specified condition does not exist
	FA_EBADKEY	90	The specified record key structure or record key number does not exist or is not correct

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.19 cobfa_strec()

Position to the record indicated by a relative record number.

```
long  cobfa_strec  (
    long            fd,        /* file descriptor        */
    long            stflgs,    /* positioning attribute  */
    unsigned  long  recnum     /* relative record number */
);
```

Description

In the file indicated by the file descriptor <*fd*>, positions to the record relating to the value of the relative record number <*recnum*>.

The positioning attribute <*stflgs*> includes the following category. If omitted, default values are assigned (default: *).

- Positioning mode (COBOL syntax)

*	FA_EQUAL	Record equivalent to <recnum> (START KEY IS =)
	FA_GREAT	Record greater than <recnum> (START KEY IS >)
	FA_GTEQ	Record equivalent to or greater than <recnum> (START KEY IS >=)
	FA_LESS	Record smaller than <recnum> (START KEY IS <)
	FA_LTEQ	Record equivalent to or smaller than <recnum> (START KEY IS <=)

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	Executable
	Indexed file	-
Open Mode	INPUT mode	Executable
	OUTPUT mode	-
	I-O mode	Executable

	EXTEND mode	-
Access Mode	Sequential	Executable
	Random	-
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_ENOTOPEN	47	The specified file is opened in a mode other than INPUT mode and I-O mode. Alternatively, an invalid file descriptor is specified
	FA_EBADACC	90	The file is opened with the file organization or access mode that does not allow this function to be executed
	FA_EBADFLAG	90	A positioning mode not allowing execution of this function is specified. Alternatively, another flag specification is incorrect
	FA_ENOREC	23	The record corresponding to the specified condition does not exist.

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.20 cobfa_wrkey()

Write a record specified by the primary record key value (random write).

```
long  cobfa_wrkey  (
    long          fd,          /* file descriptor       */
    const  char  *recarea,     /* record area           */
    long          reclen       /* written record length */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record indicated by the value of the primary record key held in the record area <*recarea*> is written according to the contents of the record area.

The written record length <*reclen*> is valid only if the file is in the variable-length format.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	-
	Indexed file	Executable
Open Mode	INPUT mode	-

	OUTPUT mode	Executable
	I-O mode	Executable
	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_EDUPL	22	Since duplication of the primary record key is not permitted, this function cannot be executed
	FA_EBADLENG	44	The specified written record length value is out of the permissible range
	FA_ENOTOPEN	48	The specified file is opened in INPUT mode. Alternatively, an invalid file descriptor is specified

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.21 cobfa_wrnext()

Write a record sequentially (sequential write).

```
long  cobfa_wrnext  (
    long          fd,          /* file descriptor       */
    const  char  *recarea,     /* record area           */
    long          reclen       /* written record length */
);
```

Description

In the file indicated by the file descriptor <*fd*>, a record is written sequentially according to the contents in the record area <*recarea*>.

The written record length <*reclen*> is valid only if the file is in the variable-length format.

Execution Conditions

File Organization	Sequential file	Executable
	Record sequential file	Executable
	Relative file	Executable
	Indexed file	Executable
Open Mode	INPUT mode	-
	OUTPUT mode	Executable
	I-O mode	-

	EXTEND mode	Executable
Access Mode	Sequential	Executable
	Random	-
	Dynamic	-

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_EDUPL	22	Since duplication of the primary record key is not permitted, this function cannot be executed
	FA_EBADLENG	44	The specified written record length value is out of the permissible range
	FA_ENOTOPEN	48	The specified file is opened in INPUT mode. Alternatively, an invalid file descriptor is specified

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.22 cobfa_wrrec()

Write a record specified by the relative record number (random write).

```
long  cobfa_wrrec  (
    long            fd,        /* file descriptor        */
    const  char    *recarea,   /* record area            */
    long            reclen,    /* written record length  */
    unsigned  long  recnum     /* relative record number */
);
```

Description

In the file indicated by the file descriptor <*fd*>, the record specified by a relative record number <*recnum*> is written according to the contents of the record area <*recarea*>.

The written record length <*reclen*> is valid only if the file is in the variable-length format.

Execution Conditions

File Organization	Sequential file	-
	Record sequential file	-
	Relative file	Executable
	Indexed file	-
Open Mode	INPUT mode	-
	OUTPUT mode	Executable
	I-O mode	Executable

	EXTEND mode	-
Access Mode	Sequential	-
	Random	Executable
	Dynamic	Executable

Return Values

0: Successful	Function executed successfully
-1: Failed	Function failed. To get extended error information, call the cobfa_errno or cobfa_stat functions

Generated Status

Return value of cobfa_errno ()		Return value of cobfa_stat ()	
Successful	FA_ENOERR	0	Function executed successfully
Failure (not all) (*)	FA_EDUPL	22	An attempt was made to write a record by using a relative record number that already exists
	FA_EBADLENG	44	The specified written record length value is out of the permissible range
	FA_ENOTOPEN	48	The specified is opened in INPUT mode. Alternatively, an invalid file descriptor is specified

* : Typical status values are described above. For other values, see "5.1 Error Number" and "5.2 I-O Status".

3.23 LOCK_cobfa()

Set an exclusive lock.

```
long  LOCK_cobfa (
    const unsigned long   *timeout,      /* wait time  */
    unsigned long         *errcode       /* error code */
);
```

Description

In a multithread environment, exclusive control is required to prevent multiple threads from accessing a file simultaneously in the COBOL runtime system.

This function sets the exclusive lock of a COBOL file accessed by the COBOL runtime system to prevent other threads from accessing the COBOL file. When a file is accessed in a multithread environment, this function is used to prevent conflicts.

Specify the number of seconds to wait in *<*timeout>*. If the exclusive lock of a file has been set by another thread and does not open the file before this function is called by another thread *<*timeout>*, execution of the called function fails. If NULL is specified for *<timeout>*, the wait time is unlimited.

When the return value of this function is -1, the system error code is set in error code *<*errcode>*. If NULL has been specified for *<errcode>*, no value is returned for *<*errcode>*. The value is unpredictable when the return value of this function is not -1.

Execution Conditions

Always accessible.

Return Values.

Condition	Return value	Meaning
Successful	0	Exclusive lock of a COBOL file is successful
Failed	-1	System error occurred. Windows system error code is set in <*errcode>.
	-2	The wait time specified in "<*timeout>" has been exceeded. Access is not locked

How to Use

Execute this function before executing an API function having an I-O function or an API function for obtaining file information. After executing either of these API functions, execute the API function for obtaining a status report, if necessary. Then execute "3.24 UNLOCK_cobfa()" to release the exclusive lock.

While an exclusive lock is held other threads may rewrite the status value. Consequently, to ensure accuracy of the status value, "cobfa_stat" must be executed immediately following any I-O related API function.

An example of exclusive control for each I-O is provided below.

```
long  ret, fd, eno, stat, recnum;
char buff[10];

ret = LOCK_cobfa ( NULL, NULL ); /* Set the exclusive lock of a file */
if ( ret < 0 ) { .... }          /* Error processing */
fd = cobfa_open ( "file.rel", FA_INPUT | FA_RELFILE | FA_FIXLEN, NULL, 10 );
stat = cobfa_stat ( );
eno  = cobfa_errno ( );
UNLOCK_cobfa ( NULL );           /* Open the exclusive lock of a file */
if ( eno == FA_EFNAME ) {
    printf ( "errno: %d, stat: %d\n", eno, stat );
    ...
}

....                             /* Perform processing not related to the input-output file */

ret = LOCK_cobfa ( NULL, NULL ); /* Lock file access exclusively. */
if ( ret < 0 ) { ....            /* Error processing */ }
cobfa_rdnext ( fd, FA_NEXT, buff );
recnum  = cobfa_recnum ( );
UNLOCK_cobfa ( NULL );           /* Perform processing not related to the input-output file */
num = recnum;

....                             /* Set the exclusive lock of the file */

ret = LOCK_cobfa ( NULL, NULL ); /* Lock file access exclusively */
if ( ret < 0 ) { .... }          /* Error processing */
cobfa_close ( fd );
UNLOCK_cobfa ( NULL );           /* Open the exclusive lock of the file */
```

A COBOL file can be controlled exclusively from the start to the end of file access. Arbitrary range extension of exclusive control is possible.

An example of continuous exclusive control is provided below.

```
long  ret, fd;
char buffer[10];

ret = LOCK_cobfa ( NULL, NULL );     /* Set the exclusive lock of a file */
if ( ret < 0 ) { .... }              /* Error processing */
fd = cobfa_open ( "file.seq", FA_OUTPUT | FA_SEQFILE | FA_FIXLEN, NULL, 10 );

cobfa_wrnext ( fd, buffer, 10 );
....                                 /* Perform input-output operation for files */
```

```
cobfa_close ( fd );
UNLOCK_cobfa ( NULL );              /* Open the exclusive lock of a file */
```

3.24 UNLOCK_cobfa()

Release an exclusive lock.

```
long  UNLOCK_cobfa (
    unsigned long  *errcode              /* error code */
);
```

Description

Release the exclusive lock of the file accessed by the COBOL runtime system.

When the return value of this function is -1, the system error code is set in error code *<*errcode>*. If NULL has been specified for *<errcode>*, no value is returned for *<*errcode>*. The value is unpredictable when the return value of this function is not -1.

Execution Conditions

Always accessible.

Return Values

0: Successful	Exclusive unlock of a COBOL file is successful
-1: Failed	System error occurred. Windows system error code is set in <*errcode>.

3.25 Dummy File

When a dummy file is specified in an API function, a physical file is not actually created. A dummy file is useful when an output file is not needed, or when no input file is available during development. For example, a dummy file can be created in place of a physical log file when an error occurs. And when a program is being developed and no input file is available, a dummy file can be used for testing purposes.

Dummy file is convenient to use in the following cases:

- When the output file is not required

  The log file is only required when an error occurs. During normal operation, using the log file as dummy file, you can suppress the output of the file. When an error occurs, remove the specification of the dummy file to output the log file.

- During development, when there is no input files

  You can test using an empty file. By using the input file as dummy files, you can test it without using an empty file.

How to use

",DUMMY" is appended to the file name specified for the cobfa_open() function, or the file name is omitted and only ",DUMMY" is specified.

```
fname = "[file-name],DUMMY";
```

Note

- Place a comma (,) in front of the character string "DUMMY", otherwise the character string "DUMMY" will be interpreted as a file name.
- Dummy file operation is the same regardless of whether the file name is specified. Even if the specified file exists, no operation is performed on the file.

Dummy File Operation

The operation of the dummy file for each function is as follows:

Function	Return Value	Condition	Error Number	I-O status
cobfa_open()	1 or greater	Successful	FA_ENOERR	0
cobfa_close()	0	Successful	FA_ENOERR	0
cobfa_wrnext()				
cobfa_wrrec()				
cobfa_wrkey()				
cobfa_strec()				
cobfa_stkey()				
cobfa_release()				
cobfa_rdnext()	-1	Failed	FA_EENDFILE	10
cobfa_rdrec()	-1	Failed	FA_ENOREC	23
cobfa_rdkey()				
cobfa_delcurr()	-1	Failed	FA_ENOCURR	43
cobfa_rewcurr()				
cobfa_delrec()	-1	Failed	FA_ENOREC	23
cobfa_delkey()				
cobfa_rewrec()				
cobfa_rewkey()				

Note

In file access routines, the specification of the record key can be omitted by opening the index file. In this case, it opens recognizing the index composition, the record format, and the record length of the existing file.

When the dummy file function is specified, the specification of the record key can be similarly omitted. However, the cobfa_indexinfo() to acquire the file information fails. The error number goes to FA_EUNDEFKEY, and the I-O status becomes 90.

3.26 High-speed File Processing

Record sequential files and line sequential files can be accessed faster by specifying an available range. This function is effective in the following API functions:

- cobfa_wrnext()
- cobfa_rdnext()

How to use

Append ",BSAM" to the file name specified for the cobfa_open() function.

```
fname = "file-name,BSAM";
```

Note

- This is effective when the file organization is record sequential (FA_SEQFILE) or line sequential (FA_LSEQFILE). For file organizations other than these, an error occurs during execution of cobfa_open().

- Records cannot be updated. If I-O mode (FA_INOUT) was specified as the open mode, an error occurs during the execution of cobfa_open().
- If files are shared and you want to permit file sharing among different processes, all files must be in shared mode and opened with INPUT specified. Operation is not guaranteed if a file is opened without specifying INPUT. File sharing is not permitted in the same process. Operation is not guaranteed if a file is shared within the same process.
- If a record read from a line sequential file includes a tab, the tab code is not replaced by a blank. If control characters 0x0C (page feed), 0x0D (return), or 0x1A (data end symbol) are included, the control characters are not handled as a record delimiter and a file terminator.

3.27 Named pipe

A named pipe can be used as a record sequential file and line sequential file. When data is received and passed between applications, a named pipe can be used instead of an intermediate file. Applications can be operated in parallel when a named pipe is used.

Using this function

The named pipe is created before the application is executed. The named pipe is specified as cobfa_open file name.

The named pipe is made by the mkfifo system command.

Functional range

Table 3.1 Indicates where a named pipe can be used.

File Types	Record sequential files Line sequential files	
Record Format	Fixed length format Variable length format	
File control entry	SELECT clause	File identifier (*1)
	LOCK MODE clause	Ignored.
Input-Output statement	OPEN statement	OUTPUT/INPUT mode (*2) WITH LOCK specification is ignored.
	READ statement	WITH LOCK/WITH NOLOCK specification is ignored.
	WRITE statement	
	REWRITE statement	Cannot be specified.
	UNLOCK statement	Ignored.

*1: File-identifier literal, data names and DISK cannot be specified.

*2: OPEN mode I-O cannot be specified.

- This is effective when the file organization is record sequential (FA_SEQFILE) or line sequential (FA_LSEQFILE). For file organizations other than these, an error occurs during execution of cobfa_open().
- Records cannot be updated. If I-O mode (FA_INOUT) was specified as the open mode, an error occurs during the execution of cobfa_open().
- If a record read from a line sequential file includes a tab, the tab code is not replaced by a blank. If control characters 0x0C (page feed), 0x0D (return), or 0x1A (data end symbol) are included, the control characters are not handled as a record delimiter and a file terminator.

- When writing and reading are done by one pipe name, it should be couple 1. The pipe breakdown might occur when the written open processing is executed two or more times. Also, when the read open processing is executed two or more times, the record might not be able to be read correctly.

3.28 Immediate Reflection of Written Contents on Close

Contents written can be ensure reflected when the file is closed (cobfa_close()).

Using this function

Set the environment variable CBR_CLOSE_SYNC to "yes" at runtime.

```
$ CBR_CLOSE_SYNC=yes; export CBR_CLOSE_SYNC
```

Note

This function issues an instruction to write the contents of the buffer managed by the OS to the disk when the file is closed (cobfa_close()).

Therefore, using this feature will result in performance degradation due to OS buffer conditions.

Chapter 4 API Structures

This chapter explains the data structures used by the API functions.

4.1 struct fa_keydesc

For the *cobfa_rdkey()* function and *cobfa_stkey()* function, arbitrary record key selection can be specified by using the *struct fa_keydesc* type. With the *cobfa_indexinfo()* function, the arbitrary record key in the indexed file that has been opened can be determined.

The values to set for the *struct fa_keydesc* type members that provide the record key are explained below.

```
#define  FA_NPARTS    254u                    /* max number of key parts     */
struct  fa_keydesc  {
    long   k_flags;                          /* flags (duplicatable or not) */
    long   k_nparts;                         /* number of parts in key      */
    struct  fa_keypart  k_part  [FA_NPARTS];  /* each key part               */
};
```

- Set either value for <*k_flags*> indicating the record key attribute:
  - FA_DUPS: This record key can be duplicated.
  - FA_NODUPS: This record key does not permit duplication.
- Set the number of parts (key parts) in the record key for <*k_nparts*>. The minimum value is 1 and the maximum value is FA_NPARTS (254).
- <*k_part*> provides information on each key part declared in the *struct fa_keypart* type as shown below.

```
#define  FA_NRECSIZE     32760u     /* max number of bytes in a record */
#define  FA_NKEYSIZE     254u       /* max number of bytes in a key    */
struct  fa_keypart  {
    short  kp_start;                /* starting byte of key part       */
    short  kp_leng;                 /* length in bytes                 */
    long   kp_flags;                /* flags (UCS-2 key part or not)   */
};
```

  - Set the 0-based byte displacement from the record start <*kp_start*>. The maximum value of this displacement is FA_NRECSIZE - 1 (typically 32759).
  - Set the key part length for <kp_*leng*>. The minimum value is 1 and the upper limit is FA_NKEYSIZE (typically 254). The sum of displacement and length must not exceed FA_NRECSIZE (typically 32760).
  - Set the information of one of the following categories for <*kp_flags*>. The value set for this member is valid only when key part flag usage (FA_USEKPFLAGS) has been specified for the open attribute argument of file open (cobfa_open() function).

Key part code system type

- FA_UCS2KPCODE: The key part code system type is UCS-2 (little-endian).
- FA_UTF32KPCODE: The key part code system type is UTF-32 (little-endian).
- FA_ANYKPCODE: The key part code system type is other than UCS-2 (little-endian).

For indexed files handled by COBOL applications whose operating mode is Unicode, specify FA_UCS2KPCODE if the key part is a national item of UCS-2 (little-endian), specify FA_UTF32KPCODE if the key part is a national item of UTF-32 (little-endian). If the key part is not a national item of UCS-2 (little-endian), UTF-32 (little-endian) or the operating mode is not Unicode, specify FA_ANYKPCODE.

To reference kp_flags when obtaining the record key organization (cobfa_indexinfo() function), use the mask value FA_KPCODEMASK to fetch the code system type of the key part.

Example

FA_KPCODEMASK use example

```
#include "cobfa.h"
#define GET_PRIM_KEY  1
struct fa_keydesc keydesc1;
  :
ret = cobfa_indexinfo ( fd, &keydesc1, GET_PRIM_KEY);
for (i = 0; i < keydesc1.k_nparts; i ++) {
    :
  switch ( keydesc1.k_part[i].kp_flags & FA_KPCODEMASK ) {
    case FA_UCS2KPCODE:
       :
      break;
    case FA_ANYKPCODE:
       :
      break;
  }
    :
}
```

Practical examples

Example

Example 1

- The primary record key can be duplicated. The number of key parts is 2.
- The first part is at the fifth byte counted from the first byte and its length is 3.
- The second part is at the eleventh byte counted from the first byte and its length is 5.

```
Byte:    | 1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16
Offset:  |0--|1--|2--|3--|4--|5--|6--|7--|8--|9--|10-|11-|12-|13-|14-|15-|
         |=============================================================....
                         |---+---+---|           |---+---+---+---+---|
                          first part              second part
```

COBOL program example

```
001000 environment division.
001100 configuration section.
001200 input-output section.
001300 file-control.
001400     select FILENAME-1 assign to SYS006
001500     organization is indexed
001600     record key is part-1 part-2 with duplicates.
   :
002000 data division.
002100 file section.
002200 fd FILENAME-1.
002300   01 record-1.
002400     02 filler pic x(4).
002500     02 part-1 pic x(3).
002400     02 filler pic x(3).
002500     02 part-2 pic x(5).
002600     02 filler pic x(...
```

C program example

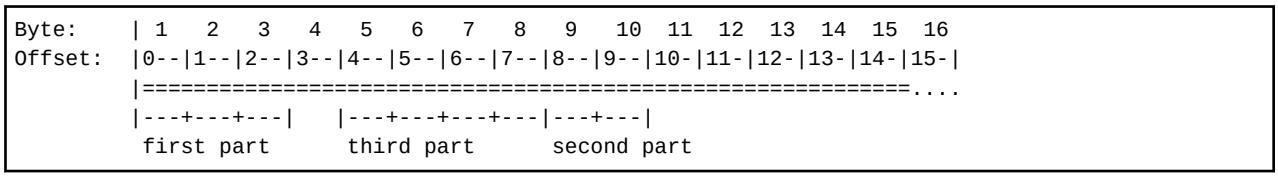
```
#include "cobfa.h"

struct fa_keydesc keydesc1;

keydesc1.k_flags  = FA_DUPS;
keydesc1.k_nparts = 2;                        /* number of key parts: 2 */
keydesc1.k_part[0].kp_start =  4;             /* part_1:  5 - 1 ==  4   */
keydesc1.k_part[0].kp_leng  =  3;
keydesc1.k_part[0].kp_flags = FA_ANYKPCODE;
keydesc1.k_part[1].kp_start = 10;             /* part_2: 11 - 1 == 10   */
keydesc1.k_part[1].kp_leng  =  5;
keydesc1.k_part[1].kp_flags = FA_ANYKPCODE;
```

Example 2

- The primary record key does not permit duplication. The number of key parts is 3.
- The first part is at the first byte and its length is 3.
- The second part is at the ninth byte counted from the first byte and its length is 2.
- The third part is at the fifth byte counted from the first byte and its length is 4.



COBOL program example

```
001000 environment division.
001100 configuration section.
001200 input-output section.
001300 file-control.
001400     select FILENAME-1 assign to SYS006
001500     organization is indexed
001600     record key is part-1 part-2 part-3.
   :
002000 data division.
002100 file section.
002200 fd FILENAME-1.
002300   01 record-1.
002400     02 part-1 pic x(3).
002500     02 filler pic x(1).
002400     02 part-3 pic x(4).
002500     02 part-2 pic x(2).
002600     02 filler pic x(...
```

C program example

```
#include "cobfa.h"
struct fa_keydesc keydesc2;
keydesc2.k_flags  = FA_NODUPS;
keydesc2.k_nparts = 3;                        /* number of key parts: 3 */
keydesc2.k_part[0].kp_start =  0;             /* part_1:  1 - 1 ==  0   */
keydesc2.k_part[0].kp_leng  =  3;
keydesc2.k_part[0].kp_flags = FA_ANYKPCODE;
keydesc2.k_part[1].kp_start =  8;             /* part_2:  9 - 1 ==  8   */
keydesc2.k_part[1].kp_leng  =  2;
keydesc2.k_part[1].kp_flags = FA_ANYKPCODE;
keydesc2.k_part[2].kp_start =  4;             /* part_3:  5 - 1 ==  4   */
keydesc2.k_part[2].kp_leng  =  4;
keydesc2.k_part[2].kp_flags = FA_ANYKPCODE;
```

4.2 struct fa_keylist

Specification for the list of all record keys.

For the *cobfa_open()* function, the list of all record keys in the indexed file to be opened is specified with the *struct fa_keylist* type.

Values to be set for the *struct fa_keylist* type members that give the overall organization of record keys are explained below.

```
#define  FA_NKEYS     126u                    /* max number of keys          */
struct  fa_keylist  {
 long  kl_nkeys;                              /* number of keydesc           */
 struct  fa_keydesc  *kl_key  [FA_NKEYS];     /* keydesc address of each key */
};
```

- Set the total number of record keys for <*kl_nkeys*>.

  Since an indexed file always contains the primary record key, the total number of record keys is always 1 or greater. The maximum value for this total number is FA_NKEYS (typically 126).

- <*kl_key*> maintains information on each record key and is declared by the *struct fa_keydesc* type pointer. For the *struct fa_keydesc* type, see "*struct fa_keydesc*".

The sum of key parts retained by the record keys must not exceed FA_NALLPARTS (typically 255).

The sum of lengths of key parts retained by the record keys must not exceed FA_NALLKEYSIZE (typically 255).

Practical examples

Example

- The list of record keys in an indexed file is one primary record key and one alternate record key (2 keys).
- The primary record key has two key parts and does not permit duplication.
- The first key part is at the first byte and its length is 4, the second key part is at the seventh byte counted from the first byte and its length is 2.
- The alternate record key has one key part can be duplicated.
- The key part is at the twelfth byte counted from the first byte and its length is 3.

```
Byte:    | 1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16
Offset:  |0--|1--|2--|3--|4--|5--|6--|7--|8--|9--|10-|11-|12-|13-|14-|15-|
         |=============================================================....
         |---+---+---+---|       |---+---|           |---+---+---|
          first part of the       second part of the  alternate
          primary record key      primary record key  record key
```

COBOL program example

```
001000 environment division.
001100 configuration section.
001200 input-output section.
001300 file-control.
001400     select FILENAME-1 assign to SYS006
001500     organization is indexed
001600     record key is part-1 part-2
001700     alternate record key is subpart with duplicates.
   :
002000 data division.
002100 file section.
002200 fd FILENAME-1.
002300   01 record-1.
002400     02 part-1  pic x(4).
002500     02 filler  pic x(2).
002400     02 part-2  pic x(2).
```

```
002500     02 filler  pic x(3).
002600     02 subpart pic x(3).
002700     02 filler  pic x(...
```

C program example

```
#include "cobfa.h"

struct fa_keylist keylist;           /* for all keys structure         */
struct fa_keydesc keydesc1;          /* for prime record key           */
struct fa_keydesc keydesc2;          /* for alternate record key       */

keylist.kl_nkeys  = 2;               /* number of keys: 2 (prim & alt) */
keylist.kl_key[0] = &keydesc1;       /* prime key address              */
keylist.kl_key[1] = &keydesc2;       /* alternate key address          */

keydesc1.k_flags  = FA_NODUPS;
keydesc1.k_nparts = 2;               /* number of key parts: 2         */
keydesc1.k_part[0].kp_start =  0;    /*  1 - 1 ==  0                   */
keydesc1.k_part[0].kp_leng  =  4;
keydesc1.k_part[0].kp_flags = FA_ANYKPCODE;
keydesc1.k_part[1].kp_start =  6;    /*  7 - 1 ==  6                   */
keydesc1.k_part[1].kp_leng  =  2;
keydesc1.k_part[1].kp_flags = FA_ANYKPCODE;

keydesc2.k_flags  = FA_DUPS;
keydesc2.k_nparts = 1;               /* number of key parts: 1         */
keydesc2.k_part[0].kp_start = 11;    /* 12 - 1 == 11                   */
keydesc2.k_part[0].kp_leng  =  3;
keydesc2.k_part[0].kp_flags = FA_ANYKPCODE;
```

4.3 struct fa_dictinfo

Information about an opened indexed file.

For the *cobfa_indexinfo()* function, information on the opened indexed file can be returned with the *struct fa_dictinfo* type. For details, see "3.6 cobfa_indexinfo()".

Values to be set for the *struct fa_dictinfo* type members used to acquire indexed file information are explained below.

```
struct  fa_dictinfo  {
    long  di_nkeys;              /* number of keys defined          */
    long  di_recsize;            /* max or fixed data record size   */
    long  di_idxsize;            /* size of indexes                 */
    long  di_flags;              /* other flags (fixed or variable) */
};
```

- The total number of record keys in the indexed file is set for <*di_nkeys*>.
- The fixed record length or maximum record length is set for <*di_recsize*> indicating the record length of the file. With this access routine, the minimum record length cannot be acquired.
- The total length of record keys is set for <*di_idxsize*> indicating the sum of lengths of all record keys.
- <*di_flags*> indicating the file attribute has information on the following category:
  - Record format
    - FA_FIXLEN: Fixed-length record format
    - FA_VARLEN: Variable-length record format

  Individual attribute values can be determined by application of a logical AND to the <*di_flags*> value.

Example

```
#include "cobfa.h"
struct  fa_dictinfo  di;
long  fd, ret;
    :
ret = cobfa_indexinfo (fd,(struct fa_keydesc *)(&di),0);   /* get indexed file info    */
if (di.di_flags & FA_FIXLEN) {
         :                              /* process for fixed length record type */
}
```

Chapter 5 Error Number and I-O Status

Errors encountered by the API functions are available from the *cobfa_errno()* and *cobfa_stat()* functions. In both cases, the value represents the error condition from the most recent API call.

Values returned by *cobfa_stat ()* are comparable to the FILE-STATUS values available at runtime. *Cobfa_errno()* provides a more comprehensive range of error conditions, including errors caused by passing incorrect parameters to the API functions.

5.1 Error Number

Error numbers and their meanings are explained below.

Error Number	Meanings
FA_ENOERR (0)	Execution of the I-O feature or file information acquisition is successful
FA_ENOSPC (28)	Insufficient disk space.
FA_EDUPL (100)	Key duplication error. The status is one of the following: - The value of the record key not permitting duplication is duplicated. - The value of a relative record number is duplicated.
FA_ENOTOPEN (101)	File open error. The status is one of the following: - The file is not opened yet. - The file is opened in a mode other than the open mode that allows execution of this feature.
FA_EBADARG (102)	Argument error. The status is one of the following: - The record area address is the NULL pointer. - For a function for file information acquisition, the function number is out of the range. - For a function for file information acquisition, the structure pointer is NULL.
FA_EBADKEY (103)	Indexed file record key specification error. The status is one of the following: - The record key list is improperly constructed. - The record key list provided does not match the file key list. - The record key number is out of range.
FA_ETOOMANY (104)	An attempt was made to open the number of files that exceeded the limit given by the OS or this access routine.
FA_EBADFILE (105)	File structure error. The status is one of the following: - The file internal information is incorrect or has been destroyed. - The correct file organization has not been specified. - The character encoding type does not match the encoding forms (ShiftJIS, UCS-2, UTF-8) of the line sequential file.
FA_ELOCKED (107)	The record is locked.
FA_EENDFILE (110)	An end of file condition occurred.
FA_ENOREC (111)	The specified record does not exist.
FA_ENOCURR (112)	Positioning to the record is undefined.
FA_EFLOCKED (113)	The file is already opened exclusively.
FA_EFNAME (114)	Error on the filename given when the file was opened. The status is one of the following: - The file does not exist.

Error Number	Meanings
	- The file cannot be accessed. - The filename is a NULL pointer or void string. - Filename organization is incorrect. - An attempt to open a file with a read-only attribute is made in a mode other than the INPUT mode.
FA_EBADMEM (116)	Acquisition of the necessary memory for executing the feature failed.
FA_EKEYSEQ (117)	Key sequence error or key change error. The status is one of the following: - The primary record key value is not in ascending order in sequential write. - The primary record key value was changed with sequential rewrite.
FA_EBADACC (118)	The combination of parameters is invalid. The status is one of the following: - Execution of a feature violating the access mode is requested. - The feature cannot be executed with this file organization. - The flag combination provided when the file is opened is incorrect.
FA_EBADFLAG (120)	The value specified for the flag is incorrect. The status is one of the following: - An unusable mode is specified for the open mode, read mode, or positioning mode. - An unacceptable value is specified for the flag.
FA_EBADLENG (121)	Length error. The status is one of the following: - The record length exceeds the permissible range. Refer to "General file features" in "7.2 Caution". - The valid key length for positioning exceeds the permissible value.
FA_EUNDEFKEY (122)	When a file is opened specifying the dummy file function and the record key specification is omitted, index file information cannot be acquired. Refer to "3.25 Dummy File" for details.
FA_EOTHER (999)	An error other than the above occurred. Obtain the return value of the cobfa_stat () function and check the status. For the cobfa_stat() function, see "3.17 cobfa_stat()".

5.2 I-O Status

The I-O status can be acquired by calling the *cobfa_stat()* function. For details, see "3.17 cobfa_stat()".

Refer to "I-O Status List" in "NetCOBOL User's Guide" for the complete list if I-O status value*s*.

Chapter 6 Example Programs

Three example programs are provided. These "makefiles" (Makefile) explicitly describes the directory name, so modify the contents depending on your environment.

6.1 Line Sequential File INPUT

Overview

This example program opens the specified file in INPUT mode as a line sequential file and displays the contents of the record that is read.

Compilation procedure

Provide the directory name in the "makefile" (Makefile) according to your environment. Then, enter the following command:

```
> make
```

Execution procedure

Run the program by using a proper text file as the argument. For example, enter the source program of cobfa01 itself.

```
> cobfa01 cobfa01.c
```

Directory

```
/opt/FJSVcbl64/samples/C/cobfa01
```

6.2 Line Sequential File INPUT and Indexed File OUTPUT

Overview

This example program opens a particular line sequential file (cobfa02.txt) in INPUT mode and writes the record contents as the record of the indexed file (cobfa02.idx). Finally, this example program opens the indexed file in INPUT mode and displays it on the screen in the primary record key order.

Compilation procedure

Provide the directory name in the "makefile" (Makefile) according to your environment. Then, enter the following command:

```
> make
```

Execution procedure

Execute this program without using any argument.

```
> cobfa02
```

Directory

```
/opt/FJSVcbl64/samples/C/cobfa02
```

6.3 Indexed File Information Acquisition

Overview

This example program opens the specified file in INPUT mode as the indexed file and displays the attributes of the file itself and each structure of the record keys.

Compilation procedure

Provide the directory name in the "makefile" (Makefile) according to your environment. Then, enter the following command:

```
> make
```

Execution procedure

Execute the program by using a proper indexed file as the argument. For example, specify the indexed file created by executing the sample program cobfa02.

```
> cobfa03 cobfa02.idx
```

Directory

```
/opt/FJSVcbl64/samples/C/cobfa03
```

Chapter 7 Notes

7.1 Limitation Items

Known limitation items are enumerated below.

Indexed file

- For the file opened according to the dynamic access method (FA_DYNACC), there is no way to sequentially rewrite or delete the record that has been positioned as a result of sequential read, under the following condition:
  - FA_REVORD is specified by the *cobfa_stkey()* function, or
  - Attribute of the primary record key allows duplicated (FA_DUPS).

7.2 Caution

General file features

- Be sure to close all of the files that have been opened by the application program before exiting the application program. Without this operation, file corruptions and /or system resource leaks may arise.
- A double open error (I-O status: 41) does not occur even though the same file is opened in a process. In this case, a different file descriptor is assigned. The maximum number of file descriptors that can be opened concurrently in the same process is 1,024.
- If two or more errors occur when the API function is executed, an I-O status different from those of the COBOL application may be returned.
- For the I-O statements in COBOL that do not need explicit description of the access method, the sequential access/random access API functions must be explicitly described to use the API. Be sure to describe the file organization and access method.
- The current status functions (cobfa_errno(), cobfa_stat(), cobfa_reclen(), and cobfa_recnum()) are shared values and do not retain values for each file descriptor. These values are replaced or destroyed by the next call to an API function. Therefore, these values should be saved in local variables before calling the next I-O function.
- To read, write, rewrite, or position a record, an area that is equivalent to or larger than the maximum/fixed record length used to open the file should be reserved in advance as the area for passing the data. For functions using other pointers, be sure to reserve the necessary area in advance as well.
- Note that errors can be detected only when API functions are executed although errors can be detected during compilation in COBOL. Practically, these errors are inconsistent bit-wise flags, incorrect index key specification, etc.
- For quantitative restrictions of files, refer to "File Organization Types" and "Other File Functions", in "File Processing" of "NetCOBOL User's Guide" and "System Quantitative Restrictions" in "NetCOBOL Language Reference."
- A large file specification for cobfa_open(), which has been used in COBOL for other UNIX systems, is no longer required. If one has been designated, it is handled normally.
- For correspondence between the COBOL data types and C language data types, refer to "Correspondence of COBOL and C Data Types", in "Calling Subprograms" of "NetCOBOL User's Guide."
  - Pay particular attention to how COMP-5 and BINARY are handled, as the internal formats are different.
- Enclose the file name with double quotation marks (") when you specify a file that contains commas (,) in the file name.

Line sequential file

- A 0-byte record cannot be written.
- 0x0A is treated as a line feed character in this product, and it adds when writing a record.
- A control character contained in a record to be read is treated as follows:
  - 0x0C (page feed): Treated as a record delimiter

  - 0x0D (return): Treated as a record delimiter
  - 0x1A (data end symbol): Treated as a file terminator
- If a record read from a line sequential file includes a tab, the tab code is replaced by a blank. Refer to "Processing Line Sequential Files", in "File Processing" of "NetCOBOL User's Guide" for details.

Record sequential file

- The print file cannot be handled, therefore the following limitations:
  - No support for LINAGE clause in the COBOL syntax.
  - No support for the line feed control /page control for the WRITE statement in the COBOL syntax.

Relative file

- The relative record number must be explicitly specified by using an argument in random delete (cobfa_delrec()), random write (cobfa_wrrec()), random rewrite (cobfa_rewrec()), and random read (cobfa_rdrec()).
- The following functions in this product are used to specify a large COBOL file intended for use with COBOL for other UNIX systems, and they do not have a functional difference with their corresponding regular functions:
  - cobfa_delrec64() function (random deletion)
  - cobfa_rewrec64() function (random rewriting)
  - cobfa_rdrec64() function (random reading)
  - cobfa_strec64() function (positioning)
  - cobfa_recnum64() function (getting a relative record number)

Indexed file

- When opening an existing file, you can open it without specifying the record key list. In this case, the file is opened based on the record key list in the existing file.
- The cobfa_indexinfo() function can be used to retrieve the indexed file attributes (record length and record format) or record key list after the file is open.

Multithreading

- The following functions must be used for exclusive control (setting and opening exclusive lock) when accessing COBOL files. For details, see the description of each API function.
  - LOCK_cobfa()
  - UNLOCK_cobfa()
- In a multithread environment, exclusive control is required to access COBOL files before an API function with input/output function is called and before calling an API function in which information on the file is acquired. If the access is not exclusively controlled, the problems listed below may occur.
  Refer to the " LOCK_cobfa()" usage example for the timing of exclusive control lock/unlock.
  - Incorrect execution of input-output files
  - Abnormal termination of the thread
  - File corruption

Unicode

- Note the following point regarding handling of Unicode:
  - Set the number of bytes for the key part length kp_leng of indexed files to be handled by COBOL applications operating in Unicode mode.

- Set the number of bytes for the record length of line sequential files to be handled by COBOL applications operating in Unicode mode.

Index