BEA MessageQ

Reference Manual

BEA MessageQ Version 5.0
Document Edition 1.0
October 1998

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA MessageQ Reference Manual

Document Edition	Date	Software Version
Version 1.0	October 1998	BEA MessageQ, Version 5.0

Contents

Preface

Purpose of This Document v
How to Use This Document vi
Related Documentation x
Contact Information xi

1. FML Functions

Fintro (3FML) 1-3
CFadd (3FML) 1-7
CFchg (3FML) 1-9
CFfind (3FML) 1-11
CFfindocc (3FML) 1-13
CFget (3FML) 1-15
CFgetalloc (3FML) 1-17
F_error (3FML) 1-19
Fadd (3FML) 1-20
Fadds (3FML) 1-22
Falloc (3FML) 1-24
Fappend (3FML) 1-25
Fboolco (3FML) 1-27
Fboolev (3FML) 1-29
Fboolpr (3FML) 1-31
Fchg (3FML) 1-32
Fchgs (3FML) 1-34
Fchksum (3FML) 1-35
Fcmp (3FML) 1-36
Fconcat (3FML) 1-37

Fcpy (3FML) 1-38
Fdel (3FML) 1-39
Fdelall (3FML) 1-40
Fdelete (3FML) 1-41
Fextread (3FML) 1-42
Ffind (3FML) 1-45
Ffindlast (3FML) 1-47
Ffindocc (3FML) 1-49
Ffinds (3FML) 1-51
Ffloatev (3FML) 1-53
Ffprint (3FML) 1-54
Ffree (3FML) 1-55
Fget (3FML) 1-56
Fgetalloc (3FML) 1-58
Fgetlast (3FML) 1-60
Fgets (3FML) 1-62
Fgetsa (3FML) 1-64
Fidnm_unload (3FML) 1-66
Fidxused (3FML) 1-67
Fielded (3FML) 1-68
Findex (3FML) 1-69
Finit (3FML) 1-70
Fjoin (3FML) 1-71
Fldid (3FML) 1-73
Fldno (3FML) 1-74
Fldtype (3FML) 1-75
Flen (3FML) 1-76
Fmkfldid (3FML) 1-77
Fmove (3FML) 1-78
Fname (3FML) 1-79
Fneeded (3FML) 1-80
Fnext (3FML) 1-81
Fnmid_unload (3FML) 1-83
Fnum (3FML) 1-84
Foccur (3FML) 1-85

Fojoin (3FML) 1-86
Fpres (3FML) 1-88
Fprint (3FML) 1-89
Fproj (3FML) 1-90
Fprojcpy (3FML) 1-92
Fread (3FML) 1-93
Frealloc (3FML) 1-95
Frstrindex (3FML) 1-96
Fsizeof (3FML) 1-98
Fstrerror (3FML) 1-99
Ftypcvt (3FML) 1-100
Ftype (3FML) 1-101
Funindex (3FML) 1-102
Funused (3FML) 1-103
Fupdate (3FML) 1-104
Fused (3FML) 1-105
Fvall (3FML) 1-106
Fvals (3FML) 1-107
Fwrite (3FML) 1-108

2. field_tables Description

field_tables(5) 2-2

3. mkfldhdr Command

mkfldhdr, mkfldhdr32 3-2

4. MessageQ/TUXEDO Bridge Functions

TMQUEUE_BMQ 4-2
TMQFORWARD_BMQ 4-5
tpdequeue (3) 4-8
tpenqueue (3) 4-14

Preface

Purpose of This Document

This document provides a detailed description of the following types of functions:

- Field Manipulation Language (FML) functions used in the development of BEA MessageQ applications
- MessageQ/TUXEDO bridge functions used to enable the exchange of messages between BEA MessageQ and BEA TUXEDO applications

For a detailed description of BEA MessageQ PAMS API functions, see the *BEA MessageQ Programmer's Guide*.

Who Should Read This Document

This document is intended for applications designers and developers who are interested in designing, developing, building, and running BEA MessageQ applications.

How This Document Is Organized

BEA MessageQ Reference Manual is organized as follows:

- Chapter 1, "FML Functions" describes the functions used to define and manipulate fielded buffers.

- Chapter 2, “field_tables Description” describes the FML mapping files for field names.
- Chapter 3, “mkfldhdr Command” describes the function used to create header files from field tables.
- Chapter 4, “MessageQ/TUXEDO Bridge Functions,”describes the functions used to enable the exchange of messages between BEA MessageQ and BEA TUXEDO applications.

How to Use This Document

This document, BEA MessageQ Reference Manual, is designed primarily as an online, hypertext document. If you are reading this as a paper publication, note that to get full use from this document you should access it as an online document via the BEA MessageQ Online Documentation CD.

The following sections explain how to view this document online, and how to print a copy of this document.

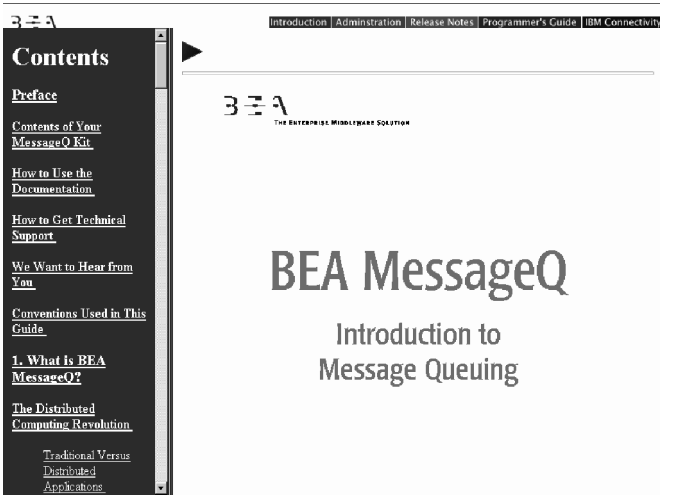
Opening the Document in a Web Browser

To access the online version of this document, open the `index.htm` file in the top-level directory of the BEA MessageQ Online Documentation CD. On the main menu, click the Introduction to Message Queuing button.

Note: The online documentation requires a Web browser that supports HTML version 3.0. Netscape Navigator version 3.0 or Microsoft Internet Explorer version 3.0 or later are recommended.

Figure 1 shows the online document with the clickable navigation bar and table of contents.

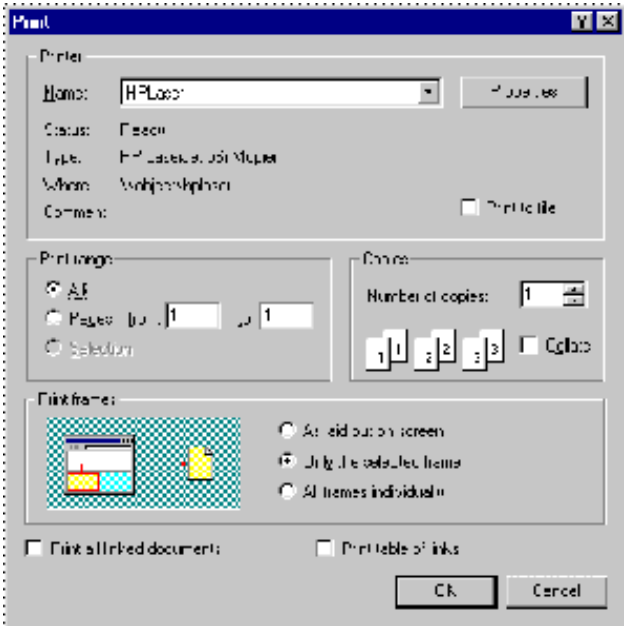
Figure 1 Online Document Displayed in a Netscape Web Browser



Printing from a Web Browser

You can print a copy of this document, one file at a time, from the Web browser. Before you print, make sure that the chapter or appendix you want is displayed and *selected* in your browser.

To select a chapter or appendix, click anywhere inside the chapter or appendix you want to print. If your browser offers a Print Preview feature, you can use the feature to verify which chapter or appendix you are about to print. If your browser offers a Print Frames feature, you can use the feature to select the frame containing the chapter or appendix you want to print. For example:



The BEA MessageQ Online Documentation CD also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document.

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

Convention	Item

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

Convention	Item

. . .	Indicates one of the following in a command line:
	♦ That an argument can be repeated several times in a command line
	♦ That the statement omits additional optional arguments
	♦ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Related Documentation

The following sections list the documentation provided with the MessageQ software, related BEA publications, and other publications related to the technology.

MessageQ Documentation

The MessageQ information set consists of the following documents:

BEA MessageQ Introduction to Message Queuing

BEA MessageQ Programmer's Guide

BEA MessageQ Installation Guide

BEA MessageQ System Messages

BEA MessageQ Client Guide

BEA MessageQ FML Programmer's Guide

Note: The BEA MessageQ Online Documentation CD also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document.

Contact Information

The following sections provide information about how to obtain support for the documentation and software.

Documentation Support

If you have questions or comments on the documentation, you can contact the BEA Information Engineering Group by e-mail at **docsupport@beasys.com**. (For information about how to contact Customer Support, refer to the following section.)

Customer Support

If you have any questions about this version of BEA MessageQ, or if you have problems installing and running BEA MessageQ, contact BEA Customer Support through BEA WebSupport at `www.beasys.com`. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

1 FML Functions

1

Fintro (3FML)

Name `Fintro`—Introduction to FML functions

Synopsis

```
"#include <fml.h>"
"#include <fml32.h>"
```

Description FML is a set of C language functions for defining and manipulating storage structures called `fielded buffers`, that contain attribute-value pairs called fields. The attribute is the field's identifier, and the associated value represents the field's data content.

Fielded buffers provide an excellent structure for communicating parameterized data between cooperating processes, by providing named access to a set of related fields. Programs that need to communicate with other processes can use the FML software to provide access to fields without concerning themselves with the structures containing them.

FML16 and FML32 There are two "sizes" of FML. The original FML interface is based on 16-bit values for the length of fields and containing information identifying fields. In this introduction, it will be referred to as FML16. FML16 is limited to 8191 unique fields, individual field lengths of up to 64K bytes, and a total fielded buffer size of 64K. The definitions, types, and function prototypes for this interface are in `fml.h` which must be included in an application program using the FML16 interface; and functions live in `-lfml`. A second interface, FML32, uses 32-bit values for the field lengths and identifiers. It allows for about 30 million fields, and field and buffer lengths of about 2 billion bytes. The definitions, types, and function prototypes for FML32 are in `fml32.h`; and functions live in `-lfml32`. All definitions, types, and function names for FML32 have a "32" suffix (for example, `MAXFBLEN32`, `FLDID32`, `Fchg32`). Also the environment variables are suffixed with "32" (for example, `FLDTBLDIR32` and `FIELDTBLS32`).

FML Buffers A fielded buffer is composed of field identifier and field value pairs for fixed length fields (for example, `long, short`), and field identifier, field length, and field value triples for varying length fields.

A field identifier is a tag for an individual data item in a fielded buffer. The field identifier consists of the name of field number and the type of the data in the field. The field number must be in the range 1 to 8191, inclusive, for FML16 and the type definition for a field identifier is `FLDID`. The field number must be in the range 1 to 33,554,431, inclusive, for FML32 and the type definition for a field identifier is `FLDID32`. Field numbers 1 to 100 are reserved for system use and should be avoided (although this is not strictly enforced). The field types can be any of the standard C language types: `short, long, float, double,` and `char`. Two other types are also

supported: string (a series of characters ending with a null character) and carray (character arrays). These types are defined in fml.h and fml32.h as FLD_SHORT, FLD_LONG, FLD_CHAR, FLD_FLOAT, FLD_DOUBLE, FLD_STRING, and FLD_CARRAY.

For FML16, a fielded buffer pointer is of type "FBFR *", a field length has the type FLDLEN, and the number of occurrences of a field has the type FLDOCC. For FML32, a fielded buffer pointer is of type "FBFR32 *", a field length has the type FLDLEN32, and the number of occurrences of a field has the type FLDOCC32.

Fields are referred to by their field identifier in the FML interface. However, it is normally easier for an application programmer to remember a field name. There are two approaches to mapping field names to field identifiers.

Field name/identifier mappings can be made available to FML programs at run-time through field table files, described in field_tables(5). The FML16 interface uses the environment variable FLDTBLDIR to specify a list of directories where field tables can be found, and FIELDTBLS to specify a list of the files in the table directories that are to be used. The FML32 interface uses FLDTBLDIR32 and FIELDTBLS32. Within applications programs, the FML functions Fldid and Fldid32 provide for a run-time translation of a field name to its field identifier and Fname and Fname32 translate a field identifier to its field name.

Compile-time field name/identifier mappings are provided by the use field header files containing macro definitions for the field names. mkfldhdr(1) and mkfldhdr32(1) are provided to make header files out of field table files. These header files are #include'd in C programs, and provide another way to map field names to field identifiers: at compile-time.

Any field in a fielded buffer can occur more than once. Many FML functions take an argument that specifies which occurrence of a field is to be retrieved or modified. If a field occurs more than once, the first occurrence is numbered 0, and additional occurrences are numbered sequentially. The set of all occurrences make up a logical sequence, but no overhead is associated with the occurrence number (that is, it is not stored in the fielded buffer). If another occurrence of a field is added, it is added at the end of the set and is referred to as the next higher occurrence. When an occurrence other than the highest is deleted, all higher occurrences of the field are shifted down by one (for example, occurrence 6 becomes occurrence 5, 5 becomes 4, etc.).

When a fielded buffer has many fields, access is expedited in FML by the use of an internal index. The user is normally unaware of the existence of this index. However, when you store a fielded buffer on disk, or transmit a fielded buffer between processes

or between computers, you can save disk space and/or transmittal time by first discarding the index using `Funindex` or `Funindex32`, and then reconstructing the index later with `Findex` or `Findex32`.

FML16 Conversion to FML32

Existing FML16 applications that are written correctly can easily be changed to use the FML32 interface. All variables used in the calls to the FML functions must use the proper typedefs (`FLDID`, `FLDLEN`, and `FLDOCC`). The application source code can be changed to use the 32-bit functions simply by changing the include of `fml.h` to inclusion of `fml32.h` followed by `fml1632.h`. The `fml1632.h` contains macros that convert all of the 16-bit type definitions to 32-bit type definitions, and 16-bit functions and macros to 32-bit functions and macros.

Error Handling

Most of the FML functions have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is usually -1 on error, or 0 for a bad field identifier (`BADFLDID`) or address. The error type is also made available in the external integer *Ferror* for FML16 and *Ferror32* for FML32. *Ferror* and *Ferror32* are not cleared on successful calls, so they should be tested only after an error has been indicated.

The `F_error` and `F_error32` functions are provided to produce a message on the standard error output. They take one parameter, a string; print the argument string appended with a colon and a blank; and then print an error message followed by a newline character. The error message displayed is the one defined for the error number currently in *Ferror* or *Ferror32*, which is set when errors occur.

`Fstrerror`(3) can be used to retrieve from a message catalog the text of an error message; it returns a pointer that can be used to as an argument to `userlog`(3).

The error codes that can be produced by an FML function are described on each FML reference page.

See Also

`CFadd`(3fml), `CFchg`(3fml), `CFfind`(3fml), `CFfindocc`(3fml), `CFget`(3fml), `CFgetalloc`(3fml), `F_error`(3fml), `Fadd`(3fml), `Fadds`(3fml), `Falloc`(3fml), `Fboolco`(3fml), `Fboolev`(3fml), `Fboolpr`(3fml), `Fchg`(3fml), `Fchgs`(3fml), `Fchksum`(3fml), `Fcmp`(3fml), `Fconcat`(3fml), `Fcpy`(3fml), `Fdel`(3fml), `Fdelall`(3fml), `Fdelete`(3fml), `Fextread`(3fml), `Ffind`(3fml), `Ffindlast`(3fml), `Ffindocc`(3fml), `Ffinds`(3fml), `Ffloatev`(3fml), `Ffprint`(3fml), `Ffree`(3fml), `Fget`(3fml), `Fgetalloc`(3fml), `Fgetlast`(3fml), `Fgets`(3fml), `Fgetsa`(3fml), `Fidnm_unload`(3fml), `Fidxused`(3fml), `Fielded`(3fml), `Findex`(3fml), `Finit`(3fml), `Fjoin`(3fml), `Fldid`(3fml), `Fldno`(3fml), Fldtype(3fml), `Flen`(3fml), `Fmkfldid`(3fml), `Fmove`(3fml), `Fname`(3fml), `Fneeded`(3fml), `Fnext`(3fml), `Fnmid_unload`(3fml), `Fnum`(3fml), `Foccur`(3fml), `Fojoin`(3fml), `Fpres`(3fml), `Fprint`(3fml), `Fproj`(3fml), `Fprojcpy`(3fml), `Fread`(3fml), `Frealloc`(3fml),

`Frstrindex`(3fml), `Fsizeof`(3fml), `Fstrerror`(3fml), `Ftypcvt`(3fml), `Ftype`(3fml), `Funindex`(3fml), `Funused`(3fml), `Fupdate`(3fml), `Fused`(3fml), `Fvall`(3fml), `Fvals`(3fml), `Fvftos`(3fml), `Fvnull`(3fml), `Fvopt`(3fml), `Fvselinit`(3fml), `Fvsinit`(3fml), `Fvstof`(3fml), `Fwrite`(3fml), `field_tables`(5), *BEA MessageQ FML Programmer's Guide*

CFadd (3FML)

Name CFadd, CFadd32—convert and add field

Synopsis

```
#include <stdio.h>
#include "fml.h"
int CFadd(FBFR *fbfr, FLDID fieldid, char *value, FLDLEN len, int
type)
#include fml32.h>
int
CFadd32(FBFR32 *fbfr, FLDID32 fieldid, char *value, FLDLEN32 len,
int type)
```

Description CFadd() acts like Fadd() but firstconverts the *value* from the user-specified type to the type of the *fieldid* for which the field is added to the fielded buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *value* is a pointer to the value to be added. *len* is the length of the value to be added; it is required only if *type* is FLD_CARRAY. *type* is the data type of the field in *value*.

Before the field is added to the buffer,the type of the data item is converted from type supplied by the user to the type specified in in *fieldid*. If the source type is FLD_CARRAY (arbitrary character array), the *len* argument should be set to the length of the array; the length is ignored in all other cases. The value for the field to be converted and added must first be put in a variable, *value*, since C does not permit constructs such as 12345L.

CFadd32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, CFadd() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed when converting from a carray to string.

[FEINVAL]
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, a NULL *value* parameter was specified).

[FNOSPACE]
"no space in fielded buffer"
A field value is to be added or changed in a field buffer but there is not enough space remaining in the buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fadd`(3)

CFchg (3FML)

Name CFchg, CFchg32—convert and change field

Synopsis

```
#include <stdio.h>
#include "fml.h"
int CFchg(FBFR *fbfr, FLDID fieldid, FLDOCC oc, char *value,
        FLDLEN len, int type)
#include "fml32.h"
int CFchg32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc,
        char *value,
FLDLEN32 len, int type)
```

Description CFchg() acts like Fchg() but first converts the *value* from the user-specified *type* to the type of the *fieldid* for which the field is changed in the fielded buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. *value* is a pointer to a new value. *len* is the length of the value to be changed; it is required only if type is FLD_CARRAY. *type* is the data type of *value*.

If a field occurrence is specified that does not exist, then NULL values are added for the missing occurrences until the desired value can be added (e.g., changing field occurrence 4 for a field that does not exist in a buffer will cause 3 NULL values to be added followed by the specified field value).

CFchg32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, CFchg() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed when converting from a carray to string.

[FEINVAL]
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, a NULL *value* parameter was specified).

[FNOSPACE]
"no space in fielded buffer"
A field value is to be added or changed in a field buffer but there is not enough space remaining in the buffer.

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also Fintro(3), CFadd(3), Fchg(3)

CFfind (3FML)

Name CFfind, CFfind32—find, convert and return pointer

Synopsis

```
#include <stdio.h>
#include "fml.h"
char * CFfind(FBFR *fbfr, FLDID fieldid, FLDOCC oc, FLDLEN *len,
  int type)
#include "fml32.h"
char *
CFfind32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc,  FLDLEN32
*len, int type)
```

Description CFfind() finds a specified field in a buffer, converts it and returns a pointer to the converted value. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. *len* is used on output and is a pointer to the length of the converted value. *type* is the data type the user wants the field to be converted to.

Like Ffind(3), the pointer returned by the function should be considered read only. The validity of the pointer returned by CFfind() is guaranteed only until the next buffer operation, even if that operation is non-destructive, since the converted value is retained in a single private buffer. This differs from the value returned by Ffind(3), which is guaranteed until the next modification of the buffer. Unlike Ffind(3), CFfind() aligns the converted value for immediate use by the caller.

CFfind32 is used with 32-bit FML.

Return Values In the SYNOPSIS section above the return value to CFfind() is described as a character pointer data type (char ** in C). Actually, the pointer returned points to an object that has the same type as the stored type of the field.

This function returns NULL on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, CFfind() fails and sets Ferror to:

[FALIGNERR]
: "fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
: "buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed when converting from a carray to string.

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Ffind`(3)

CFfindocc (3FML)

Name CFfindocc, CFfindocc32—find occurrence of converted value

Synopsis

```
#include <stdio.h>
#include "fml.h"
FLDOCC
CFfindocc(FBFR *fbfr, FLDID fieldid, char *value, FLDLEN len, int
  type)
#include "fml32.h"
FLDOCC32
CFfindocc32(FBFR32 *fbfr, FLDID32 fieldid, char *value, FLDLEN32
  len, int type)
```

Description CFfindocc() acts like Ffindocc() but first converts the *value* from the user-specified type to the type of *fieldid*. CFfindocc() looks for an occurrence of the specified field in the buffer that matches a user-supplied value, length and type. CFfindocc() returns the occurrence number of the first field that matches. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *value* is a pointer to the value being sought. *len* is the length of the value to be compared to input value if *type* is carray. *type* is the data type of the field in *value*.

CFfindocc32 is used with 32-bit FML.

Return Values If the field value is not found or if other errors are detected, -1 is returned and CFfindocc() sets Ferror to indicate the error condition.

Errors Under the following conditions, CFfindocc() fails and sets Ferror to:

[FALIGNERR]
: fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
: "buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FMALLOC]
: "malloc failed"
Allocation of space dynamically using malloc(3) failed when converting from a carray to string.

[FEINVAL]
: "invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, a NULL *value* parameter was specified).

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also Fintro(3), Ffindocc(3)

CFget (3FML)

Name `CFget, CFget32`—get field and convert

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
CFget(FBFR *fbfr, FLDID fieldid, FLDOCC oc, char *buf, FLDLEN *len,
 int type)
#include "fml32.h"
int
CFget32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, char *buf,
 FLDLEN32 *len, int type)
```

Description `CFget()` is the conversion analog of `Fget`(3). The main difference is that it copies a converted value to the user supplied buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. *buf* is a pointer to private data area. On input, *len* is a pointer to the length of the private data area. On return, *len* is a pointer to the length of the returned value. If the *len* parameter is NULL on input, it is assumed that the buffer is big enough to contain the field value and the length of the value is not returned. If the *buf* parameter is NULL, the field value is not returned. *type* is the data type the user wants the returned value converted to.

`CFget32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `CFget`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using malloc(3) failed when converting from a carray to string.

`[FNOSPACE]`
"no space in fielded buffer"
The size of the data area, as specified in *len*, is not large enough to hold the field value.

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also Fintro(3), Fget(3)

CFgetalloc (3FML)

Name `CFgetalloc, CFgetalloc32`—get field, space, convert

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
CFgetalloc(FBFR *fbfr, FLDID fieldid, FLDOCC oc, int type, FLDLEN
 *extralen)
#include "fml32.h"
char *
CFgetalloc32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, int type,
 FLDLEN32 *extralen)
```

Description `CFgetalloc()` gets a specified field from a buffer, allocates space, converts the field to the type specified by the user and returns a pointer to its location. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. *type* is the data type the user wants the field to be converted to. On call, *extralen* is a pointer to the length of additional space that may be allocated to receive the value; on return, it is a pointer actual amount of space used. If *extralen* is `NULL`, then no additional space is allocated and the actual length is not returned. The user is responsible for freeing the returned (converted) value.

`CFgetalloc32` is used with 32-bit FML.

Return Values On success, `CFgetalloc()` returns a pointer to the converted value. On error, the function returns `NULL` and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `CFgetalloc`() fails and sets `Ferror` to:

`[FALIGNERR]`
: "fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
: "buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FMALLOC]`
: "`malloc` failed"
Allocation of space dynamically using `malloc`(3) failed.

`[FNOTPRES]`
: "field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fgetalloc`(3)

F_error (3FML)

Name F_error, F_error32—print error message for last error

Synopsis

```
#include stdio.h>
#include "fml.h"
extern int Ferror;
void
F_error(char *msg)
#include "fml32.h"
extern int Ferror32;
void
F_error32(char *msg)
```

Description The function F_error() works like perror(3) for UNIX System errors; that is, it produces a message on the standard error output (file descriptor 2), describing the last error encountered during a call to a system or library function. The argument string *msg* is printed first, then a colon and a blank, then the message and a newline. If *msg* is a null pointer or points to a null string, the colon is not printed. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable Ferror, which is set when errors occur but not cleared when non-erroneous calls are made. In the MS-DOS and OS/2 environments, Ferror is redefined to FMLerror.

To immediately print an error message, F_error() should be called on an error return from another FML function. When the error message is FEUNIX. Uunix_err(3) is called.

F_error32 is used with 32-bit FML.

Return Values F_error() is declared a void and as such does not have return values.

See Also Fintro(3), perror(3) in a UNIX System reference manual
Uunix_err(3)

Fadd (3FML)

Name `Fadd, Fadd32`—add new field occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"
int Fadd(FBFR *fbfr, FLDID fieldid, char *value, FLDLEN len)
#include "fml32.h"
int Fadd32(FBFR32 *fbfr, FLDID32 fieldid, char *value, FLDLEN32
len)
```

Description `Fadd()` adds the specified field value to the given buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *value* is a pointer to a new value; the pointer's type must be the same fieldid type as the value to be added. *len* is the length of the value to be added; it is required only if type is `FLD_CARRAY`

The value to be added is contained in the location pointed to by the *value* parameter. If one or more occurrences of the field already exist, then the value is added as a new occurrence of the field, and is assigned an occurrence number 1 greater than the current highest occurrence (to add a specific occurrence, `Fchg`(3) must be used).

In the `SYNOPSIS` section above the value argument to `Fadd`() is described as a character pointer data type (char * in C). Technically, this describes only one particular kind of value passable to `Fadd`(). In fact, the type of the *value* argument should be a pointer to an object of the same type as the type of the fielded-buffer representation of the field being added. For example, if the field is stored in the buffer as type `FLD_LONG`, then *value* should be of type pointer-to-long (long * in C). Similarly, if the field is stored as `FLD_SHORT`, then *value* should be of type pointer-to-short (short * in C). The important thing is that `Fadd`() assumes that the object pointed to by *value* has the same type as the stored type of the field being added.

For values of type `FLD_CARRAY`, the length of the value is given in the *len* argument.For all types other than `FLD_CARRAY`, the length of the object pointed to by *value* is inferred from its type (e.g. a value of type `FLD_FLOAT` is of length sizeof(float)), and the contents of *len* are ignored.

`Fadd32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors

Under the following conditions, `Fadd`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded" The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FEINVAL]`
"invalid argument to function" One of the arguments to the function invoked was invalid. (For example, specifying a NULL value parameter to `Fadd`.)

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added in a fielded buffer but there is not enough space remaining in the buffer.

`[FBADFLD]`
"unknown field number or type"
A field number is specified which is not valid.

See Also

`Fintro`(3fml)
`CFadd`(3fml)
`Fadds`(3fml)
`Fchg`(3fml)

Fadds (3FML)

Name `Fadds, Fadds32`—convert value from type FLD_STRING and add to buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fadds(FBFR *fbfr, FLDID fieldid, char *value)
#include "fml32.h"
int
Fadds32(FBFR32 *fbfr, FLDID32 fieldid, char *value)
```

Description `Fadds()` has been provided to handle the case of conversion from a user type of `FLD_STRING` to the field type of *`fieldid`* and add it to the fielded buffer. *`fbfr`* is a pointer to a fielded buffer. *`fieldid`* is a field identifier. *value* is a pointer to the value to be added.

This function calls `CFadd` providing a `type` of `FLD_STRING`, and a `len` of 0.

`Fadds32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fadds`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added in a fielded buffer but there is not enough space remaining in the buffer.

`[FTYPERR]`
"invalid field type"
A field type is specified which is not valid.

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, specifying a `NULL` *value* parameter to `Fadds`)

`[FMALLOC]`
"`malloc` failed"
Allocation of space dynamically using `malloc`(3) failed during conversion of carray to string.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fchgs`(3), `Fgets`(3), `Fgetsa`(3), `Ffinds`(3), `CFchg`(3), `CFget`(3), `CFget`(3), `CFfind`(3)

Falloc (3FML)

Name `Falloc, Falloc32`—allocate and initialize fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
FBFR *
Falloc(FLDOCC F, FLDLEN V)
#include "fml32.h"
FBFR32 *
Falloc32(FLDOCC32 F, FLDLEN32 V)
```

Description `Falloc()` dynamically allocates space using `malloc`(3) for a fielded buffer and calls `Finit()` to initialize it. The parameters are the number of fields, *F*, and the number of bytes of value space, *V*, for all fields that are to be stored in the buffer.

`Falloc32` is used for larger buffers with more fields.

Return Values This function returns NULL on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Falloc`() fails and sets `Ferror` to:

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, number of fields is less than 0, *V* is 0 or total size is greater than 65534).

See Also `Fintro`(3), `Ffree`(3), `Fielded`(3), `Finit`(3), `Fneeded`(3), `Frealloc`(3), `Fsizeof`(3), `Funused`(3), `malloc`(3)

Fappend (3FML)

Name Fappend, Fappend32—append new field occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fappend(FBFR *fbfr, FLDID fieldid, char *value, FLDLEN len)
#include "fml32.h"
int
Fappend32(FBFR32 *fbfr, FLDID32 fieldid, char *value, FLDLEN32 len)
```

Description Fappend() adds the specified field value to the end of thegiven buffer. Fappend() is useful in building large buffers in that it does not maintain the internal structures and ordering necessary for general purpose FML access. The side effect of this optimization is that a call to Fappend() may be followed only by additional calls to Fappend(), calls to the FML indexing routines Findex(3) and Funindex(3), or calls to Free(3), Fused(3), Funused(3) and Fsizeof(3). Calls to other FML routines made before calling Findex(3) or Funindex(3) will result in an error with Ferror set to FNOTFLD.

fbfr is a pointer to a fielded buffer. *fieldid* is a field identifier. *value* is a pointer to a new value;the pointer's type must be the same fieldid type as the value to be added. *len* is the length of the value to be added; it is required only if type is FLD_CARRAY

The value to be added is contained in the location pointed toby the *value* parameter. If one or more occurrences of the field already exist, then the value is added as a new occurrence of the field, and is assigned an occurrence number 1 greater than the current highest occurrence (to add a specific occurrence, Fchg(3) must be used).

In the SYNOPSIS section abovethe *value* argument to Fappend() is described as a character pointer data type (char * in C). Technically, this describes only one particular kind of value passable to Fappend(). In fact, the type of the *value* argument should be a pointer to an object of the same type as the type of the fielded-buffer representation of the field being added. For example, if the field is stored in the buffer as type FLD_LONG, then *value* should be of type pointer-to-long (long * in C). Similarly, if the field is stored as FLD_SHORT, then value should be of type pointer-to-short (short * in C). The important thing is that Fappend() assumes that the object pointed to by *value* has the same type as the stored type of the field being added.

For values of type FLD_CARRAY,the length of the value is given in the *len* argument. For all types other than FLD_CARRAY, the length of the object pointed to by *value* is inferred from its type (e.g. a value of type FLD_FLOAT is of length sizeof(float)), and the contents of *len* are ignored.

Fappend32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror`to indicate the error condition.

Errors Under the following conditions, `Fappend`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid. (for example, specifying a `NULL` *value* parameter to `Fappend`)

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added in a fielded buffer but there is not enough space remaining in the buffer.

`[FBADFLD]`
"unknown field number or type"
A field number is specified which is not valid.

See Also `Fintro`(3), `Fadd`(3, `Ffree`(3), `Findex`(3), `Fsizeof`(3), `Funindex`(3), `Funused`(3), `Fused`(3)

Fboolco (3FML)

Name Fboolco, Fboolco32—compile expression, return evaluation tree

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Fboolco(char *expression)
#include "fml32.h"
char *
Fboolco32(char *expression)
```

Description Fboolco() compiles a Boolean expression, pointed to by *expression*, and returns a pointer to the evaluation tree. The expressions recognized are close to the expressions recognized in C. A description of the grammer can be found in the *FML Programmer's Guide*.

The evaluation tree produced by Fboolco() is used by the other boolean functions listed under SEE ALSO; this avoids having to recompile the expression.

Fboolco32 is used with 32-bit FML.

These functions are not supported on Workstation platforms.

Return Values This function returns NULL on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fboolco() fails and sets Ferror to:

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed.

[FSYNTAX]
"bad syntax in Boolean expression"
A syntax error was found in a Boolean expression by Fboolco() other than an unrecognized field name.

[FBADNAME]
"unknown field name"
A field name is specified which cannot be found in the field tables.

[FEINVAL]
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, *expression* is NULL).

Example

```
#include "stdio.h"
#include "fml.h"
extern char *Fboolco();
char *tree;
...
if((tree=Fboolco("FIRSTNAME %% 'J.*n' & SEX = 'M'")) == NULL)
 F_error("pgm_name");
```

compiles a boolean expression that checks if the `FIRSTNAME` field is in the buffer, begins with 'J' and ends with 'n' (for example, John, Jean, Jurgen, etc.) and the `SEX` field equal to `'M'`.

The first and second characters of the tree array form the least significant byte and the most significant byte, respectively, of an unsigned 16 bit quantity that gives the length, in bytes, of the entire array. This value is useful for copying or otherwise manipulating the array.

See Also `Fboolev`(3), `Fboolpr`(3), `Fldid`(3)

Fboolev (3FML)

Name Fboolev, Fboolev32—evaluate buffer against tree

Synopsis

```
#include stdio.h>
#include "fml.h"
int
Fboolev(FBFR *fbfr, char *tree)
#include "fml32.h"
int
Fboolev32(FBFR32 *fbfr, char *tree)
```

Description Fboolev() takes a pointer to a fielded buffer, *fbfr*, and a pointer to the evaluation tree returned from Fboolco(), *tree*, and returns true (1) if the fielded buffer matches the specified Boolean conditions and false (0) if it does not. This function does not change either the fielded buffer or evaluation tree. The evaluation tree is one previously compiled by Fboolco(3).

Fboolev32 is used with 32-bit FML.

These functions are not supported on Workstation platforms.

Return Values Fboolev() returns 1 if the expression in the buffer matches the evaluation tree. It returns 0 if the expression fails to match the evaluation tree. This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fboolev() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The *fbfr* buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The *fbfr* buffer is not a fielded buffer or has not been initialized by Finit().

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed.

[FEINVAL]
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, specifying a NULL tree parameter).

[FSYNTAX]
"bad syntax in Boolean expression"
A syntax error was found in a Boolean expression other than an unrecognized field name.

Example Using the evaluation tree compiled in the example for Fboolco(3):

```
#include stdio.h>
#include "fml.h"
#include "fld.tbl.h"
FBFR *fbfr;
...
Fchg(fbfr,FIRSTNAME,0,"John",0);
Fchg(fbfr,SEX,0,"M",0);
if(Fboolev(fbfr,tree) > 0)
 fprintf(stderr,"Buffer selected\\\\n");
else
 fprintf(stderr,"Buffer not selected\\\\n");
```

would print "Buffer selected".

See Also Fintro(3), Fboolco(3), Fboolpr(3)

Fboolpr (3FML)

NAME

`Fboolpr, Fboolpr32`—print Boolean expression as parsed

Synopsis

```
#include <stdio.h>
#include "fml.h"
void
Fboolpr(char *tree, FILE *iop)
#include "fml32.h"
void
Fboolpr32(char *tree, FILE *iop)
```

Description

`Fboolpr()` prints a compiled expression to the specified output stream. The evaluation tree, *tree*, is one previously created with `Fboolco`(3). *iop* is a pointer of type `FILE` to the output stream. The output is fully parenthesized, as it was parsed (as indicated by the evaluation tree). The function is useful for debugging.

`Fboolpr32` is used with 32-bit FML.

These functions are not supported on Workstation platforms.

Return Values

`Fboolpr`() is declared as returning a `void`, so there are no return values.

See Also

`Fintro`(3), `Fboolco`(3)

Fchg (3FML)

Name Fchg, Fchg32—change field occurrence value

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fchg(FBFR *fbfr, FLDID fieldid, FLDOCC oc, char *value, FLDLEN len)
#include "fml32.h"
int
Fchg32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, char *value,
 FLDLEN32 len)
```

Description Fchg() changes the value of a field in the buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. value is a pointer to a new value, its type must be the same type as the value to be changed (see below). *len* is the length of the value to be changed; it is required only if field type is FLD_CARRAY.

If an occurrence of -1 is specified, then the field value is added as a new occurrence to the buffer. If the specified field occurrence is found, then the field value is modified to the value specified. If a field occurrence is specified that does not exist, then NULL values are added for the missing occurrences until the desired occurrence can be added (for example, changing field occurrence 4 for a field that does not exist on a buffer will cause 3 NULL values to be added followed by the specified field value). NULL values consist of the NULL string (1 byte in length) for string and character values, 0 for long and short fields, 0.0 for float and double values, and a zero-length string for a character array. The new or modified value is contained in *value* and its length is given in *len* if it is a character array (ignored in other cases). If *value* is NULL, then the field occurrence is deleted. A value to be deleted that is not found, is considered an error.

In the SYNOPSIS section above the *value* argument to Fchg() is described as a character pointer data type (char * in C). Technically, this describes only one particular kind of value passable to Fchg(). In fact, the type of the *value* argument should be a pointer to an object of the same type as the type of the fielded-buffer representation of the field being changed. For example, if the field is stored in the buffer as type FLD_LONG, then *value* should be of type pointer-to-long (long * in C). Similarly, if the field is stored as FLD_SHORT, then value should be of type pointer-to-short (short * in C). The important thing is that Fchg() assumes that the object pointed to by *value* has the same type as the stored type of the field being changed.

Fchg32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, `Fchg`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOTPRES]`
"field not present"
A field occurrence is requested for deletion but the specified field and/or occurrence was not found in the fielded buffer.

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added or changed in a fielded buffer but there is not enough space remaining in the buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `CFchg`(3c)
`Fintro`(3fml)
`Fadd`(3fml)
`Fcmp`(3fml)
`Fdel`(3fml)

Fchgs (3FML)

Name Fchgs, Fchgs32—change field occurrence - caller presents string

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fchgs(FBFR *fbfr, FLDID fieldid, FLDOCC oc, char *value)
#include "fml32.h"
int
Fchgs32(FBFR32 *fbfr, FLDID32 fieldid, int oc, char *value)
```

Description Fchgs(), is provided to handle the case of conversion from a user type of FLD_STRING. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. *value* is a pointer to the string to be added. The function calls its non-string-function counterpart, CFchg(3), providing a type of FLD_STRING, and a len of 0 to convert from a string to the field type of *fieldid*.

Fchgs32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fchgs() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FNOSPACE]
"no space in fielded buffer"
A field value is to be added or changed in a fielded buffer but there is not enough space remaining in the buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
"invalid field type"
A field identifier is specified which is not valid.

See Also Fintro(3), Fchg(3), CFchg(3)

Fchksum (3FML)

Name Fchksum, Fchksum32—compute checksum for fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
long
Fchksum(FBFR *fbfr)
#include "fml32.h"
long
Fchksum32(FBFR32 *fbfr)
```

Description For extra-reliable I/O, a checksum may be calculated using Fchksum() and stored in a fielded buffer being written out. *fbfr* is a pointer to a fielded buffer. The stored checksum may be inspected by the receiving process to verify that the entire buffer was received.

Fchksum32 is used with 32-bit FML.

Return Values On success, Fchksum returns the checksum. This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fchksum() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

See Also Fintro(3), Fread(3), Fwrite(3)

Fcmp (3FML)

Name Fcmp, Fcmp32—compare two fielded buffers

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fcmp(FBFR *fbfr1, FBFR *fbfr2)
#include "fml32.h"
int
Fcmp32(FBFR32 *fbfr1, FBFR32 *fbfr2)
```

Description Fcmp() compares the field identifiers and then the field values of two FML buffers. *fbfr1* and *fbfr2* are pointers to the fielded buffers to be compared.

Fcmp32 is used with 32-bit FML.

Return Values The function returns a 0 if the two buffers are identical. It returns a -1 on any of the following conditions:

- The fieldid of a *fbfr1* field is less than the fieldid of the corresponding field of *fbfr2*.
- The value of a field in *fbfr1* is less than the value of the corresponding field of *fbfr2*.
- *fbfr1* has fewer fields or field occurrences than *fbfr2*.

Fcmp(\|) returns a 1 if any of the reverse set of conditions is true, for example, the fieldid of a *fbfr1* field is greater than the fieldid of the corresponding field of *fbfr2*. The actual sizes of the buffers (that is, the sizes passed to Falloc()) are not considered; only the data in the buffers. This function returns \-2 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fcmp() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

See Also Fintro(3), Fadd(3), Fchg(3)

Fconcat (3FML)

Name Fconcat, Fconcat32—concatenate source to destination buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fconcat(FBFR *dest, FBFR *src)
#include "fml32.h"
int
Fconcat32(FBFR32 *dest, FBFR32 *src)
```

Description Fconcat() adds fields from the source buffer to the fields that already exist in the destination buffer. *dest* and *src* are pointers to the destination and source fielded buffers, respectively. Occurrences in the destination buffer, if any, are maintained and new occurrences from the source buffer are added with greater occurrence numbers for the field.

Fconcat32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fconcat() fails and sets Ferror to:

[FALIGNERR]
: "fielded buffer not aligned"
Either the source buffer or the destination buffer does not begin on the proper boundary.

[FNOTFLD]
: "buffer not fielded"
Either the source or the destination buffer is not a fielded buffer or has not been initialized by Finit().

[FNOSPACE]
: "no space in fielded buffer"
A field value is to be added in a fielded buffer but there is not enough space remaining in the buffer.

See Also Fintro(3), Fjoin(3), Fupdate(3)

Fcpy (3FML)

Name `Fcpy, Fcpy32`—copy source to destination buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fcpy(FBFR *dest, FBFR *src)
#include "fml32.h"
int
Fcpy32(FBFR32 *dest, FBFR32 *src)
```

Description `Fcpy()` is used to copy the contents of one fielded buffer to another fielded buffer. *dest* and *src* are pointers to the destination and source fielded buffers respectively. `Fcpy()` expects the destination to be a fielded buffer, and thus can check that it is large enough to accommodate the data from the source buffer.

`Fcpy32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fcpy`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
Either the source buffer or the destination buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
Either the source or the destination buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOSPACE]`
"no space in fielded buffer"
The destination buffer is not large enough to hold the source buffer.

See Also `Fintro`(3), `Fmove`(3)

Fdel (3FML)

Name `Fdel, Fdel32`—delete field occurrence from buffer

Synopsis

```
#include stdio.h>
#include "fml.h"
int
Fdel(FBFR *fbfr, FLDID fieldid, FLDOCC oc)
#include "fml32.h"
int
Fdel32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc)
```

Description `Fdel()` deletes the specified field occurrence from the buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field.

Note that when multiple occurrences of a field exist in the fielded buffer and a field occurrence is deleted that is not the last occurrence, also higher occurrences in the buffer are shifted down by one. To maintain the same occurrence number for all occurrences, use `Fchg`(3) to set the field occurrence value to a "null" value.

`Fdel32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fdel`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOTPRES]`
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fadd`(3), `Fchg`(3), `Fdelall`(3), `Fdelete`(3)

Fdelall (3FML)

Name `Fdelall, Fdelall32`—delete all field occurrences from buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fdelall(FBFR *fbfr, FLDID fieldid)
#include "fml32.h"
int
Fdelall32(FBFR32 *fbfr, FLDID32 fieldid)
```

Description `Fdelall()` deletes all occurrences of the specified field in the buffer. *`fbfr`* is a pointer to a fielded buffer. *`fieldid`* is a field identifier. If no occurrences of the field are found, it is considered an error.

`Fdelall32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fdelall`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOTPRES]`
"field not present"
A field is requested but the specified field was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fdel`(3), `Fdelete`(3)

Fdelete (3FML)

Name `Fdelete, Fdelete32`-delete list of fields from buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fdelete(FBFR *fbfr, FLDID *fieldid)
#include "fml32.h"
int
Fdelete32(FBFR32 *fbfr, FLDID32 *fieldid)
```

Description `Fdelete`() deletes all occurrences of all fields listed in the array of field identifiers, *fieldid*[]. The last entry in the array must be `BADFLDID`. *fbfr* is a pointer to a fielded buffer. *fieldid* is a pointer to an array of field identifiers. This is a more efficient way of deleting several fields from a buffer instead of using several `Fdelall`() calls. The update is done in-place. The array of field identifiers may be re-arranged by `Fdelete`() (they are sorted, if not already, in numeric order).

Fdelete() returns success even if no fields are deleted from the fielded buffer.

`Fdelete32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fdelete`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fdel`(3), `Fdelall`(3)

Fextread (3FML)

Name `Fextread, Fextread32`-build fielded buffer from printed format

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fextread(FBFR *fbfr, FILE *iop)
#include "fml32.h"
int
Fextread32(FBFR32 *fbfr, FILE *iop)
```

Description `Fextread`() may be used to construct a fielded buffer from its printed format (that is, from the output of `Fprint`(3)). The parameters are a pointer to a fielded buffer, *fbfr*, and a pointer to a file stream, *iop*. The input file format is basically the same as the output format of Fprint(3), that is:

```
[flag] fldname or fldid tab> fldval (or fldname, if flag is ''='')
```

The optional flags and their meanings are as follows:

+
: occurrence 0 of the field in the fielded buffer should be changed to the value provided.

\-
: occurrence 0 of the field named should be deleted from the fielded buffer. The tab character is required; any field value is ignored.

=
: In this case, the last field on the input line is the name of a field in the fielded buffer. The value of occurrence 0 of that field should be assigned to occurrence 0 of the first field named on the input line.

#
: the line is treated as a comment and is ignored.

If no *flag* is specified, a new occurrence of the field named by *fldname* with value *fldval* is added to the fielded buffer. A trailing newline (`-`) must be provided following each completed input buffer.

`Fextread32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fextread()` fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added or changed in a field buffer but there is not enough space remaining in the buffer.

`[FBADFLD]`
"unknown field number or type"
A field number is specified which is not valid.

`[FEUNIX]`
"UNIX system call error"
A UNIX system call error occurred. The external integer `errno` should have been set to indicate the error by the system call, and the external integer `Uunixerr` (values defined in `Uunix.h`) is set to the system call that returned the error.

`[FBADNAME]`
"unknown field name"
A field name is specified which cannot be found in the field tables.

`[FSYNTAX]`
"bad syntax in format"
A syntax error was found in the external buffer format. Possible errors are: an unexpected end-of-file indicator, input lines not in the form `fieldid or name tab> value` two control characters, field values greater than 1000 characters, or an invalid hex escape sequence.

`[FNOTPRES]`
"field not present"
A field to be deleted is not found in the fielded buffer.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

`[FEINVAL]`
"invalid parameter"
The value of *iop* is NULL.

See Also `Fintro`(3), `Fprint`(3)

Ffind (3FML)

Name `Ffind, Ffind32`-find field occurrence in buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Ffind(FBFR *fbfr, FLDID fieldid, FLDOCC oc, FLDLEN *len)
#include "fml32.h"
char *
Ffind32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, FLDLEN32 *len)
```

Description `Ffind()` finds the value of the specified field occurrence in the buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. If the field is found, its length is set into **len*, and its location is returned as the value of the function. If the value of *len* is NULL, then the field length is not returned. `Ffind()` is useful for gaining read-only access to a field. In no case should the value returned by `Ffind()` be used to modify the buffer.

In general, the locations of values of types `FLD_LONG`, `FLD_FLOAT`, and `FLD_DOUBLE` are not suitable for direct use as their stored type, since proper alignment within the buffer is not guaranteed. Such values must be copied first to a suitably aligned memory location. Accessing such fields through the conversion function `CFfind`(3) does guarantee the proper alignment of the found converted value. Buffer modification should only be done by the functions `Fadd`(3) or `Fchg`(3). The values returned by `Ffind()` and `Ffindlast()` are valid only so long as the buffer remains unmodified.

`Ffind32` is used with 32-bit FML.

Return Values In the SYNOPSIS section above the return value to `Ffind()` is described as a character pointer data type (char * in C). Actually, the pointer returned points to an object that has the same type as the stored type of the field.

This function returns a pointer to NULL on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Ffind`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FNOTPRES]`
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3fml), `Ffindlast`(3fml), `Ffindocc`(3fml), `Ffinds`(3fml)

Ffindlast (3FML)

Name `Ffindlast, Ffindlast32`-find last occurrence of field in buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Ffindlast(FBFR *fbfr, FLDID fieldid, FLDOCC *oc, FLDLEN *len)
#include "fml32.h"
char *
Ffindlast32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 *oc, FLDLEN32
*len)
```

Description `Ffindlast`() finds the last occurrence of a field in a buffer. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is a pointer to an integer that is used to receive the occurrence number of the field. *len* is the length of the value. If there are no occurrences of the field in the buffer, NULL is returned. Generally, `Ffindlast`() acts like Ffind(3). The major difference is that with `Ffindlast` the user does not supply a field occurrence. Instead, both the value and occurrence number of the last occurrence of the field are returned. In order to return the occurrence number of the last field, the occurrence argument, *oc*, to `Ffindlast`() is a pointer-to-integer, and not an integer, as it is to `Ffind()`. If *oc* is specified to be NULL, the occurrence number of the last occurrence is not returned. If the value of *len* is NULL, then the field length is not returned.

In general, the locations of values of types `FLD_LONG`, `FLD_FLOAT`, and `FLD_DOUBLE` are not suitable for direct use as their stored type, since proper alignment within the buffer is not guaranteed. Such values must be copied first to a suitably aligned memory location. Accessing such fields through the conversion function CFfind(3) does guarantee the proper alignment of the found converted value. Buffer modification should only be done by the functions Fadd(3) or Fchg(3). The values returned by `Ffind`() and `Ffindlast`() are valid only so long as the buffer remains unmodified.

`Ffindlast32` is used with 32-bit FML.

Return Values In the SYNOPSIS section above the return value to `Ffindlast`() is described as a character pointer data type (char * in C). Actually, the pointer returned points to an object that has the same type as the stored type of the field.

This function returns NULL on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Ffindlast`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FNOTPRES]`
"field not present"
A field is requested but the specified field was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3fml), `CFfind`(3fml), `Fadd`(3fml), `Fchg`(3fml), `Ffind`(3fml), `Ffindocc`(3fml), `Ffinds`(3fml)

Ffindocc (3FML)

Name Ffindocc, Ffindocc32-find occurrence of field value

Synopsis

```
#include <stdio.h>
#include "fml.h"
FLDOCC
Ffindocc(FBFR *fbfr, FLDID fieldid, char *value, FLDLEN len)
#include "fml32.h"
FLDOCC32
Ffindocc32(FBFR32 *fbfr, FLDID32 fieldid, char *value, FLDLEN32
len)
```

Description Ffindocc() looks at occurrences of the specified field in the buffer and returns the occurrence number of the first field occurrence that matches the user specified field value. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. The value to be found is contained in the location pointed to by the *value* parameter. *len* is the length of the value if its type is FLD_CARRAY. If *fieldid* is field type FLD_STRING and if *len* is not 0, pattern matching is done on the string. The pattern match supported is the same as the patterns described in regcmp(3) (in UNIX reference manuals). In addition, the alternation of regular expressions is supported (for example, ''A|B" matches with ''A" or ''B"). The pattern must match the entire field value (that is, the pattern ''value" is implicitly treated as ''^value$"). The version of Ffindocc() provided for use in the MS-DOS and OS/2 environments does not support the regcmp(3) pattern matching for FLD_STRING fields; it uses strcmp(3) (in UNIX reference manuals).

In the SYNOPSIS section above the value argument to Ffindocc() is described as a character pointer data type (char * in C). Technically, this describes only one particular kind of value passable to Ffindocc(). In fact, the type of the value argument should be a pointer to an object of the same type as the type of the fielded-buffer representation of the field being found. For example, if the field is stored in the buffer as type FLD_LONG, then value should be of type pointer-to-long (long * in C). Similarly, if the field is stored as FLD_SHORT, then value should be of type pointer-to-short (short * in C). The important thing is that Ffindocc() assumes that the object pointed to by value has the same type as the stored type of the field being found.

Ffindocc32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, `Ffindocc`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FNOTPRES]`
"field not present"
A field value is requested but the specified field and/or value was not found in the fielded buffer.

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, passing a NULL value parameter to `Ffindocc` or specifying an invalid string pattern).

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3fml), `Ffind`(3fml), `Ffindlast`(3fml), `Ffinds`(3fml), `regcmp`(3) in a UNIX System reference manual

Ffinds (3FML)

Name `Ffinds, Ffinds32`-return ptr to string representation

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Ffinds(FBFR *fbfr, FLDID fieldid, FLDOCC oc)
#include "fml32.h"
char *
Ffinds32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc)
```

Description `Ffinds`() is provided to handle the case of conversion to a user type of `FLD_STRING`. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. The specified field occurrence is found and converted from its type in the buffer to a null-terminated string. Basically, this macro calls its conversion function counterpart, CFfind(3), providing a *utype* of `FLD_STRING`, and a *ulen* of 0. The duration of the validity of the pointer returned by `Ffinds`() is the same as that described for CFfind(3).

`Ffinds32` is used with 32-bit FML.

Return Values This function returns NULL on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Ffinds`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FNOTPRES]`
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

`[FTYPERR]`
"invalid field type"
A field type is specified which is not valid.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed while converting carray to string.

See Also `Fintro`(3), `CFfind`(3), `Ffind`(3)

Ffloatev (3FML)

Name `Ffloatev, Ffloatev32`-return value of expression as a double

Synopsis

```
#include <stdio.h>
#include "fml.h"
double
Ffloatev(FBFR *fbfr, char *tree)
#include "fml32.h"
double
Ffloatev32(FBFR32 *fbfr, char *tree)
```

Description `Ffloatev`() takes a pointer to a fielded buffer, *fbfr*, and a pointer to the evaluation tree returned from Fboolco(3), *tree*, and returns the value of the (arithmetic) expression, represented by the tree, as a double. This function does not change either the fielded buffer or the evaluation tree.

`Ffloatev32` is used with 32-bit FML.

These functions are not supported on /WS platforms.

Return Values On success Ffloatev() returns the value of an expression as a double.

This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, Ffloatev() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

`[FSYNTAX]`
"bad syntax in Boolean expression"
A syntax error was found in a Boolean expression tree.

See Also `Fintro`(3), `Fboolco`(3), `Fboolev`(3)

Ffprint (3FML)

Name `Ffprint, Ffprint32`-print fielded buffer to specified stream

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Ffprint(FBFR *fbfr, FILE *iop)
#include "fml32.h"
int
Ffprint32(FBFR32 *fbfr, FILE *iop)
```

Description `Ffprint` is similar to Fprint(3), except the text is printed to a specified output stream. *fbfr* is a pointer to a fielded buffer. *iop* is a pointer of type `FILE` that points to the output stream.

For each field in the buffer, the output prints the field name and field value separated by a tab. Fname(3) is used to determine the field name; if the field name cannot be determined, then the field identifier is printed. Non-printable characters in string and character array field values are represented by a backslash followed by their two-character hexadecimal value. A newline is printed following the output of the printed buffer.

`Ffprint32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, Ffprint() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit()`.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

See Also `Fintro`(3), `Fprint`(3)

Ffree (3FML)

Name `Ffree, Ffree32`-free space allocated for fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Ffree(FBFR *fbfr)
#include "fml32.h"
int
Ffree32(FBFR32 *fbfr)
```

Description `Ffree`() is used to recover space allocated to its argument fielded buffer. *fbfr* is a pointer to a fielded buffer. The fielded buffer is invalidated, that is, made non-fielded, and then freed.

Ffree() is recommended as opposed to free(3) (in UNIX System reference manuals), because Ffree() invalidates a fielded buffer whereas `free(3)` does not. It is important to invalidate fielded buffers because `malloc`(3) (in UNIX System reference manuals) re-uses memory that has been freed without clearing it. Thus, if `free(3)` were used, it would be possible for `malloc` to return a piece of memory that looks like a valid fielded buffer but is not.

`Ffree32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Ffree`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned" The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded" The buffer is not a fielded buffer or has not been initialized by `Finit`().

See Also `Fintro`(3), `malloc`(3), `free`(3) in UNIX reference manuals, `Falloc`(3), `Frealloc`(3)

Fget (3FML)

Name `Fget, Fget32`-get copy and length of field occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fget(FBFR *fbfr, FLDID fieldid, FLDOCC oc, char *value, FLDLEN
  *maxlen)
#include "fml32.h"
int
Fget32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, char *value,
  FLDLEN32 *maxlen)
```

Description `Fget`() should be used to retrieve a field from a fielded buffer when the value is to be modified. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. The caller provides `Fget`() with a pointer to a private data area, *loc*, as well as the length of the data area, **maxlen*, and the length of the field is returned in **maxlen*. If *maxlen* is NULL when the function is called, then it is assumed that the data area for the field value *loc* is big enough to contain the field value and the length of the value is not returned. If *loc* is NULL, the value is not retrieved. Thus, the function call can be used to determine the existence of the field.

In the SYNOPSIS section above the value argument to `Fget`() is described as a character pointer data type (char * in C). Technically, this describes only one particular kind of value passable to `Fget`(). In fact, the type of the value argument should be a pointer to an object of the same type as the type of the fielded-buffer representation of the field being retrieved. For example, if the field is stored in the buffer as type `FLD_LONG`, then value should be of type pointer-to-long (long * in C). Similarly, if the field is stored as `FLD_SHORT`, then value should be of type pointer-to-short (short * in C). The important thing is that `Fget`() assumes that the object pointed to by value has the same type as the stored type of the field being retrieved.

`Fget32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fget`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

`[FNOSPACE]`
"no space"
The size of the data area, as specified in maxlen, is not large enough to hold the field value.

`[FNOTPRES]`
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

See Also `Fintro`(3fml), `CFget`(3c), `Fgetalloc`(3fml), `Fgetlast`(3fml),`Fgets`(3fml), `Fgetsa`(3fml)

Fgetalloc (3FML)

Name Fgetalloc, Fgetalloc32-allocate space and get copy of field occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Fgetalloc(FBFR *fbfr, FLDID fieldid, FLDOCC oc, FLDLEN *extralen)
#include "fml32.h"
char *
Fgetalloc32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, FLDLEN32
  *extralen)
```

Description Like Fget(3), Fgetalloc() finds and makes a copy of a buffer field, but it acquires space for the field via a call to malloc(3) (in UNIX System programmer's reference manuals). *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. The last argument to Fgetalloc(), *extralen*, provides an extra amount of space to be acquired in addition to the field value size. It can be used if the retrieved value is to be expanded before re-insertion into the fielded-buffer. If *extralen* is NULL, then no additional space is allocated and the actual length is not returned. It is the caller's responsibility to free(3) space acquired by Fgetalloc(). The buffer will be aligned properly for any field type.

Fgetalloc32 is used with 32-bit FML.

Return Values In the SYNOPSIS section above the return value to Fgetalloc() is described as a character pointer data type (char * in C). Actually, the pointer returned points to an object that has the same type as the stored type of the field. This function returns NULL on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fgetalloc() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

`[FMALLOC]`
“malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

See Also `Fintro`(3fml), `CFget`(3c), `Fget`(3fml), `Fgetlast`(3fml), `Fgets`(3fml), `Fgetsa`(3fml) `free`(3), `malloc`(3) in a UNIX System reference manual

Fgetlast (3FML)

Name `Fgetlast, Fgetlast32`-get copy of last occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fgetlast(FBFR *fbfr, FLDID fieldid, FLDOCC *oc, char *value,  FLDLEN
  *maxlen)
#include "fml32.h"
int
Fgetlast32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 *oc, char
  *value,  FLDLEN32 *maxlen)
```

Description `Fgetlast`() is used to retrieve both the value and occurrence number of the last occurrence of the field identified by *fieldid*. *fbfr* is a pointer to a fielded buffer. In order to return the occurrence number of the last field, the occurrence argument, *oc*, is a pointer-to-integer, not an integer.

The caller provides `Fgetlast()` with a pointer to a private buffer, *loc*, as well as the length of the buffer, **maxlen*, and the length of the field is returned in **maxlen*. If *maxlen* is NULL when the function is called, then it is assumed that the buffer for the field value is big enough to contain the field value and the length of the value is not returned. If *loc* is NULL, the value is not returned. If *oc* is NULL, the occurrence is not returned.

In the SYNOPSIS section above the value argument to `Fgetlast`() is described as a character pointer data type (char * in C). Technically, this describes only one particular kind of value passable to `Fgetlast`(). In fact, the type of the value argument should be a pointer to an object of the same type as the type of the fielded-buffer representation of the field being retrieved. For example, if the field is stored in the buffer as type `FLD_LONG`, then value should be of type pointer-to-long (long * in C). Similarly, if the field is stored as `FLD_SHORT`, then value should be of type pointer-to-short (short * in C). The important thing is that `Fgetlast`() assumes that the object pointed to by value has the sametype as the stored type of the field being retrieved.

`Fgetlast32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fgetlast`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOSPACE]`
"no space"
The size of the data area, as specified in maxlen, is not large enough to hold the field value.

`[FNOTPRES]`
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
|A field identifier is specified which is not valid.

See Also `Fintro`(3fml), `Fget`(3fml), `Fgetalloc`(3fml), `Fgets`(3fml), `Fgetsa`(3fml)

Fgets (3FML)

Name Fgets, Fgets32-get value converted to string

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Fgets(FBFR *fbfr, FLDID fieldid, FLDOCC oc, char *buf)
#include "fml32.h"
int
Fgets32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, char *buf)
```

Description Fgets() retrieves a field occurrence from the fielded buffer first converting the value to a user type of FLD_STRING. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. The caller of Fgets() provides *buf*, a pointer to a private buffer, which is used for the retrieved field value. It is assumed that *buf* is large enough to hold the value. Basically, Fgets() calls CFget(3) with an assumed *utype* of FLD_STRING, and a *ulen* of 0.

Fgets32 is used with 32-bit FML.

Return Values This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fgets() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

`[FTYPERR]`
"invalid field type"
A field identifier is specified which is not valid.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

See Also `Fintro`(3), `CFget`(3), `Fget`(3), `Fgetalloc`(3), `Fgetlast`(3), `Fgetsa`(3)

Fgetsa (3FML)

Name `Fgetsa, Fgetsa32`-malloc space and get converted value

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Fgetsa(FBFR *fbfr, FLDID fieldid, FLDOCC oc, FLDLEN *extra)
#include "fml32.h"
char *
Fgetsa32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, FLDLEN32
  *extra)
```

Description `Fgetsa`() is a macro that calls `CFgetalloc`(3). *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field. The function uses malloc(3) (in UNIX System programmer's reference manuals) to allocate space for the retireved field value that has been converted to a string. If *extra* is not NULL, it specifies the extra space to allocate in addition to the field value size; the total size is returned in *extra*.

It is the responsibility of the user to free(3) (in UNIX System reference manuals) the space `malloc`'d.

`Fgetsa32` is used with 32-bit FML.

Return Values On success, the function returns a pointer to the allocated buffer.

This function returns NULL on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fgetsa`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOTPRES]`
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

`[FTYPERR]`
"invalid field type"
A field identifier is specified which is not valid.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

See Also `Fintro`(3), `malloc`(3), `free`(3) in UNIX System reference manuals, `CFget`(3), `Fget`(3), `Fgetlast`(3), `Fgets`(3),

Fidnm_unload (3FML)

Name `Fidnm_unload, Fidnm_unload32`-recover space from id->nm mapping tables

Synopsis

```
#include <stdio.h>
#include "fml.h"
void
Fidnm_unload(void);
#include "fml32.h"
void
Fidnm_unload32(void);
```

Description `Fidnm_unload`() recovers space allocated by `Fname`(3) for field identifier to field name mapping tables.

`Fidnm_unload32` is used with 32-bit FML.

Return Values This function is declared as a void and so does not return anything.

See Also `Fintro`(3), `Fname`(3), `Fnmid_unload`(3)

Fidxused (3FML)

Name `Fidxused, Fidxused32`-return amount of space used

Synopsis

```
#include <stdio.h>
#include "fml.h"
long
Fidxused(FBFR *fbfr)
#include "fml32.h"
long
Fidxused32(FBFR32 *fbfr)
```

Description `Fidxused`() indicates the current amount of space used by the buffer's index. *fbfr* is a pointer to a fielded buffer.

`Fidxused32` is used with 32-bit FML.

Return Values On success, the function returns the amount of space in the buffer used by the index. This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fidxused`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

See Also `Fintro`(3), `Findex`(3), `Frstrindex`(3), `Funused`(3), `Fused`(3)

Fielded (3FML)

Name `Fielded, Fielded32`-return true if buffer is fielded

Synopsis

```
#include stdio.h>
#include "fml.h"
int
Fielded(FBFR *fbfr)
#include "fml32.h"
int
Fielded32(FBFR32 *fbfr)
```

Description `Fielded`() is used to test whether the specified buffer is fielded. *fbfr* is a pointer to a fielded buffer.

`Fielded32` is used with 32-bit FML.

Return Values `Fielded()` returns true (1) if the buffer is fielded. It returns false (0) if the buffer is not fielded and does not set `Ferror` in this case.

See Also `Fintro`(3), `Finit`(3), `Fneeded`(3), `Fsizeof`(3)

Findex (3FML)

Name Findex, Findex32-index a fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Findex(FBFR *fbfr, FLDOCC intvl)
#include "fml32.h"
int
Findex32(FBFR32 *fbfr, FLDOCC32 intvl)
```

Description The function Findex() is called explicitly to index a fielded buffer. *fbfr* is a pointer to a fielded buffer. The second parameter, *intvl*, gives the indexing interval, that is, the ideal separation of indexed fields. If this argument has value 0, then the buffer's current indexing value is used. If the current value itself is 0, the value FSTDXINTVL (defaults to 16) is used. Using an indexing value of 1 will ensure that every field in the buffer is indexed. The size of the index interval and the amount of space allocated to a buffer's index are inversely proportional: the smaller the interval, the more fields are indexed and thus the larger the amount of space used for indexing.

Findex32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Findex() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FNOSPACE]
"no space in fielded buffer"
An ENTRY is to be added to the index but there is not enough space remaining in the buffer.

See Also Fintro(3fml), Fidxused(3fml), Frstrindex(3fml), Funindex(3fml)

Finit (3FML)

Name `Finit, Finit32`-initialize fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
int
Finit(FBFR *fbfr, FLDLEN buflen)
#include "fml32.h"
int
Finit32(FBFR32 *fbfr, FLDLEN32 buflen)
```

Description `Finit()` can be called to initialize a fielded buffer statically. *fbfr* is a pointer to a fielded buffer. *buflen* is the length of the buffer. The function takes the buffer pointer and buffer length, and sets up the internal structure for a buffer with no fields. `Finit()` can also be used to re-initialize a previously used buffer.

`Finit32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Finit`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer pointer is NULL.

`[FNOSPACE]`
"no space in fielded buffer"
The buffer size specified is too small for a fielded buffer.

Example The correct way to re-initialize a buffer to have no fields is: `Finit(fbfr, (FLDLEN)Fsizeof(fbfr));`

See Also `Fintro`(3), `Falloc`(3), `Fneeded`(3), `Frealloc`(3)

Fjoin (3FML)

Name `Fjoin, Fjoin32`-join source into destination buffer

Synopsis

```
#include stdio.h>
#include "fml.h"
int
Fjoin(FBFR *dest, FBFR *src)
#include "fml32.h"
int
Fjoin32(FBFR32 *dest, FBFR32 *src)
```

Description `Fjoin()` is used to join two fielded buffers based on matching fieldid/occurrence. *dest* and *src* are pointers to the destination and source fielded buffers respectively. For fields that match on fieldid/occurrence, the field value is updated in the destination buffer with the value in the source buffer. Fields in the destination buffer that have no corresponding fieldid/occurrence in the source buffer are deleted.

This function may fail due to lack of space if the new values are larger than the old; in this case, the destination buffer will have been modified. However, if this happens, the destination buffer may be re-allocated using `Frealloc`(3) and the `Fjoin`() function repeated. Even if the destination buffer has been partially updated, repeating the function will give the correct results.

`Fjoin32` is used with 32-bit FML.

Return Values This function returns -1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fjoin`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
Either the source buffer or the destination buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
Either the source buffer or the destination buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added or changed in a field buffer but there is not enough space remaining in the buffer.

Example In the following example:

```
FBFR *src, *dest;  ...  if(Fjoin(dest,src)  0)  F_error("pgm_name");
```

if `dest` has fields A, B, and two occurrences of C, and `src` has fields A, C, and D, the resultant `dest` will have source field value A and source field value C.

See Also `Fintro`(3), `Fconcat`(3), `Fojoin`(3), `Fproj`(3), `Fprojcpy`(3), `Frealloc`(3)

Fldid (3FML)

Name `Fldid`, `Fldid32`-map field name to field identifier

Synopsis

```
#include <stdio.h>
#include "fml.h"

FLDID
Fldid(char *name)

#include "fml32.h"

FLDID32
Fldid32(char *name)
```

Description `Fldid`() provides a runtime translation of a field-name to its field identifier and returns a `FLDID` corresponding to its field *name* parameter. The first invocation causes space to be dynamically allocated for the field tables and the tables to be loaded. To recover data space used by the field tables loaded by `Fldid`(), the user may unload the files by a call to the `Fnmid_unload`(3) function.

`Fldid32` is used with 32-bit FML.

Return Values This function returns `BADFLDID` on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fldid`() fails and sets `Ferror` to:

`[FBADNAME]`
"unknown field name"
A field name is specified which cannot be found in the field tables.

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using `malloc`(3) failed.

See Also `Fintro`(3), `malloc`(3) in UNIX System reference manuals, `Fldno`(3), `Fname`(3), `Fnmid_unload`(3)

Fldno (3FML)

Name `Fldno`, `Fldno32`-map field identifier to field number

```
#include <stdio.h>
#include "fml.h"

int
Fldno(FLDID fieldid)

#include "fml32.h"

long
Fldno32(FLDID32 fieldid)
```

Description `Fldno`() accepts a field identifier, *fieldid*, as a parameter and returns the field number contained in the identifier.

`Fldno32` is used with 32-bit FML.

Return Values This function returns the field number and does not return an error.

See Also `Fintro`(3), `Fldid`(3), `Fldtype`(3)

Fldtype (3FML)

Name `Fldtype`, `Fldtype32`-map field identifier to field type

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fldtype(FLDID fieldid)

#include "fml32.h"

int
Fldtype32(FLDID32 fieldid)
```

Description `Fldtype`() accepts a field identifier, *fieldid*, and returns the field type contained in the identifier (an integer), as defined in `fml.h`.

`Fldtype32` is used with 32-bit FML.

Return Values This function returns the field type.

See Also `Fintro`(3), `Fldid`(3), `Fldno`(3)

Flen (3FML)

Name Flen, Flen32-return len of field occurrence in buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Flen(FBFR *fbfr, FLDID fieldid, FLDOCC oc)

#include "fml32.h"

long
Flen32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc)
```

Description Flen() finds the value of the specified field occurrence in the buffer and returns its length. fbfr is a pointer to a fielded buffer. fieldid is a field identifier. oc is the occurrence number of the field.

Flen32 is used with 32-bit FML.

Return Values On success, Flen() returns the field length.

This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Flen() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FNOTPRES]
"field not present"
A field occurrence is requested but the specified field and/or occurrence was not found in the fielded buffer.

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

See Also Fintro(3), Fnum(3), Fpres(3)

Fmkfldid (3FML)

Name `Fmkfldid`, `Fmkfldid32`-make a field identifier

```
#include <stdio.h>
#include "fml.h"

FLDID
Fmkfldid(int type, FLDID num)

#include "fml.h"

FLDID32
Fmkfldid32(int type, FLDID32 num)
```

Description `Fmkfldid`() allows the creation of a valid field identifier from a valid type (as defined in `fml.h`) and a field number. This is useful for writing an application generator that chooses field numbers sequentially, or for recreating a field identifier.

type is a valid type (an integer; see `Fldtype`(3)). *num* is a field number (it should be an unused field number, to avoid confusion with existing fields)

`Fmkfldid32` is used with 32-bit FML.

Return Values This function returns `BADFLDID` on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fmkfldid`() fails and sets `Ferror` to:

`[FBADFLD]`
"unknown field number or type"
A field number is specified which is not valid.

`[FTYPERR]`
"invalid field type"
A field type is specified which is not valid (as defined in `fml.h`).

See Also `Fintro`(3), `Fldtype`(3)

Fmove (3FML)

Name `Fmove`, `Fmove32`-move fielded buffer to destination

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fmove(char *dest, FBFR *src)

#include "fml32.h"

int
Fmove32(char *dest, FBFR32 *src)
```

Description `Fmove`() should be used when copying from a fielded buffer to any type of buffer. *dest* and *src* are pointers to the destination buffer and the source fielded buffers respectively.

The difference between `Fmove`() and `Fcpy`(3) is that `Fcpy`(3) expects the destination to be a fielded buffer and thus can make sure it is of sufficient size to accommodate the data from the source buffer. `Fmove`() makes no such check, blindly moving `Fsizeof`(3) bytes of data from the source fielded buffer to the target buffer. The destination buffer must be aligned on a short boundary.

`Fmove32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fmove`() fails and sets `Ferror` to:

`[FALIGNERR]`
: "fielded buffer not aligned"
The source or destination buffer does not begin on the proper boundary.

`[FNOTFLD]`
: "buffer not fielded"
The source buffer is not a fielded buffer or has not been initialized by `Finit`().

See Also `Fintro`(3), `Fcpy`(3), `Fsizeof`(3)

Fname (3FML)

Name Fname, Fname32-map field identifier to field name

Synopsis

```
#include <stdio.h>
#include "fml.h"

char *
Fname(FLDID fieldid)

#include "fml32.h"

char *
Fname32(FLDID32 fieldid)
```

Description Fname() provides a runtime translation of a field identifier, *fieldid*, to its field name and returns a pointer to a character string containing the name corresponding to its argument. The first invocation causes space to be dynamically allocated for the field tables and the tables to be loaded. The table space used by the mapping tables created by Fname() may be recovered by a call to the function Fidnm_unload(3).

Fname32 is used with 32-bit FML.

Return Values This function returns NULL on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fname() fails and sets Ferror to:

[FBADFLD]
"unknown field number or type"
A field number is specified for which a field name cannot be found or is invalid (0).

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed.

See Also Fintro(3), Ffprint(3), Fidnm_unload(3), Fldid(3), Fprint(3)

Fneeded (3FML)

Name `Fneeded`, `Fneeded32`-compute size needed for buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

long
Fneeded(FLDOCC F, FLDLEN V)

#include "fml32.h"

long
Fneeded32(FLDOCC32 F, FLDLEN32 V)
```

Description `Fneeded`() if used to determine the space that must be allocated for *F* fields and *V* bytes of value space.

`Fneeded32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fneeded`() fails and sets `Ferror` to:

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, number of fields is less than 0, *V* is 0 or total size is greater than 65534).

See Also `Fintro`(3), `Falloc`(3), `Finit`(3), `Fielded`(3), `Fsizeof`(3), `Funused`(3), `Fused`(3)

Fnext (3FML)

Name Fnext, Fnext32-get next field occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fnext(FBFR *fbfr, FLDID *fieldid, FLDOCC *oc, char *value,  FLDLEN
*len)

#include "fml32.h"

int
Fnext32(FBFR32 *fbfr, FLDID32 *fieldid, FLDOCC32 *oc, char *value,
FLDLEN32 *len)
```

Description Fnext() finds the next field in the buffer after the specified field occurrence. *fbfr* is a pointer to a fielded buffer. *fieldid* is a pointer to a field identifier. *oc* is a pointer to the occurrence number of the field. *value* is a pointer to the value of the next field. *len* is the length of the next value.

The field identifier, FIRSTFLDID, should be specified to get the first field in the buffer (for example, on the first call to Fnext()). If *value* is not NULL, the next field value is copied into *value*; **len* is used to determine if the buffer has enough space allocated to contain the value. The value's length is returned in **len*. If *len* is NULL when the function is called, it is assumed that there is enough space and the new value length is not returned. If *value* is NULL, the value is not retrieved and only *fieldid* and *oc* are updated. The **fieldid* and **oc* parameters are respectively set to the next found field and occurrence. If no more fields are found, 0 is returned (end of buffer) and **fieldid*, **oc*, and **value* are left unchanged. Fields are returned in field identifier order.

Although the type of value is char *, the value returned will be of the same type as the next field being retrieved.

Fnext32 is used with 32-bit FML.

Return Values Fnext() returns 1 when the next occurrence is successfully found. It returns 0 when the end of the buffer is reached.

This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, `Fnext`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOSPACE]`
"no space"
The size of value, as specified in *len*, is not large enough to hold the field value.

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, specifying NULL for *fieldid* or *oc*).

See Also `Fintro`(3), `Fget`(3), `Fnum`(3)

Fnmid_unload (3FML)

Name `Fnmid_unload`, `Fnmid_unload32`-recover space from nm->id mapping tables

Synopsis

```
#include <stdio.h>
#include "fml.h"
void  Fnmid_unload(void)
#include "fml32.h"
void  Fnmid_unload32(void)
```

Description To recover data space used by the field tables loaded by `Fldid`(3), the user may unload the files by a call to the `Fnmid_unload`() function.

`Fnmid_unload32` is used with 32-bit FML.

Return Values This function is declared as a `void` and so does not return anything.

See Also `Fintro`(3), `Fidnm_unload`(3), `Fldid`(3)

Fnum (3FML)

Name Fnum, Fnum32-return count of all occurrences in buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

FLDOCC
Fnum(FBFR *fbfr)

#include "fml32.h"

FLDOCC32
Fnum32(FBFR *fbfr)
```

Description Fnum() returns the number of fields contained in the specified buffer. *fbfr* is a pointer to a fielded buffer.

Fnum32 is used with 32-bit FML.

Return Values This function returns -1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fnum() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

See Also Fintro(3), Foccur(3), Fpres(3)

Foccur (3FML)

Name Foccur, Foccur32-return count of field occurrences in buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

FLDOCC
Foccur(FBFR *fbfr, FLDID fieldid)

#include "fml32.h"

FLDOCC32 Foccur32(FBFR32 *fbfr, FLDID32 fieldid)
```

Description Foccur() is used to determine the number of occurrences of the field specified by *fieldid* in the buffer pointed to by *fbfr*.

Foccur32 is used with 32-bit FML.

Return Values On success, Foccur() returns the number of occurrences; if none are found, it returns 0.

This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Foccur() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FBADFLD]
"unknown field number or type"
A field identifier is specified which is not valid.

See Also Fintro(3), Fnum(3), Fpres(3)

Fojoin (3FML)

Name `Fojoin, Fojoin32`-outer join source into destination buffer

```
#include <stdio.h>
#include "fml.h"

int
Fojoin(FBFR *dest, FBFR *src)

#include "fml32.h"

int
Fojoin32(FBFR32 *dest, FBFR32 *src)
```

Description `Fojoin`() is similar to `Fjoin`(3), but it keeps fields from the destination buffer, *dest*, that have no corresponding fieldid/occurrence in the source buffer, *src*. Fields that exist in the source buffer that have no corresponding fieldid/occurrence in the destination buffer are not added to the destination buffer.

As with `Fjoin`(3), this function can fail for lack of space; it can be re-issued again after allocating more space to complete the operation.

`Fojoin32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fojoin`() fails and sets `Ferror` to:

[`FALIGNERR`]
"fielded buffer not aligned"
Either the source buffer or the destination buffer does not begin on the proper boundary.

[`FNOTFLD`]
"buffer not fielded"
Either the source buffer or the destination buffer is not a fielded buffer or has not been initialized by Finit().

[`FNOSPACE`]
"no space in fielded buffer"
A field value is to be added or changed in a field buffer but there is not enough space remaining in the buffer.

Example In the following example,

```
if(Fojoin(dest,src)  0)
  F_error("pgm_name");
```

if `dest` has fields A, B, and two occurrences of C, and `src` has fields A, C, and D, the resultant `dest` will contain the source field value A, the destination field value B,the source field value C, and the second destination field value C.

See Also `Fintro`(3), `Fconcat`(3), `Fjoin`(3), `Fproj`(3)

Fpres (3FML)

Name `Fpres, Fpres32`-true if field occurrence is present in buffer

```
#include <stdio.h>

#include "fml.h"

int
Fpres(FBFR *fbfr, FLDID fieldid, FLDOCC oc)

#include "fml32.h"

int
Fpres32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc)
```

Description `Fpres`() is used to detect if a given occurrence, *oc*, of a specified field, *fieldid*, exists in the buffer pointed to by *fbfr*.

`Fpres32` is used with 32-bit FML.

Return Values `Fpres`() returns `true` (1) if the specified occurrence exists and `false` (0) otherwise.

See Also `Fintro`(3), `Ffind`(3), `Fnum`(3), `Foccur`(3)

Fprint (3FML)

Name Fprint, Fprint32-print buffer to standard output

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fprint(FBFR *fbfr)

#include "fml32.h"

int
Fprint32(FBFR32 *fbfr)
```

Description Fprint() prints the specified buffer to the standard output. *fbfr* is a pointer to a fielded buffer. For each field in the buffer, the output prints the field name and field value separated by a tab. Fname(3) is used to determine the field name; if the field name cannot be determined, then the field identifier is printed. Non-printable characters in string and character array field values are represented by a backslash followed by their two-character hexadecimal value. A newline is printed following the output of the printed buffer.

Fprint32 is used with 32-bit FML.

Return Values This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fprint() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FMALLOC]
"malloc failed"
Allocation of space dynamically using malloc(3) failed.

See Also Fintro(3), Fextread(3), Fname(3), Ffprint(3)

Fproj (3FML)

Name `Fproj`, `Fproj32`-projection on buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fproj(FBFR *fbfr, FLDID *fieldid)

#include "fml32.h"

int
Fproj32(FBFR32 *fbfr, FLDID32 *fieldid)
```

Description `Fproj`() is used to update a buffer so as to keep only the desired fields. *fbfr* is a pointer to a fielded buffer. The desired fields are specified in an array of field identifiers pointed to by *fieldid*. The last entry in the array must be `BADFLDID`. The update is done in-place; fields that are not in the result of the projection are deleted from the fielded buffer. The array of field identifiers may be re-arranged (if they are not already in numeric order, they are sorted).

`Fproj32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fproj`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

Example

```
#include "fld.tbl.h"
FBFR *fbfr;
FLDID fieldid[20];
...
fieldid[0] = A;          /* field id for field A */
fieldid[1] = D;          /* field id for field D */
fieldid[2] = BADFLDID;   /* sentinel value */
...
if(Fproj(fbfr, fieldid)  0)
    F_error("pgm_name");
```

If the buffer has fields A, B, C, and D, the example results in a buffer that contains only occurrences of fields A and D. The entries in the array of field identifiers do not need to be in any specific order, but the last value in the array of field identifiers must be field identifier 0 (`BADFLDID`).

See Also `Fintro`(3), `Fjoin`(3), `Fojoin`(3), `Fprojcpy`(3)

Fprojcpy (3FML)

Name Fprojcpy, Fprojcpy32-projection and copy on buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fprojcpy(FBFR *dest, FBFR *src, FLDID *fieldid)

#include "fml32.h"

int
Fprojcpy32(FBFR32 *dest, FBFR32 *src, FLDID32 *fieldid)
```

Description Fprojcpy() is similar to Fproj(3) but the projection is done into a destination buffer instead of in-place. *dest* and *src* are pointers to the destination and source fielded buffers respectively. *fieldid* is a pointer to an array of field identifiers. Any fields in the destination buffer are first deleted and the results of the projection on the source buffer are put into the destination buffer. The source buffer is not changed. The array of field identifiers may be re-arranged (if they are not already in numeric order, they are sorted).

This function can fail for lack of space; it can be re-issued after allocating enough additional space to complete the operation.

Fprojcpy32 is used with 32-bit FML.

Return Values This function returns \-1 on error and sets **Ferror** to indicate the error condition.

Errors Under the following conditions, Fprojcpy() fails and sets Ferror to:

[FALIGNERR]
: "fielded buffer not aligned"
Either the source buffer or the destination buffer does not begin on the proper boundary.

[FNOTFLD]
: "buffer not fielded"
Either the source buffer or the destination buffer is not a fielded buffer or has not been initialized by Finit().

[FNOSPACE]
: "no space in fielded buffer"
A field value is to be copied to the destination fielded buffer but there is not enough space remaining in the buffer.

See Also Fintro(3), Fjoin(3), Fojoin(3), Fproj(3)

Fread (3FML)

Name Fread, Fread32-read fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fread(FBFR *fbfr, FILE *iop)

#include "fml32.h"

int
Fread32(FBFR32 *fbfr, FILE32 *iop)
```

Description Fielded buffers may be read from file streams using Fread(). *fbfr* is a pointer to a fielded buffer. *iop* is a pointer of type FILE to the input stream. (See stdio(3S) in a UNIX System reference manual for a discussion of streams). Fread() reads the fielded buffer from the stream into *fbfr*, clearing any data previously stored in the buffer, and recreates the buffer's index.

Fread32 is used with 32-bit FML.

Return Values This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fread() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit(). This error is also returned if the data that is read is not a fielded buffer.

[FNOSPACE]
"no space in fielded buffer"
There is not enough space in the buffer to hold the fielded buffer being read from the stream.

[FEUNIX]
"UNIX system call error"
The read() system call failed. The external integer errno should have been set to indicate the error by the system call.

See Also `Fintro`(3), `stdio`(3S) in UNIX System reference manuals, `Findex`(3), `Fwrite`(3)

Frealloc (3FML)

Name Frealloc, Frealloc32-re-allocate fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

FBFR *
Frealloc(FBFR *fbfr, FLDOCC nf, FLDLEN nv)

#include "fml32.h"

FBFR32 *
Frealloc32(FBFR32 *fbfr, FLDOCC32 nf, FLDLEN32 nv)
```

Description Frealloc() can be used to re-allocate space to enlarge a fielded buffer. *fbfr* is a pointer to a fielded buffer. The second and third parameters are the new number of fields, *nf*, and the new number of bytes value space, *nv*. These are not increments.

Frealloc32 is used with 32-bit FML.

Return Values On success, Frealloc returns a pointer to the re-allocated FBFR.

This function returns NULL on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Frealloc() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FEINVAL]
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, number of fields is less than 0, *V* is 0 or total size is greater than 65534).

[FMALLOC]
"malloc failed"
The new size is smaller than what is currently in the buffer, or allocation of space dynamically using realloc(3) failed.

See Also Fintro(3), Falloc(3), Ffree(3)

Frstrindex (3FML)

Name Frstrindex, Frstrindex32-restore index in a buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Frstrindex(FBFR *fbfr, FLDOCC numidx)

#include "fml32.h"

int
Frstrindex32(FBFR32 *fbfr, FLDOCC32 numidx)
```

Description A fielded buffer that has been unindexed may be reindexed by either calling Findex(3) or Frstrindx(). *fbfr* is a pointer to a fielded buffer. The former performs a total index calculation on the buffer, and is fairly expensive (requiring a full scan of the buffer). It should be used when an unindexed buffer has been altered, or the previous state of the buffer is unknown (for example, when it has been sent from one process to another without an index). Frstrindex() is much faster, but may only be used if the buffer has not been altered since its previous unindexing operation. The second argument to Frstrindx(), *numidx*, is the return from the Funindex(3) function.

Frstrindex32 is used with 32-bit FML.

Return Values This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Frstrindex() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

Example In order to transmit a buffer without its index, something like the following should be performed:

```
save = Funindex(fbfr);
num_to_send = Fused(fbfr);
transmit(fbfr,num_to_send);        /* A hypothetical function */
Frstrindx(fbfr,save);
```

These four statements do the following:

```
1. - /* unindex, saving for Frstrindx */
2. - /* determine number of bytes to send */
3. - /* send fbfr, without index */
4. - /* restore index */
```

In this case, `transmit`() is passed a memory pointer and a length. The data to be transmitted begins at the memory pointer and has `num_to_send` number of significant bytes. Once the buffer has been sent, its index may be restored (assuming `transmit`() does not alter it in any way) using `Frstrindex`(). On the receiving end of the transmission, the process accepting the fielded buffer would index it with `Findex`(3), as in:

```
receive(fbfr); /* get fbfr from wherever .. into fbfr */
Findex(fbfr);  /* index it */
```

The receiving process cannot call `Frstrindx`() because:

1. it did not call Funindex(3) and so has no idea of what the value of the *numidx* argument to `Frstrindex`() should be
2. the index itself is not available because it was not sent.

The solution is to call `Findex`(3) explicitly. Of course, the user is always free to transmit the indexed versions of a fielded buffer (that is, send `Fsizeof(*fbfr)` bytes) and avoid the cost of `Findex`(3) on the receiving side.

See Also `Fintro`(3), `Findex`(3), `Fsizeof`(3), `Funindex`(3)

Fsizeof (3FML)

Name `Fsizeof`, `Fsizeof32`-return size of fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

long
Fsizeof(FBFR *fbfr)

#include "fml32.h"

long
Fsizeof32(FBFR32 *fbfr)
```

Description `Fsizeof`() returns the size of a fielded buffer in bytes. *fbfr* is a pointer to a fielded buffer.

`Fsizeof32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fsizeof`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

See Also `Fintro`(3), `Fidxused`(3), `Fused`(3), `Funused`(3)

Fstrerror (3FML)

Name Fstrerror, Fstrerror32—get error message string for FML error

Synopsis

```
#include <fml.h>

char *
Fstrerror(int err)

#include <fml32.h>

char *
Fstrerror32(int err)
```

Description Fstrerror is used to retrieve the text of an error message from LIBFML_CAT. *err* is the error code set in F_error when a FML function call returns a -1 or other failure value.

The user can use the pointer returned by Fstrerror as an argument to userlog or F_error.

Fstrerror32 is used with 32-bit FML.

Return Values If *err* is an invalid error code, Fstrerror returns a NULL. On success, the function returns a pointer to a string that contains the error message text.

Errors Fstrerror returns a NULL on error, but does not set F_error.

See Also Fintro(3fml), tpstrerror(3c), F_error(3fml), userlog(3c)

Ftypcvt (3FML)

Name `Ftypcvt`, `Ftypcvt32`-convert from one field type to another

Synopsis

```
#include <stdio.h>
#include "fml.h"

char *
Ftypcvt(FLDLEN *tolen, int totype, char *fromval, int fromtype,
  FLDLEN fromlen)

#include "fml32.h"

char *
Ftypcvt32(FLDLEN32 *tolen, int totype, char *fromval, int fromtype,
  FLDLEN32 fromlen)
```

Description `Ftypcvt`() converts the value **fromval*, which has type *fromtype*, and length *fromlen* (if *fromtype* is FLD_CARRAY; otherwise, *fromlen* is inferred from *fromtype*), to a value of type *totype*. `Ftypcvt`() returns a pointer to the converted value, and sets **tolen* to the converted length, upon success. Upon failure, `Ftypcvt`() returns NULL.

`Ftypcvt32` is used with 32-bit FML.

Return Values This function returns NULL on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, `Ftypcvt`() fails and sets `Ferror` to:

`[FMALLOC]`
"malloc failed"
Allocation of space dynamically using malloc(3) failed when converting from a carray to string.

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, a NULL *tolen* or *fromval* parameter was specified).

`[FTYPERR]`
"invalid field type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `CFadd`(3), `CFchg`(3), `CFget`(3), `CFgetalloc`(3), `CFfind`(3)

Ftype (3FML)

Name `Ftype`, `Ftype32`-return pointer to type of field

Synopsis

```
#include <stdio.h>
#include "fml.h"

char *
Ftype(FLDID fieldid)

#include "fml32.h"

char *
Ftype32(FLDID32 fieldid)
```

Description `Ftype`() returns a pointer to a string containing the name of the type of a field, given a field identifier, *fieldid*. For example, if the `FLDID` of a field of type `short` is supplied to `Ftype`(), a pointer is returned to the string "short." This data area is "read-only."

`Ftype32` is used with 32-bit FML.

Return Values On success, `Ftype`() returns a pointer to a character string that identifies the field type.

This function returns NULL on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Ftype`() fails and sets `Ferror` to:

`[FTYPERR]`
"invalid field type"
A field identifier is specified which is not valid.

See Also `Fintro`(3), `Fldid`(3), `Fldno`(3)

Funindex (3FML)

Name Funindex, Funindex32-discard fielded buffer's index

Synopsis

```
#include <stdio.h>
#include "fml.h"

FLDOCC
Funindex(FBFR *fbfr)

#include "fml32.h"

FLDOCC32
Funindex32(FBFR32 *fbfr)
```

Description Funindex() discards a fielded buffer's index. *fbfr* is a pointer to a fielded buffer. When the function returns successfully, the buffer is unindexed. As a result, none of the buffer's space is allocated to an index and more space is available to user fields (at the cost of potentially slower access time). Unindexing a buffer is useful when it is to be stored on disk or to be transmitted somewhere. In the first case disk space is conserved, in the second, transmission costs may be reduced.

The number of significant bytes from the buffer start, after a buffer has been unindexed is determined by the function call: `Fused(fbfr)`

Funindex32 is used with 32-bit FML.

Return Values Funindex() returns the number of index elements the buffer has before the index is stripped.

This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Funindex() fails and sets Ferror to:

[FALIGNERR]
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

See Also Fintro(3), Findex(3), Frstrindex(3), Fsizeof(3), Funused(3)

Funused (3FML)

Name `Funused`, `Funused32`-return number of unused bytes in fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

long
Funused(FBFR *fbfr)

#include "fml32.h"

long
Funused32(FBFR32 *fbfr)
```

Description `Funused`() returns the amount of space currently unused in the buffer. Space is unused if it contains neither user data nor overhead data such as the header and index.

fbfr is a pointer to a fielded buffer.

`Funused32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Funused`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

See Also `Fintro`(3), `Fidxused`(3), `Fused`(3)

Fupdate (3FML)

Name `Fupdate`, `Fupdate32`-update destination buffer with source

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fupdate(FBFR *dest, FBFR *src)

#include "fml32.h"

int
Fupdate32(FBFR32 *dest, FBFR32 *src)
```

Description `Fupdate`() updates the destination buffer with the field values in the source buffer. *dest* and *src* are pointers to fielded buffers. For fields that match on fieldid/occurrence, the field value is updated in the destination buffer with the value in the source buffer. Fields in the destination buffer that have no corresponding field in the source buffer are left untouched. Fields in the source buffer that have no corresponding field in the destination buffer are added to the destination buffer.

`Fupdate32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fupdate`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
Either the source buffer or the destination buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The source or destination buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FNOSPACE]`
"no space in fielded buffer"
A field value is to be added or changed in the destination buffer but there is not enough space remaining in the buffer.

See Also `Fintro`(3), `Fjoin`(3), `Fojoin`(3), `Fproj`(3), `Fprojcpy`(3)

Fused (3FML)

Name `Fused`, `Fused32`-return number of used bytes in fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

long
Fused(FBFR *fbfr)

#include "fml32.h"

long
Fused32(FBFR32 *fbfr)
```

Description `Fused`() returns the amount of used space in a fielded buffer in bytes, including both user data and the header (but not the index, which can be dropped at any time). *fbfr* is a pointer to a fielded buffer.

`Fused32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fused`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

See Also `Fintro`(3), `Fidxused`(3), `Funused`(3)

Fvall (3FML)

Name `Fvall`, `Fvall32`-return long value of field occurrence

```
#include <stdio.h>
#include "fml.h"

long
 Fvall(FBFR *fbfr, FLDID fieldid, FLDOCC oc)

#include "fml32.h"

long
Fvall32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc)
```

Description `Fvall`() works like `Ffind`(3) for long and short values, but returns the actual value of the field as a long, instead of a pointer to the value. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field.

If the specified field occurrence is not found, then 0 is returned. This function is useful for passing the value of a field to another function without checking the return value. This function is valid only for fields of type `FLD_LONG` or `FLD_SHORT`.

`Fvall32` is used with 32-bit FML.

Return Values For fields of types other than `FLD_LONG` or `FLD_SHORT`, `Fvall`() returns 0 and sets `Ferror` to `FTYPERR`.

This function returns 0 on other errors and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fvall`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

`[FBADFLD]`
"unknown field number or type"
A field identifier is specified which is not valid.

`[FTYPERR]`
"invalid field type"
Bad fieldid or the field type is not `FLD_SHORT` or `FLD_LONG`.

See Also `Fintro`(3), `Ffind`(3), `Fvals`(3)

Fvals (3FML)

Name Fvals, Fvals32-return string value of field occurrence

Synopsis

```
#include <stdio.h>
#include "fml.h"

char *
Fvals(FBFR *fbfr, FLDID fieldid, FLDOCC oc)

#include "fml32.h"

char *
Fvals32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc)
```

Description Fvals() works like Ffind(3) for string values but guarantees that a value is returned. *fbfr* is a pointer to a fielded buffer. *fieldid* is a field identifier. *oc* is the occurrence number of the field.

If the specified field occurrence is not found, then the null string is returned. This function is useful for passing the value of a field to another function without checking the return value. This function is valid only for fields of type FLD_STRING; the null string is automatically returned for other field types (that is, no conversion is done).

Fvals32 is used with 32-bit FML.

Return Values This function returns the null string on error and sets Ferror to indicate the error condition.

Errors Under the following conditions, Fvals() fails and sets Ferror to:

[FALIGNERR]
: "fielded buffer not aligned"
The buffer does not begin on the proper boundary.

[FNOTFLD]
: "buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by Finit().

[FBADFLD]
: "unknown field number or type"
A field identifier is specified which is not valid.

[FTYPERR]
: "invalid field type"
Bad fieldid or the field type is not FLD_STRING.

See Also Fintro(3), CFfind(3), Ffind(3), Fvall(3)

Fwrite (3FML)

Name `Fwrite`, `Fwrite32`-write fielded buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"

int
Fwrite(FBFR *fbfr, FILE *iop)

#include "fml32.h"

int
Fwrite32(FBFR32 *fbfr, FILE *iop)
```

Description Fielded buffers may be written to streams by `Fwrite`(). (See `stdio`(3S) in a UNIX System reference manual for a discussion of streams). `Fwrite`() discards a buffer's index.

fbfr is a pointer to a fielded buffer. *iop* is a pointer of type `FILE` to the output stream.

`Fwrite32` is used with 32-bit FML.

Return Values This function returns \-1 on error and sets `Ferror` to indicate the error condition.

Errors Under the following conditions, `Fwrite`() fails and sets `Ferror` to:

`[FALIGNERR]`
"fielded buffer not aligned"
The buffer does not begin on the proper boundary.

`[FNOTFLD]`
"buffer not fielded"
The buffer is not a fielded buffer or has not been initialized by `Finit`().

`[FEUNIX]`
"UNIX system call error"
The `write` system call failed. The external integer `errno` should have been set to indicate the error by the system call, and the external integer Uunixerr (values defined in `Uunix.h`) is set to the system call that returned the error.

See Also `Fintro`(3), `stdio`(3S) in UNIX System reference manuals, `Findex`(3), `Fread`(3)

2 field_tables Description

field_tables(5)

Name `field_tables`-FML mapping files for field names

description The Field Manipulation Language functions implement and manage fielded buffers. Each field in a fielded buffer is tagged with an identifying integer. Fields that can variable in length (for example, a string) have an additional length modifier. The buffer then consists of a series of numeric-identifier/data pairs and numeric-identifier/length/data triples.

The numeric-identifier of a field is called its "field identifier" (fldid), and is typedef'd by `FLDID`. A field is named by relating an alphanumeric string (the name) to a `FLDID` in a field table.

The original FML interface supports 16-bit field identifiers, field lengths, and buffer sizes. A newer 32-bit interface, FML32, supports larger identifiers, field lengths, and buffer sizes. All types, function names, etc. are suffixed with "32" (for example, the field identifier type definition is `FLDID32`).

field identifiers FML functions allow field values to be typed. Currently supported types include char, string, short, long, float, double, and character array. Constants for field types are defined in `fml.h` (fml32.h for FML32). So that fielded buffers can be truly self-describing, the type of a field is carried along with the field by encoding the field type in the FLDID. Thus, a FLDID is composed of two elements: a field type, and a field number. Field numbers must be above 100; the numbers 1-100 are reserved for system use.

field mapping For efficiency, it is desirable that the field name to field identifier mapping be available at compile time. For utility, it is also desirable that these mappings be available at run time. To accommodate both these goals, FML represents field tables in text files, and provides commands to generate corresponding C header files. Thus, compile time mapping is done by the C preprocessor, `cpp`, by the usual `#define` macro. Runtime mapping is done by the function `Fldid(\|)` (Fldid32(\|) for FML32), which maps its argument, a field name, to a field identifier by consulting the source field table files.

field table files Files containing field tables have the following format:

- blank lines and lines beginning with # are ignored.
- lines beginning with $ are ignored by the mapping functions but are passed through (without the $) to header files generated by mkfldhdr(1) (the command name is `mkfldhdr32` for FML32). For example, this would allow the application to pass C comments, `what` strings, etc. to the generated header file.

- lines beginning with the string `*base` contain a base for offsetting subsequent field numbers. This optional feature provides an easy way to group and renumber sets of related fields.
- lines that don't begin with either * nor # should have the form:

  ```
  name    rel-numb   type
  ```

  where:

  - name is the identifier for the field. It should not exceed `cpp` restrictions.
  - `rel-numb` is the relative numeric value of the field. It is added to the current base to obtain the field number of the field.
  - `type` is the type of the field, and is specified as one of: char, string, short, long, float, double, carray.

Entries are white-space separated (any combination of tabs and spaces).

conversion of field tables to header files

The command `mkfldhdr` (or `mkfldhdr32`) converts a field table, as described above, into a file suitable for processing by the C compiler. Each line of the generated header file is of the form:

```
#define  name  fldid
```

where `name` is the name of the field, and `fldid` is its field identifier. The field identifier includes the field type and field number, as previously discussed. The field number is an absolute number, that is, base + rel-number. The resulting file is suitable for inclusion in a C program.

environment variables

Functions such as `Fldid`(), which access field tables, and commands such as `mkfldhdr`(1) and `vuform`(1), which use them, both need the shell variables `FLDTBLDIR` and `FIELDTBLS` (`FLDTBLDIR32` and `FIELDTBLS32` for FML32) to specify the source directories and files, respectively, from which the in-memory version of field tables should be created. `FIELDTBLS` specifies a comma-separated list of field table file names. If `FIELDTBLS` has no value, `fld.tbl` is used as the name of the field table file. The `FLDTBLDIR` environment variable is a colon-separated list of \%directories in which to look for each field table whose name is not an absolute path name. (The search for field tables is very similar to the search for executable commands using the `PATH` variable) If `FLDTBLDIR` is not defined, it is taken to be the current directory. Thus, if `FIELDTBLS` and `FLDTBLDIR` are not set, the default is to take `fld.tbl` from the current directory.

The use of multiple field tables is a convenient way to separate groups of fields, such as groups of fields that exist in a database from those which are used only by the application. However, in general field names should be unique across all field tables, since such tables are capable of being converted to C header files (by the `mkfldhdr` command), and identical field names would produce a compiler name conflict warning. In addition, the function `Fldid`, which maps a name to a `FLDID`, does so by searching the multiple tables, and stops upon finding the first successful match.

example The following is a sample field table in which the base shifts from 500 to 700:

```
# employee ID fields are based at 500
*base 500

#name  rel-numb  type  comment
#----  --------  ----  -------
EMPNAM  1     string emp's name
EMPID   2     long  emp's id
EMPJOB  3     char  job type: D,M,F or T
SRVCDAY  4     carray service date

# address fields are based at 700

*base 700

EMPADDR  1     string street address
EMPCITY  2     string city
EMPSTATE 3     string state
EMPZIP  4     long  zip code
```

The associated header file would be

```
#define EMPADDR  ((FLDID)41661) /* number: 701 type: string */
#define EMPCITY  ((FLDID)41662) /* number: 702 type: string */
#define EMPID   ((FLDID)8694)  /* number: 502 type: long */
#define EMPJOB  ((FLDID)16887) /* number: 503 type: char */
#define EMPNAM  ((FLDID)41461) /* number: 501 type: string */
#define EMPSTATE ((FLDID)41663) /* number: 703 type: string */
#define EMPZIP  ((FLDID)8896)  /* number: 704 type: long */
#define SRVCDAY  ((FLDID)49656) /* number: 504 type: carray */
```

see also `mkfldhdr`(1), *BEA MessageQ FML Programmer's Guide*

3 mkfldhdr Command

mkfldhdr, mkfldhdr32

name — `mkfldhdr`, `mkfldhdr32` - Create header files from field-tables

synopsis — `mkfldhdr` [`-d` *outdir*] [*field_table*...] `mkfldhdr32` [`-d` *outdir*] [*field_table*...]

description — `mkfldhdr` translates each field table file to a corresponding header file suitable for inclusion in C programs. The resulting header files provide `#define` macros for converting from field names to field IDs. Header file names are formed by concatenating a `.h` to the simple file name for each file to be converted.

The field table names may be specified on the command line; each file is converted to a corresponding header file.

If the field table names are not given on the command line, then the program uses the `FIELDTBLS` environment variable as the list of field tables to be converted, and the `FLDTBLDIR` environment variable as a list of directories to be searched for the files. `FIELDTBLS` specifies a comma-separated list of field table file names. If `FIELDTBLS` has no value, `fld.tbl` is used as the name of the (only) field table file (in this case, the resulting header file will be (`fld.tbl.h`). The `FLDTBLDIR` environment variable is a colon-separated list of directories in which to look for each field table whose name is not an absolute path name; the search for field tables is very similar to the search for executable commands using the UNIX System `PATH` variable. If `FLDTBLDIR` is not defined, only the current directory is searched. Thus, if no field table names are specified on the command line and `FIELDTBLS` and `FLDTBLDIR` are not set, `mkfldhdr` will convert the field table `fld.tbl` in the current directory into the header file `fld.tbl.h`.

The `-d` option is available to specify that the output header files are to be created in a directory other than the present working directory.

`mkfldhdr32` is used for 32-bit FML. It uses the `FIELDTBLS32` and `FLDTBLDIR32` environment variables.

errors — Error messages are printed if the field table load fails or if an output file cannot be created.

examples —

```
FLDTBLDIR=/project/fldtbls
FIELDTBLS=maskftbl,DBftbl,miscftbl,
export FLDTBLDIR FIELDTBLS
```

`mkfldhdr` produces the `#include` files `maskftbl.h`, `DBftbl.h`, and `miscftbl.h` in the current directory by processing the files `maskftbl`, `DBftbl`, and `miscftbl` in directory `/project/fldtbls`.

With environment variables set as in the example above, the command `mkfldhdr -d$FLDTBLDIR` processes the same input field-table files, and produces the same output files, but places them in the directory given by the value of the environment variable `FLDTBLDIR`.

The command `mkfldhdr myfields` processes the input file `myfields` and produces `myfields.h` in the current directory.

see also `Fintro`(3), `field_tables`(5)

4 MessageQ/TUXEDO Bridge Functions

TMQUEUE_BMQ

Name `TMQUEUE_BMQ` - MessageQ / TUXEDO Messaging Bridge Server

Synopsis
```
TMQUEUE_BMQ
SRVGRP="identifier"
SRVID="number" CLOPT=" [ -A ] [ servopts options  ] --
[-b bmq_bus_id] [-g bmq_group_id] [-t timeout]
[-U user] [-G group] [-E errorqueuename]"
```

Description The MessageQ / TUXEDO messaging bridge manager is a System/T-supplied server that enqueues and dequeues messages from BEA MessageQ queues on behalf of programs calling `tpenqueue`(3c) and `tpdequeue`(3c), respectively. The server also performs the required data and semantic transformations between MessageQ and TUXEDO. The application administrator enables message enqueuing and dequeuing for the application by specifying this server as an application server in the *SERVERS section of the BEA TUXEDO `ubbconfig` file.

Messages originating from BEA TUXEDO have the MessageQ class of MSG_CLAS_TUXEDO. Reply messages from BEA TUXEDO have either the MessageQ class of MSG_CLAS_TUXEDO_TPSUCCESS or MSG_CLAS_TUXEDO_TPFAIL.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. The following is a list of additional command line options that are available for customization:

`-b` *bmq_bus_id*
: The MessageQ bus with which the server communicates. The *bmq_bus_id* option is used instead of the DMQ_BUS_ID environment variable.

`-g` *bmq_group_id*
: The MessageQ group with which the server communicates. The *bmq_group_id* option is used instead of the DMQ_GROUP_ID environment variable.

`-t` *timeout*
: The time in seconds at which an operation specified with flags:TPNOTIME will timeout. If no value is specified, the default value is 60 seconds.This option provides consistency with the transaction timeout in TUXEDO.

-U *user*

The user name or user identification number (UID) for all messages handled by TMQUEUE_BMQ. The *user* argument is used for access control list (ACL) checks when security is configured for a TUXEDO application.

-G *group*

The group name or group identification number (GID) for all messages handled by TMQUEUE_BMQ. The *group* argument is used for access control list (ACL) checks when security is configured for a TUXEDO application.

If the TUXEDO default security mechanism is used, and the *user* option is specified as a user name, the *group* option is not required and should not be specified.

-E *errorqueuename*

The name of the MessageQ error queue. Each MessageQ group includes a reserved queue (queue 97) which is used to store error messages. Specifying the *errorqueuename* option allows BEA TUXEDO and BEA M3 applications and processes to address the error queue by name.

The TMQUEUE_BMQ server must be located on the same physical machine as the BEA MessageQ group from which it dequeues messages. The machine must be configured to run servers on behalf of a BEA TUXEDO application. TMQUEUE_BMQ may enqueue messages to any queue on any machine in the MessageQ group as long as a path exists between the group to which TMQUEUE_BMQ is attached and the target group.

A TMQUEUE_BMQ server is booted as part of a TUXEDO application to facilitate application access to its associated MessageQ bus and group. Any configuration condition that prevents the TMQUEUE_BMQ server from initiating its services will cause TMQUEUE_BMQ to fail at boot time with an error posted to the BEA TUXEDO user log (ULOG) file.

EXAMPLES

```
*GROUPS
TMQUEUE_BMQGRPHQMGR GRPNO=1
TMQUEUE_BMQGRPHQPLEBE GRPNO=2
TMQUEUE_BMQGRPREMOTENA GRPNO=3
TMQUEUE_BMQGRPREMOTEEUROPE GRPNO=4


*SERVERS
TMQUEUE_BMQ SRVGRP="TMQUEUE_BMQGRPHQMGR" SRVID=1000 RESTART=Y
   GRACE=0 CLOPT="-s Payroll:TMQUEUE_BMQ -s
```

```
   Promote:TMQUEUE_BMQ -- -b 5 -g 7"
TMQUEUE_BMQ SRVGRP="TMQUEUE_BMQGRPHQPLEBE" SRVID=1000 RESTART=Y
   GRACE=0 CLOPT="-s Payroll:TMQUEUE_BMQ -s
   Promote:TMQUEUE_BMQ -- -b 5 -g 10"
TMQUEUE_BMQ SRVGRP="TMQUEUE_BMQGRPREMOTENA" SRVID=2002 RESTART=Y
   GRACE=0 CLOPT="-s Sales:TMQUEUE_BMQ -- -b 5 -g 42"
TMQUEUE_BMQ SRVGRP="TMQUEUE_BMQGRPREMOTEEUROPE" SRVID=2002
   RESTART=Y GRACE=0 CLOPT="-s Sales:TMQUEUE_BMQ -- -b 12 -g 53"


*SERVICES
Payroll  ROUTING="SALARYROUTE"
Payroll  ROUTING="HAIRCOLORROUTE"


*ROUTING
SALARYROUTE  FIELD=Salary BUFTYPE="FML32"
   RANGES="MIN - 50000:TMQUEUE_BMQGRPPLEBE,50001
   -MAX:TMQUEUE_BMQGRPHQMGR"
HAIRCOLORROUTE  FIELD=Hair BUFTYPE="FML32"
   RANGES="'Gray':TMQUEUE_BMQGRPHQMGR,*:TMQUEUE_BMQGRPPLEBE"
```

SEE ALSO `ubbconfig`(5), `servopts`(5), `buildserver`(1), `tpenqueue`(3c), `tpdequeue`(3c), `TMQFORWARD_BMQ`(5), BEA TUXEDO Administrator's Guide, BEA TUXEDO Programmer's Guide, BEA MessageQ Introduction to Message Queueing, BEA MessageQ Programmer's Guide

TMQFORWARD_BMQ

Name TMQFORWARD_BMQ - MessageQ / TUXEDO Forwarding Agent Server

Synopsis

```
TMQFORWARD_BMQ
SRVGRP="identifier"
SRVID="number" REPLYQ=N CLOPT=" [ -A ] [ servopts options  ] --
-q queuename[,queuename] [-b bmq_bus_id] [-g bmq_group_id]
[-t timeout] [-i idletime] [-d] [-f delay][-U uid] [-G gid]
[-E errorqueuename] [-R retries]"
```

Description The MessageQ / TUXEDO forwarding agent is a BEA TUXEDO managed server that forwards messages to BEA TUXEDO services from BEA MessageQ queues. The messages are placed on a BEA MessageQ queue using either `pams_put_msg` or `tpenqueue`. The server also performs the required data and semantic transformations between MessageQ and TUXEDO. The application administrator enables message processing for the application by specifying this server as an application server in the *SERVERS section of the BEA TUXEDO `ubbconfig` file.

Messages originating from BEA TUXEDO have the MessageQ class of MSG_CLAS_TUXEDO. Reply messages from BEA TUXEDO have either the MessageQ class of MSG_CLAS_TUXEDO_TPSUCCESS or MSG_CLAS_TUXEDO_TPFAIL.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. Note that REPLYQ=N must be specified, as shown in the synopsis. The following is a list of additional command line options that are available for customization:

`-q` *queuename*`[,`*queuename*`]`
: The names of one or more queues for which TMQFORWARD_BMQ forwards messages.

`-b` *bmq_bus_id*
: The MessageQ bus with which the server communicates. The *bmq_bus_id* option is used instead of the DMQ_BUS_ID environment variable.

`-g` *bmq_group_id*
: The MessageQ group with which the server communicates. The *bmq_group_id* option is used instead of the DMQ_GROUP_ID environment variable.

`-t` *timeout*

The time in seconds at which an operation specified with flags:TPNOTIME will timeout. If no value is specified, the default value is 60 seconds.This option provides consistency with the transaction timeout in TUXEDO.

`-i` *idletime*

The time that the server is idle after draining the queue(s) that it is reading. A value of zero indicates that the server will continually read the queue(s), which can be inefficient if the queues do not continually have messages. If not specified, the default is 30 seconds.

`-d`

Causes messages that result in service failure and have a reply message (non-zero in length) to be deleted from the queue.

`-f` *delay*

Causes the server to forward the message to the service instead of using `tpcall`. The message is sent such that a reply is not expected from the service. The TMQFORWARD_BMQ server does not block waiting for the reply from the service and can continue processing the next message from the queue. To throttle the system such that TMQFORWARD_BMQ does not flood the system with requests, the *delay* numeric value can be used to indicate a delay, in seconds, between processing requests. Use zero for no delay.

`-U` *uid*

The user name or user identification number (UID) for all messages handled by TMQFORWARD_BMQ. The *user* argument is used for access control list (ACL) checks when security is configured for a TUXEDO application.

`-G` *gid*

The group name or group identification number (GID) for all messages handled by TMQFORWARD_BMQ. The *group* argument is used for access control list (ACL) checks when security is configured for a TUXEDO application.

If the TUXEDO default security mechanism is used, and the *user* option is specified as a user name, the *group* option is not required and should not be specified.

`-E` *errorqueuename*

The name of the MessageQ error queue. Each MessageQ group includes a reserved queue (queue 97) which is used to store error messages. Specifying the *errorqueuename* option allows BEA TUXEDO and BEA M3 applications and processes to address the error queue by name.

`-R` *retries*

The number of times that the TMQFORWARD_BMQ server attempts to retry message delivery. The number of retries is in addition to the initial attempt to deliver the message. If the `-R` option is not specified or is specified as zero, only the initial delivery is attempted.

The TMQFORWARD_BMQ server must be located on the same physical machine as the BEA MessageQ group from which it dequeues messages. The machine must be configured to run servers on behalf of a BEA TUXEDO application.

A TMQFORWARD_BMQ server is booted as part of a TUXEDO application to facilitate application access to its associated MessageQ bus and group.Any configuration condition that prevents the TMQFORWARD_BMQ server from initiating its services will cause TMQFORWARD_BMQ to fail at boot time with an error posted to the BEA TUXEDO user log (ULOG) file.

TMQFORWARD_BMQ forwards messages to a server providing a service whose name matches the queue name from which the message is read. The message priority is the priority specified when the message was enqueued. If the message is associated with a reply queue, then any reply from the service will be enqueued to the specified reply queue, along with the returned `tpurcode`. If the reply queue does not exist, the reply will be dropped.

EXAMPLES

```
*GROUPS
TMQUEUE_BMQGRPHQMGR GRPNO=1


*SERVERS
TMQFORWARD_BMQ SRVGRP=”TMQUEUE_BMQGRP” SRVID=1001 RESTART=Y GRACE=0
        CLOPT=” -- -qservice1,service2” REPLYQ=N
TMQUEUE_BMQ SRVGRP="TMQUEUE_BMQGRP" SRVID=1000 RESTART=Y GRACE=0
        CLOPT="-s Payroll:TMQUEUE_BMQ -- -b 5 -g 7"
```

SEE ALSO

`ubbconfig`(5), `servopts`(5), `buildserver`(1), `tpenqueue`(3c), `tpdequeue`(3c), `TMQUEUE_BMQ`(5), BEA TUXEDO Administrator’s Guide, BEA TUXEDO Programmer’s Guide, BEA MessageQ Introduction to Message Queueing, BEA MessageQ Programmer’s Guide

tpdequeue (3)

Name `tpdequeue` - routine to dequeue a message from a queue

Synopsis

```
#include <atmi.h>

int tpdequeue(char *qspace, char *qname, TPQCTL *ctl, char **data,
   long *len,long flags)
```

Description tpdequeue() dequeues a message for processing from the queue named by *qname* in the *qspace* queue space.

By default, the message at the top of the queue is dequeued. The default order of messages on the queue is defined when the queue is created. The application can request a particular message for dequeuing by specifying its message identifier using the *ctl* parameter. *ctl* flags can also be used to indicate that the application wants to wait for a message, in the case where a message is not currently available. See the section below describing this parameter.

data is the address of a pointer to the buffer into which a message is read, and *len* points to the length of that message. **data* must point to a buffer originally allocated by tpalloc(3c). To determine whether a message buffer changed in size, compare its (total) size before tpdequeue() was issued with **len*. Note that **data* may change for reasons other than the buffer's size increased. If **len* is 0 upon return, then the message dequeued has no data portion and neither **data* nor the buffer it points to were modified. It is an error for **data* or *len* to be NULL.

The `TPNOTRAN` flag must be set when exchanging messages between BEA MessageQ and BEA TUXEDO, so messages are not dequeued in transaction mode. The message is dequeued in a separate transaction. If a communication error or a timeout occurs (either transaction or blocking timeout), the application will not know whether or not the message was successfully dequeued and the message may be lost.

Following is a list of valid *flags*.

`TPNOTRAN`
: This flag must be set when exchanging messages between BEA MessageQ and BEA TUXEDO. If the caller is in transaction mode and this flag is set, then the message is not dequeued within the same transaction as the caller. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when dequeuing the message. If message dequeuing fails, the caller's transaction is not affected.

`TPNOBLOCK`
: The message is not dequeued if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). If such a condition occurs, the call fails and `tperrno` is set to `TPEBLOCK`. When `TPNOBLOCK` is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). This blocking condition does not include blocking on the queue itself if the TPQWAIT option is specified.

`TPNOTIME`
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

`TPNOCHANGE`
: When this flag is set, the type of the buffer pointed to by **data* is not allowed to change. By default, if a buffer is received that differs in type from the buffer pointed to by **data*, then **data*'s buffer type changes to the received buffer's type so long as the receiver recognizes the incoming buffer type. That is, the type and sub-type of the dequeued message must match the type and sub-type of the buffer pointed to by **data*.

`TPSIGRSTRT`
: If a signal interrupts any underlying system calls, then the interrupted system call is re-issued. When `TPSIGRSTRT` is not specified and a signal interrupts a system call, then tpdequeue() fails and `tperrno` is set to `TPGOTSIG`.

If tpdequeue() returns successfully, the application can retrieve additional information about the message using *ctl* data structure. The information may include the message identifier for the dequeued message, a correlation identifier that should accompany any reply or failure message so that the originator can correlate the message with the original request, the name of a reply queue if a reply is desired, and the name of the failure queue on which the application can queue information regarding failure to dequeue the message. This is described below.

Control Parameter

The TPQCTL structure is used by the application program to pass and retrieve parameters associated with dequeuing the message. The *flags* element of TPQCTL is used to indicate what other elements in the structure are valid.

On input to tpdequeue(), the following elements may be set in the TPQCTL structure:

```
long flags;             /* indicates which of the values
                         * are set */
char msgid[32];         /* id of message to dequeue */
char corrid[32];        /* correlation identifier of
                           * message to dequeue */
```

Following is a list of valid bits for the *flags* parameter controlling input information for tpdequeue().

`TPNOFLAGS`
: No flags are set. No information is taken from the control structure.

`TPQGETBYMSGID`
: If set, it requests that the message identified by `ctl->msgid` be dequeued. The message identifier would be one that was returned by a prior call to tpenqueue(3c). Note that the message identifier is not valid if the message has moved from one queue to another; in this case, use the correlation identifier. This option cannot be used with the `TPQWAIT` option.

`TPQGETBYCORRID`
: If set, it requests that the message with the correlation identifier specified by `ctl->corrid` be dequeued. The correlation identifier would be one that the application specified when enqueuing the message with tpenqueue(). This option cannot be used with the `TPQWAIT` option.

`TPQWAIT`
: If set, it indicates that an error should not be returned if the queue is empty. Instead, the process should block until a message is available.

On output from tpdequeue(), the following elements may be set in the TPQCTL structure:

```
long flags;            /* indicates which of the values
                        * should be set */
long priority;         /* enqueue priority */
char msgid[32];        /* id of message dequeued */
char corrid[32];       /* correlation identifier used to
                        * identify the message */
char replyqueue[16];   /* queue name for reply */
char failurequeue[16]; /* queue name for failure */
long diagnostic;       /* reason for failure */
long appkey;           /* application authentication client
                        * key */
long urcode;           /* user-return code */
CLIENTID cltid;        /* client identifier for originating
                          * client */
```

Following is a list of valid bits for the flags parameter controlling output information from tpdequeue(). If the flag bit is turned on when tpdequeue() is called, then the associated element in the structure is populated if available and the bit remains set. If the value is not available, the flag bit will be turned off after tpdequeue() completes.

`TPQPRIORITY`
: If set and the value is available, the priority at which the message was queued is stored in *ctl->priority*. The priority is in the range 1 to 100, inclusive, and the higher the number, the higher the priority (that is, a message with a higher number is dequeued before a message with a lower number).

`TPQMSGID`
: If set and the call to tpdequeue() was successful, the message identifier will be stored in *ctl->msgid*.

`TPQCORRID`
: If set and the call to tpdequeue() was successful and the message was queued with a correlation identifier, the value will be stored in *ctl->corrid*. Any reply to a queue must have this correlation identifier.

`TPQREPLYQ`
: If set and the message is associated with a reply queue, the value will be stored in *ctl->replyqueue.* Any reply to the message should go to the named reply queue within the same queue space as the request message.

`TPQFAILUREQ`
: If set and the message is associated with a failure queue, the value will be stored in *ctl->failurequeue*. Any failure message should go to the named failure queue within the same queue space as the request message.

If the call to tpdequeue() failed and `tperrno` is set to TPEDIAGNOSTIC, a value indicating the reason for failure is returned in *ctl->diagnostic*. The possible values are defined below in the DIAGNOSTICS section.

Additionally on output, *ctl->appkey* is set to application authentication key, *ctl->cltid* is set to the identifier for the client originating the request, and *ctl->urcode* is set to the user-return code value that was set when the message was enqueued.

If the *ctl* parameter is NULL, the input flags are considered to be TPNOFLAGS and no output information is made available to the application program.

Return Values

This function returns -1 on error and sets `tperrno` to indicate the error condition.

Errors

Under the following conditions, tpdequeue() fails and sets `tperrno` to one of the following (unless otherwise noted, failure does not affect the caller's transaction, if one exists):

`[TPEINVAL]`
: Invalid arguments were given (for example, *qname* is NULL, *data* does not point to space allocated with tpalloc(3c) or *flags* are invalid).

[TPENOENT]
Cannot access the *qspace* because it is not available (the associated TMQUEUE(5) server is not available) or the name begins with "..".

[TPEOTYPE]
Either the type and sub-type of the dequeued message are not known to the caller; or, TPNOCHANGE was set in *flags* and the type and sub-type of **data* do not match the type and sub-type of the dequeued message. Regardless, neither **data*, its contents nor **len* are changed. When this error occurs, the transaction is marked abort-only and the message will remain on the queue.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is to be aborted; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. If a transaction timeout occurred, any attempts to dequeue new messages will fail with TPETIME until the transaction has been aborted.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
tpdequeue() was called in an improper context. There is no effect on the queue or the transaction.

[TPESYSTEM]
A System /T error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]
An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]
Dequeuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via ctl structure.

Diagnostic

The following diagnostic values are returned during the dequeuing of a message.

[QMEINVAL]
An invalid flag value was specified.

[QMEBADRMID]
An invalid resource manager identifier was specified.

[QMENOTOPEN]
The resource manager is not currently open.

[QMETRAN]
The call was made with the TPNOTRAN flag and an error occurred trying to start a transaction in which to dequeue the message.

[QMEBADMSGID]
An invalid message identifier was specified for dequeuing.

[QMEINUSE]
When dequeuing a message by correlation or message identifier, the specified message is in-use by another transaction. Otherwise, all messages currently on the queue are in-use by other transactions.

[QMESYSTEM]
A system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]
An operating system error has occurred.

[QMEABORTED]
The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]
A dequeue was done when the transaction state was not active.

[QMEBADQUEUE]
An invalid or deleted queue name was specified.

[QMENOMSG]
No message was available for dequeuing.

See Also tpalloc(3c), tpenqueue(3c), TMQUEUE_BMQ(5)

tpenqueue (3)

Name `tpenqueue` - routine to enqueue a message

Synopsis

```
#include <atmi.h>
int tpenqueue(char *qspace, char *qname,
TPQCTL *ctl, char *data, long len, long flags)
```

Description `tpenqueue`() stores a message on the queue named by *qname* in the *qspace* queue space. A queue space is a collection of queues, one of which must be *qname*.

When the message is intended for a System/T server, the *qname* matches the name of a service provided by a server. The system provided server, TMQFORWARD_BMQ(5), provides a default mechanism for dequeuing messages from the queue and forwarding them to servers that provide a service matching the queue name. If the originator expected a reply, then the reply to the forwarded service request is stored on the originator's (stable) queue. The originator will dequeue the reply message at a subsequent time. Queues can also be used for a reliable message transfer mechanism between any pair of System/T processes (clients and/or servers). In this case, the queue name does not match a service name but some agreed upon title for transferring the message.

If *data* is non-NULL, it must point to a buffer previously allocated by `tpalloc`(3c) and *len* should specify the amount of data in the buffer that should be queued. Note that if *data* points to a buffer of a type that does not require a length to be specified (for example, an FML fielded buffer), then *len* is ignored. If *data* is NULL, *len* is ignored and a message is queued with no data portion.

The message is queued at the priority defined for *qspace* unless overridden by a previous call to `tpsprio`(3c).

The `TPNOTRAN` flag must be set when exchanging messages between BEA MessageQ and BEA TUXEDO, so messages are not enqueued in transaction mode. The message is not queued in transaction mode if either the caller is not in transaction mode, or the `TPNOTRAN` flag is set. In this case, the queued message is stored on the queue in a separate transaction. Once `tpenqueue`() returns successfully, the submitted message is guaranteed to be available. If a communication error or a timeout occurs (either transaction or blocking timeout), the application will not know whether or not the message was successfully stored on the queue.

The order in which messages are placed on the queue is controlled by the application via *ctl* data structure as described below; the default queue ordering is set when the queue is created.

Following is a list of valid flags.

`TPNOTRAN`
: This flag must be set when exchanging messages between BEA MessageQ and BEA TUXEDO. If the caller is in transaction mode and this flag is set, then the message is not queued within the same transaction as the caller. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when queuing the message. If message queuing fails, the caller's transaction is not affected.

`TPNOBLOCK`
: The message is not enqueued if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). If such a condition occurs, the call fails and tperrno is set to `TPEBLOCK`. When `TPNOBLOCK` is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

`TPNOTIME`
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

`TPSIGRSTRT`
: If a signal interrupts any underlying system calls, then the interrupted system call is re-issued. When `TPSIGRSTRT` is not specified and a signal interrupts a system call, then `tpenqueue`() fails and `tperrno` is set to `TPGOTSIG`.

Additional information about queuing the message can be specified via *ctl* data structure. This information includes values to override the default queue ordering placing the message at the top of the queue or before an enqueued message; an absolute or relative time after which a queued message is made available; a correlation identifier that aids in correlating a reply or failure message with the queued message; the name of a queue to which a reply should be enqueued; and the name of a queue to which any failure message should be enqueued.

Control Parameter

The TPQCTL structure is used by the application program to pass and retrieve parameters associated with enqueuing the message. The *flags* element of TPQCTL is used to indicate what other elements in the structure are valid.

On input to `tpenqueue`(), the following elements may be set in the TPQCTL structure:

```
long flags;             /* indicates which of the values
                         * are set */
long deq_time;          /* absolute/relative for dequeuing */
long priority;          /* enqueue priority */
```

```
long urcode;           /* user-return code */
char msgid[32];        /* id of message before which to queue
                        * request */
char corrid[32];       /* correlation identifier used to
                        * identify the msg */
char replyqueue[16];   /* queue name for reply message */
char failurequeue[16];   /* queue name for failure message */
```

The following is a list of valid bits for the *flags* parameter controlling input information for `tpenqueue`().

`TPNOFLAGS`
: No flags or values are set. No information is taken from the control structure.

`TPQTOP`
: This flag is not supported when exchanging messages between BEA MEssageQ and BEA TUXEDO.

`TPQBEFOREMSGID`
: Setting this flag bit indicates that the queue ordering be overridden and the message placed in the queue before the message identified by *ctl->msgid*. This request may not be granted depending on whether or not the queue was configured to allow overriding the queue ordering. `TPQTOP` and `TPQBEFOREMSGID` are mutually exclusive flags.

`TPQTIME_ABS`
: This flag is not supported when exchanging messages between BEA MEssageQ and BEA TUXEDO.

`TPQTIME_REL`
: This flag is not supported when exchanging messages between BEA MEssageQ and BEA TUXEDO.

`TPQPRIORITY`
: If set, the priority at which the message should be enqueued is stored in *ctl->priority*. The priority must be in the range 1 to 100, inclusive. The higher the number, the higher the priority (that is, a message with a higher number is dequeued before a message with a lower number).

`TPQCORRID`
: If set, the correlation identifier value specified in *ctl->corrid* is available when a message is dequeued with `tpdequeue`(3c). This identifier accompanies any reply or failure message that is queued such that an application can correlate a reply with a particular request. The entire value should be initialized (e.g., padded with null characters) such that the value can be matched at a later time.

`TPQREPLYQ`
: If set, a reply queue named in *ctl->replyqueue* is associated with the queued message. Any reply to the message will be queued to the named queue within the same queue space as the request message. This string must be NULL terminated (maximum 15 characters in length).

`TPQFAILUREQ`
: If set, a failure queue named in *ctl->failurequeue* is associated with the queued message. If a failure occurs when the enqueued message is subsequently dequeued, a failure message will go to the named queue within the same queue space as the original request message. This string must be NULL terminated (maximum 15 characters in length).

Additionally, the *urcode* element of TPQCTL can be set with a user-return code. This value will be returned to the application that dequeues the message.

On output from `tpenqueue()`, the following elements may be set in the TPQCTL structure:

```
long flags;             /* indicates which of the values
                         * are set */
char msgid[32];         /* id of enqueued message */
long diagnostic;          /* indicates reason for failure */
```

Following is a list of valid bits for the flags parameter controlling output information from `tpenqueue()`. If the flag bit is turned on when `tpenqueue()` is called, then the associated element in the structure is populated if available and the bit remains set. If the value is not available, the flag bit will be turned off after `tpenqueue()` completes.

`TPQMSGID`

If set and the call to `tpenqueue()` was successful, the message identifier will be stored in *ctl->msgid*.

If the call to `tpenqueue()` failed and *tperrno* is set to TPEDIAGNOSTIC, a value indicating the reason for failure is returned in *ctl->diagnostic*. The possible values are defined below in the DIAGNOSTICS section.

If this parameter is NULL, the input flags are considered to be TPNOFLAGS and no output information is made available to the application program.

Return Values

This function returns -1 on error and sets *tperrno* to indicate the error condition. Otherwise, the message has been successfully queued when `tpenqueue()` returns.

Errors

Under the following conditions, `tpenqueue`() fails and sets *tperrno* to the following values (unless otherwise noted, failure does not affect the caller's transaction, if one exists):

`[TPEINVAL]`
: Invalid arguments were given (for example, *qspace* is NULL, *data* does not point to space allocated with `tpalloc`(3c), or *flags* are invalid).

`[TPENOENT]`
: Cannot access the *qspace* because it is not available (the associated TMQUEUE(5) server is not available) or the name begins with "..".

`[TPETIME]`
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is to be aborted; otherwise, a blocking timeout occurred and neither `TPNOBLOCK` nor `TPNOTIME` was specified. If a transaction timeout occurred, any attempts to enqueue new messages will fail with `TPETIME` until the transaction has been aborted.

`[TPEBLOCK]`
: A blocking condition exists and `TPNOBLOCK` was specified.

`[TPGOTSIG]`
: A signal was received and `TPSIGRSTRT` was not specified.

`[TPEPROTO]`
: `tpenqueue`() was called in an improper context.

`[TPESYSTEM]`
: A System/T error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`
: An operating system error has occurred.

`[TPEDIAGNOSTIC]`
: Enqueuing a message on the specified queue failed. The reason for failure can be determined by the diagnostic returned via *ctl*.

Diagnostic

The following diagnostic values are returned during the enqueuing of a message.

`[QMEINVAL]`
: An invalid flag value was specified.

`[QMEBADRMID]`
: An invalid resource manager identifier was specified.

`[QMENOTOPEN]`
: The resource manager is not currently open.

`[QMETRAN]`
The call was made with the `TPNOTRAN` flag and an error occurred trying to start a transaction in which to enqueue the message.

`[QMEBADMSGID]`
An invalid message identifier was specified.

`[QMESYSTEM]`
A system error has occurred. The exact nature of the error is written to a log file.

`[QMEOS]`
An operating system error has occurred.

`[QMEABORTED]`
The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

`[QMEPROTO]`
An enqueue was done when the transaction state was not active.

`[QMEBADQUEUE]`
An invalid or deleted queue name was specified.

`[QMENOSPACE]`
There is no space on the queue for the message.

See Also `gp_mktime`(3c), `tpacall`(3c), `tpalloc`(3c), `tpdequeue`(3c), `tpinit`(3c), `tpsprio`(3c)TMQFORWARD_BMQ(5), TMQUEUE_BMQ(5),