BEA TUXEDO

Reference Manual

Section 3FML — FML Commands

BEA TUXEDO 6.5 for WLE 5.0
Document Edition 6.5
December 1999

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA TUXEDO Reference Manual

Document Edition	Date	Software Version
6.5	December 1999	BEA TUXEDO 6.5 for WLE 5.0

About This Document

The TUXEDO 6.5 Reference Manual for WebLogic Enterprise 5.0 includes the following components:

- "Section 1 — Commands" provides information about shell-level commands included with TUXEDO and WebLogic Enterprise (WLE) software.
- "Section 3C — C Functions" describes C language functions that comprise the .Application-Transaction Monitor Interface (ATMI). ATMI provides routines to open and close resources, manage transactions, manage typed buffers, and invoke request/response and conversational service calls.
- "Section 3CBL — COBOL Functions" describes the COBOL bindings for the ATMI interface.
- "Section 3FML — FML Commands" describes C language functions for defining and manipulating Field Manipulation Language (FML) storage structures.
- "Secrtion 5 — File Formats and Data Descriptions" describes various files and tables. This includes the configuration files, `UBBCONFIG` and `TUXCONFIG`, and the TUXEDO Management Information Base (TMIB) classes that provide an interface for managing WLE or TUXEDO systems.

Who Should Use This Document

This document is intended for system administrators and programmers who are interested in creating, configuring, or managing TUXEDO or WebLogic Enterprise applications.

e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the “e-docs” Product Documentation page at http://e-docs.beasys.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Enterprise documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA TUXEDO, distributed object computing, transaction processing, C++ programming, and Java programming, see the WLE Bibliography in the WebLogic Enterprise online documentation.

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at **docsupport@beasys.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.0 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, 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

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.

Convention	Item
italics	Indicates emphasis or book titles.

`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. *Examples*: 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`*`]...`

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

. . .	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.

Contents

Section 3FML - FML Commands

Section 3FML — FML Commands

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.

FML also provides a facility called `VIEWS` that allows you to map fielded buffers to C structures (and the reverse as well). `VIEWS` lets you perform lengthy manipulations of data in structures rather than in fielded buffers; applications will run faster if data is transferred to structures for manipulation. `VIEWS` allows the data independence of fielded buffers to be combined with the efficiency and simplicity of classic record structures.

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`, `FIELDTBLS32`, `VIEWFILES32`, and `VIEWDIR32`).

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`). Any call to `tpalloc` for an FML typed buffer should use the `FMLTYPE` definition instead of "FML". 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.

VIEWS

`VIEWS` is a part of the Field Manipulation Language that allows the exchange of data between fielded buffers and C structures in a C language program, by specifying mappings of fields to members of C structures. If extensive manipulations of fielded buffer information are to be done, transferring the data to C structures will improve performance. Information in a fielded buffer can be extracted from the fields in a buffer and placed in a C structure using `VIEWS` functions, manipulated, and the updated values returned to the buffer, again using `VIEWS` functions.

Typed buffers is a feature of BEA TUXEDO that grew out of the FML idea of a fielded buffer. Two of the standard buffer types delivered with BEA TUXEDO are FML typed buffers and `VIEW` typed buffers. An additional difference of BEA TUXEDO `VIEW` buffers is that they can be totally unrelated to an FML fielded buffer. The buffer types FML32 and `VIEW32` can also be used.

A view description is created and stored in a source viewfile, as described in `viewfile`(5). The view description maps fields in fielded buffers to members in C structures. The source view descriptions are compiled, using `viewc`(1) or `viewc32`(1), creating a view object file and can then be used to map data transferred between fielded buffers and C structures in a C program. The view compiler also creates C header files that can be included in applications programs to define the structures described in view

descriptions. A view disassembler, `viewdis`(1) or `viewdis32`(1), is provided to translate object view descriptions into readable form (that is, back into source view descriptions); the output of the disassembler can be re-input to the view compiler

The object files are used at run-time to manipulate the `VIEW` structures using the `VIEWFILES` and `VIEWDIR` environment variables. `VIEWFILES` should contain a comma separated list of object viewfiles for the application. Files given as full pathnames are used as is; files listed as relative path names are searched for through the list of directories specified by the `VIEWDIR` variable (see below). `VIEWDIR` specifies a colon separated list of directories to be used to find view object files with relative file names. For `VIEW32` structures, `VIEWFILES32` and `VIEWDIR32` are used.

In addition to the data types supported by most FML functions, `VIEWS` supports type `int` in source view descriptions. When the view description is compiled the view compiler automatically converts any `int` types to either short or long types, depending on your machine.

A decimal data type is also supported in `VIEWS`. It is defined as a field of type "`dec_t`", and the size of the packed decimal value is given as the total number of bytes and the bytes to the right of the decimal point. While this field is not supported directly in FML, conversion of this field is automatic to/from any other field type supported in FML. Packed decimals exist in the COBOL environment as two decimal digits packed into one byte with the low-order half byte used to store the sign. In the C environment, the data type is defined by the `dec_t` type definition, which contains the decimal exponent, sign, digits, and the packed decimal value.

An FML buffer can be converted to a view using `Fvftos` or `Fvftos32`. A `VIEW` can be converted to a fielded buffer using `Fvstof` or `Fvstof32`. When transferring data between fielded buffers and structures, the source data is automatically converted to the type of the destination data. Multiple field occurrences are supported; they are treated as an array in the structure. Null values are used to indicate empty members in a structure, and can be specified by the user for each structure member in a viewfile. If the user does not specify a null value for a member, default null values are used. It is also possible to inhibit the transfer of data between a C structure member and a field in a fielded buffer, even though a mapping exists between them.

A `VIEW` can also be converted to and from a target record format. The default target format is IBM System/370 COBOL records. The `Fvstot` function takes care of converting byte ordering, floating point and decimal format, and character sets (ASCII to EBCDIC), and `Fvttos` converts back to the native format. 32-bit versions of these functions also exist. The `Fcodeset` function can be used to specify alternate ASCII/EBCDIC transaction tables.

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), `viewfile`(5), *BEA TUXEDO 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)

F32to16 (3FML)

Name

F32to16, F16to32—convert 16-bit FML to/from 32-bit FML buffer

Synopsis

```
#include <stdio.h>
#include "fml.h"
#include "fml32.h"
int
F32to16(FBFR *dest, FBFR32 *src)
int
F16to32(FBFR32 *dest, FBFR *src)
```

Description

F32to16() converts a 32-bit FML buffer to a 16-bit FML buffer. It does this by converting the buffer on a field-by-field basis and then creating the index for the fielded buffer. A field is converted by generating a FLDID from a FLDID32, and copying the field value (and field length for string and carray fields). *dest* and *src* are pointers to the destination and source fielded buffers respectively. The source buffer is not changed.

These functions can fail for lack of space; they can be re-issued after allocating enough additional space to complete the operation.

F16to32 converts a 16-bit FML buffer to a 32-bit FML buffer. It lives in the fml32 library or shared object and sets *Ferror32* on error.

F32to16 lives in the fml library or shared object and sets *Ferror* on error. Note that both fml.h and fml32.h must be included to use these functions; fml1632.h may not be included in the same file.

Return Values

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

Errors

Under the following conditions, F32to16()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. This error is also returned if a 32-bit FML field is too long to fit into a 16-bit FML field. When this error is returned, the destination buffer will contain no fields.

`[FBADFLD]`

"invalid field number or type"

For `F32to16` only, the source buffer has a field identifier whose field type is not one of the seven field types supported by 16-bit FML, or the field number is greater than 8091.

See Also `Fintro`(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, Fvboolco, Fvboolco32—compile expression, return evaluation tree

Synopsis

```
#include <stdio.h>
#include "fml.h"
char *
Fboolco(char *expression)
char *
Fvboolco(char *expression, char *viewname)
#include "fml32.h"
char *
Fboolco32(char *expression)
char *
Fvboolco32(char *expression, char *viewname)
```

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.

Fvboolco and Fvboolco32 provide the same functionallity for views. The *viewname* parameter indicates the view from which the field offsets are taken.

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 or view files.

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

[FBADVIEW]
"cannot find or get view"
viewname was not found in the files specified by VIEWDIR or VIEWFILES.

[FVFOPEN]
"cannot find or open view file"
While trying to find *viewname*, the program failed to find one of the files specified by VIEWDIR or VIEWFILES.

[EUNIX]
"operating system error"
While trying to find *viewname*, the program failed to open one of the files specified by VIEWDIR or VIEWFILES for reading.

[FVFSYNTAX]
"bad viewfile"
While trying to find *viewname*, one of the files specified by VIEWDIR or VIEWFILES was corrupted or not a view file.

[FMALLOC]
"malloc failed"
While trying to find *viewname*, malloc() failed while allocating space to hold the view information.

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, Fvboolev, Fvboolev32—evaluate buffer against tree

Synopsis

```
#include stdio.h>
#include "fml.h"
int
Fboolev(FBFR *fbfr, char *tree)
int
Fvboolev(char *cstruct, char *tree, char *viewname)
#include "fml32.h"
int
Fboolev32(FBFR32 *fbfr, char *tree)
int
Fvboolev32(char *cstruct, char *tree, char *viewname)
```

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.

Fvboolev and Fvboolev32 provide the same functionallity for views. The *viewname* parameter indicates the view from which the field offsets are taken, and should be the same view specified for Fvboolco or Fvboolco32.

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.

[FBADVIEW]
"cannot find or get view"
viewname was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

[FVFOPEN]
"cannot find or open view file"
While trying to find *viewname*, the program failed to find one of the files specified by `VIEWDIR` or `VIEWFILES`.

[EUNIX]
"operating system error"
While trying to find *viewname*, the program failed to open one of the files specified by `VIEWDIR` or `VIEWFILES` for reading.

[FVFSYNTAX]
"bad viewfile"
While trying to find *viewname*, one of the files specified by `VIEWDIR` or `VIEWFILES` was corrupted or not a view file.

[FMALLOC]
"malloc failed"
While trying to find *viewname*, `malloc`() failed while allocating space to hold the view information.

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, Fvboolpr, Fvboolpr32—print Boolean expression as parsed

Synopsis

```
#include <stdio.h>
#include "fml.h"
void
Fboolpr(char *tree, FILE *iop)
int
Fvboolpr(char *tree, FILE *iop, char *viewname)
#include "fml32.h"
void
Fboolpr32(char *tree, FILE *iop)
int
Fvboolpr32(char *tree, FILE *iop, char *viewname)
```

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.

Fvboolpr and Fvboolpr32 provide the same functionallity for views. The *viewname* parameter indicates the view from which the field offsets are taken, and should be the same view specified for Fvboolco or Fvboolco32.

These functions are not supported on Workstation platforms.

Return Values

Fboolpr() is declared as returning a void, so there are no return values. Fvboolpr returns -1 if the view name is not valid.

Errors

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

[FBADVIEW]
: "cannot find or get view"
viewname was not found in the files specified by VIEWDIR or VIEWFILES.

[FVFOPEN]
: "cannot find or open view file"
While trying to find *viewname*, the program failed to find one of the files specified by VIEWDIR or VIEWFILES.

`[EUNIX]`
"operating system error"
While trying to find *viewname*, the program failed to open one of the files specified by `VIEWDIR` or `VIEWFILES` for reading.

`[FVFSYNTAX]`
"bad viewfile"
While trying to find *viewname*, one of the files specified by `VIEWDIR` or `VIEWFILES` was corrupted or not a view file.

`[FMALLOC]`
"malloc failed"
While trying to find *viewname*, `malloc`() failed while allocating space to hold the view information.

Portability This function is not supported using the BEA TUXEDO System Workstation DLL for OS/2 and Microsoft Windows.

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.

Portability This function is not supported using the TUXEDO System /WS DLL for OS/2 and Microsoft Windows.

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, Fvfloatev, Fvfloatev32-return value of expression as a double

Synopsis

```
#include <stdio.h>
#include "fml.h"
double
Ffloatev(FBFR *fbfr, char *tree)
double
Fvfloatev(char *cstruct, char *tree, char *viewname)
#include "fml32.h"
double
Ffloatev32(FBFR32 *fbfr, char *tree)
double
Fvfloatev32(char *cstruct, char *tree, char *viewname)
```

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.

Fvfloatev and Fvfloatev32 provide the same functionallity for views. The *viewname* parameter indicates the view from which the field offsets are taken, and should be the same view specified for Fvboolco or Fvboolco32.

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.

`[FBADVIEW]`
"cannot find or get view"
viewname was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

`[FVFOPEN]`
"cannot find or open view file"
While trying to find *viewname*, the program failed to find one of the files specified by `VIEWDIR` or `VIEWFILES`.

`[EUNIX]`
"operating system error"
While trying to find *viewname*, the program failed to open one of the files specified by `VIEWDIR` or `VIEWFILES` for reading.

`[FVFSYNTAX]`
"bad viewfile"
While trying to find *viewname*, one of the files specified by `VIEWDIR` or `VIEWFILES` was corrupted or not a view file.

`[FMALLOC]`
"malloc failed"
While trying to find *viewname*, malloc() failed while allocating space to hold the view information.

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.

Portability This function is not supported using the TUXEDO System /WS DLL for OS/2 and Microsoft Windows.

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)

FIdtype (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.

Portability This function is not supported using the TUXEDO System /WS DLL for OS/2 and Microsoft Windows.

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

1. 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)

Fvftos (3FML)

Name

`Fvftos`, `Fvftos32`-copy from fielded buffer to C structure

Synopsis

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

int
Fvftos(FBFR *fbfr, char *cstruct, char *view)

#include "fml32.h"

int
Fvftos32(FBFR32 *fbfr, char *cstruct, char *view)
```

Description

The `Fvftos`() function transfers data from a fielded buffer to a C structure. *fbfr* is a pointer to a fielded buffer. *cstruct* is a pointer to a C structure. *view* is a pointer to the name of a compiled view description.

Fields are copied from the fielded buffer into the structure based on the member descriptions in the *view*. If a field in the fielded buffer has no corresponding member in the C structure, it is ignored. If a member specified in the C structure has no corresponding field in the fielded buffer, a null value is copied into the member. The null value used is definable for each member in the view description.

To store multiple occurrences in the C structure, the structure member should be an array (for example, `int zip`[4] can store 4 occurrences of `zip`). If the buffer has fewer occurrences of the field than there are elements in the array, the extra element slots are assigned null values. On the other hand, if the buffer has more occurrences of the field than there are elements in the array, the surplus occurrences are ignored.

There are view description options that inhibit mappings even though a mapping entry exists for a fldid and a member. These options are initially specified in the view file, but can be changed at runtime using `Fvopt`(3).

`Fvftos32` 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, `Fvftos`() 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 *cstruct* parameter to `Fvftos`)

`[FBADACM]`
"ACM contains negative value"
An Associated Count Member should not be a negative value while transferring data from a structure to a fielded buffer.

`[FBADVIEW]`
"cannot find or get view"
The view description specified was NULL or was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

See Also `Fintro`(3), `Fvopt`(3), `viewfile`(5)

Fvnull (3FML)

Name

Fvnull, Fvnull32-check if a structure element is null

Synopsis

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

int
Fvnull(char *cstruct, char *cname, FLDOCC oc, char *view)

#include "fml32.h"

int
Fvnull32(char *cstruct, char *cname, FLDOCC32 oc, char *view)
```

Description

Fvnull() is used to determine if an occurrence of a structure element is null. *cstruct* is a pointer to a C structure. *cname* is a pointer to the name of an element within *cstruct*. *oc* is the occurrence number of the element. *view* is a pointer to the name of a compiled view description.

Options of Fvopt(3) such as do not affect this function.

Fvnull32 is used for views defined with viewc32 or VIEW32 typed buffers for larger views with more fields.

Return Values

Fvnull() returns 1, if the specified *cname* in a C structure is null and returns 0 if not null. This function returns \-1 on error and sets Ferror to indicate the error condition.

Errors

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

[FBADVIEW]
: "cannot find or get view"
The view description specified was not found in the files specified by VIEWDIR or VIEWFILES.

[FNOCNAME]
: "cname not found"
The C structure field name is not found in the view description.

See Also

Fintro(3), Fvopt(3), viewfile(5)

Fvopt (3FML)

Name Fvopt, Fvopt32-change flag options of a mapping entry

Synopsis

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

int
Fvopt(char *cname, int option, char *view)

#include "fml32.h"

int
Fvopt32(char *cname, int option, char *view)
```

Description Fvopt() allows users to specify buffer-to-structure mapping options at runtime. *cname* is a pointer to the name of an element in a view description, *view*. *option* specifies the desired setting for the mapping option. Valid options and their meanings are:

F_FTOS
: one-way mapping from fielded buffer to structure, flag S in the view description

F_STOF
: one-way mapping from structure to fielded buffer, flag F in the view description

F_OFF
: no mapping between the fielded buffer and the structure, flag N in the view description

F_BOTH
: two-way mapping between the fielded buffer and the structure, flag S,F in the view description

Fvopt32 is used for views defined with viewc32 or VIEW32 typed buffers for larger views with more fields.

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

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

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid (for example, specifying a NULL *cname* or *view* parameter or specifying an invalid *option*).

`[FBADVIEW]`
"cannot find or get view"
The view was not found in the files specified by VIEWDIR and VIEWFILES.

`[FNOCNAME]`
"cname not found"
The C structure field name is not found in the view description.

See Also `Fintro`(3), `viewfile`(5)

Fvrefresh (3FML)

Name `Fvrefresh`, `Fvrefresh32`-copy from C structure to fielded buffer

Synopsis

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

void
Fvrefresh()

#include "fml32.h"

void
Fvrefresh32()
```

Description `Fvrefresh`() clears and reinitializes the internal cache of view structure mappings. This is necessary only when frequently accessed views are updated dynamically.

`Fvrefresh32` is used for views defined with `viewc32` or `VIEW32` typed buffers for larger views with more fields.

Return Values This routine is a void function and does not return a value.

Errors This routine is a void function and no error codes are set.

See Also `Fintro`(3)

Fvselinit (3FML)

Name

Fvselinit, Fvselinit32-initialize structure element to null

Synopsis

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

int
Fvselinit(char *cstruct, char *cname, char *view)

#include "fml32.h"

int
Fvselinit32(char *cstruct, char *cname, char *view)
```

Description

Fvselinit() initializes an individual element of a C structure to its appropriate null value. *cstruct* is a pointer to a C structure. *cname* is a pointer to the name of an element of *cstruct*. *view* is a pointer to the name of a compiled view description.

Fvselinit() sets the associated count member of the element to 0 if the C flag was used when the view was compiled, and sets the associated length member to the length of the associated null value if the L flag was used in the view file.

Fvselinit32 is used for views defined with viewc32 or VIEW32 typed buffers for larger views with more fields.

Return Values

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

Errors

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

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

[FBADVIEW]
: "cannot find or get view"
: The view description specified was NULL or was not found in the files specified by VIEWDIR or VIEWFILES.

[FNOCNAME]
: "cname not found"
: The C structure field name is not found in the view description.

See Also

Fintro(3), Fvsinit(3), viewfile(5)

Fvsinit (3FML)

Name `Fvsinit`, `Fvsinit32`-initialize C structure to null

Synopsis

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

int
Fvsinit(char *cstruct, char *view)

#include "fml32.h"

int
Fvsinit32(char *cstruct, char *view)
```

Description `Fvsinit`() initializes all members in a C structure to the null values specified in the view description, *view*. *cstruct* is a pointer to a C structure. *view* is a pointer to a compiled view description.

`Fvsinit`() sets the associated count member of an element to 0 if the `C` flag was used when the view was compiled, and sets the associated length member to the length of the associated null value if the `L` flag was used in the view file.

`Fvsinit32` is used for views defined with `viewc32` or `VIEW32` typed buffers for larger views with more fields.

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

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

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

`[FBADVIEW]`
"cannot find or get view"
The view description specified was NULL or was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

See Also `Fintro`(3), `Fvselinit`(3), `viewfile`(5)

Fvstof (3FML)

Name

Fvstof, Fvstof32-copy from C structure to fielded buffer

Synopsis

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

int
Fvstof(FBFR *fbfr, char *cstruct, int mode, char *view)

#include "fml32.h"

int
Fvstof32(FBFR32 *fbfr, char *cstruct, int mode, char *view)
```

Description

Fvstof() transfers data from a C structure to a fielded buffer. *fbfr* is a pointer to a fielded buffer. *cstruct* is a pointer to a C structure. *mode* specifies the manner in which the transfer is made. *view* is a pointer to a compiled view description. *mode* has four possible values:

- FUPDATE
- FOJOIN
- FJOIN
- FCONCAT

The action of these modes are the same as that described in Fupdate(3), Fojoin(3), Fjoin(3), and Fconcat(3). One can even think of Fvstof() as the same as these functions, except that where they specify a source buffer, Fvstof() specifies a C structure. Bear in mind that FUPDATE does not move structure elements that have null values.

Fvstof32 is used for views defined with viewc32 or VIEW32 typed buffers for larger views with more fields.

Return Values

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

Errors Under the following conditions, `Fvstof`() 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 *cstruct* parameter or an invalid *mode* to `Fvstof`)

`[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.

`[FBADACM]`
"ACM contains negative value"
An Associated Count Member should not be a negative value while transferring data from a structure to a fielded buffer.

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

See Also `Fintro`(3), `Fconcat`(3), `Fjoin`(3), `Fojoin`(3), `Fupdate`(3), `Fvftos`(3)

Fvstot (3FML)

Name

`Fvstot`, `Fvttos`-convert C structure to/from target record type

Synopsis

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

long
Fvstot(char *cstruct, char *trecord, long treclen, char *viewname)

long
Fvttos(char *cstruct, char *trecord, char *viewname)

#include "fml32.h"

int
Fvstot32(char *cstruct, char *trecord, long treclen, char
*viewname)

int
Fvttos32(char *cstruct, char *trecord, char *viewname)

int Fcodeset(char *translation_table)
```

Description

The `Fvstot`() function transfers data from a C structure to a target record type. The `Fvttos`() function transfers data from a target record to a C structure. *trecord* is a pointer to the target record. *cstruct* is a pointer to a C structure. *viewname* is a pointer to the name of a compiled view description. The `VIEWDIR` and `VIEWFILES` are used to find the directory and file containing the compiled view description.

`Fvttos32` and `Fvstot32` are used with 32-bit VIEWS.

To convert from an FML buffer to a target record, first call `Fvftos` to convert the FML buffer to a C structure, and call `Fvstot` to convert to a target record. To convert from a target record to an FML buffer, first call `Fvttos` to convert to a C structure and then call `Fvstof` to convert the structure to an FML buffer.

Default Conversion-IBM /370

The default target is IBM/370 COBOL records. The default data conversion is done based on the following table.

Default Data Conversion

Struct	Record
float	COMP-1
double	COMP-2
long	S9(9) COMP
short	S9(4) COMP
int	S9(9) COMP or S9(4) COMP
dec_t(*m*, *n*)	S9(2**m*-(*n*+1))V9(*n*)COMP-3
ASCII char	EBCDIC char
ASCII string	EBCDIC string
carray	character array

No filler bytes are provided between fields in the IBM/370 record. The COBOL SYNC clause should not be specified for any data items that are a part of the structure corresponding to the view.

An integer field is converted to either a four or two-byte integer depending on the size of integers on the machine on which the conversion is done.

A string field in the view must be terminated with a null when converting to/from the IBM/370 format.

The data in a carray field is passed unchanged; no data translation is performed.

Packed decimals exist in the IBM/370 environment as two decimal digits packed into one byte with the low-order half byte used to store the sign. The length of a packed decimal may be 1 to 16 bytes with storage available for 1 to 31 digits and a sign.

Packed decimals are supported in C structures using the dec_t field type. The dec_t field has a defined size consisting of two numbers separated by a comma. The number to the left of the comma is the total number of bytes that the decimal occupies. The number to the right is the number of digits to the right of the decimal point. The formula for conversion is:

`dec_t(`*m, n*`) => S9(2*`*m*`-(`*n*`+1))V9(`*n*`)COMP-3`

Decimal values may be converted to and from other data types (e.g., int, long, string, double, and float) using the functions described in `decimal`(3).

The following table provides the hex values for default character conversion of ASCII (on the left) to/from EBCDIC (on the right).

```
| 00 00 |01 01 |02 02 |03 03 |04 37 |05 2d |06 2e |07 2f |
| 08 16 |09 05 |0a 25 |0b 0b |0c 0c |0d 0d |0e 0e |0f 0f |
| 10 10 |11 11 |12 12 |13 13 |14 3c |15 3d |16 32 |17 26 |
| 18 18 |19 19 |1a 3f |1b 27 |1c 1c |1d 1d |1e 1e |1f 1f |
| 20 40 |21 5a |22 7f |23 7b |24 5b |25 6c |26 50 |27 7d |
| 28 4d |29 5d |2a 5c |2b 4e |2c 6b |2d 60 |2e 4b |2f 61 |
| 30 f0 |31 f1 |32 f2 |33 f3 |34 f4 |35 f5 |36 f6 |37 f7 |
| 38 f8 |39 f9 |3a 7a |3b 5e |3c 4c |3d 7e |3e 6e |3f 6f |
| 40 7c |41 c1 |42 c2 |43 c3 |44 c4 |45 c5 |46 c6 |47 c7 |
| 48 c8 |49 c9 |4a d1 |4b d2 |4c d3 |4d d4 |4e d5 |4f d6 |
| 50 d7 |51 d8 |52 d9 |53 e2 |54 e3 |55 e4 |56 e5 |57 e6 |
| 58 e7 |59 e8 |5a e9 |5b ad |5c e0 |5d bd |5e 5f |5f 6d |
| 60 79 |61 81 |62 82 |63 83 |64 84 |65 85 |66 86 |67 87 |
| 68 88 |69 89 |6a 91 |6b 92 |6c 93 |6d 94 |6e 95 |6f 96 |
| 70 97 |71 98 |72 99 |73 a2 |74 a3 |75 a4 |76 a5 |77 a6 |
| 78 a7 |79 a8 |7a a9 |7b c0 |7c 6a |7d d0 |7e a1 |7f 07 |
| 80 20 |81 21 |82 22 |83 23 |84 24 |85 15 |86 06 |87 17 |
| 88 28 |89 29 |8a 2a |8b 2b |8c 2c |8d 09 |8e 0a |8f 1b |
| 90 30 |91 31 |92 1a |93 33 |94 34 |95 35 |96 36 |97 08 |
| 98 38 |99 39 |9a 3a |9b 3b |9c 04 |9d 14 |9e 3e |9f e1 |
| a0 41 |a1 42 |a2 43 |a3 44 |a4 45 |a5 46 |a6 47 |a7 48 |
| a8 49 |a9 51 |aa 52 |ab 53 |ac 54 |ad 55 |ae 56 |af 57 |
| b0 58 |b1 59 |b2 62 |b3 63 |b4 64 |b5 65 |b6 66 |b7 67 |
| b8 68 |b9 69 |ba 70 |bb 71 |bc 72 |bd 73 |be 74 |bf 75 |
| c0 76 |c1 77 |c2 78 |c3 80 |c4 8a |c5 8b |c6 8c |c7 8d |
| c8 8e |c9 8f |ca 90 |cb 9a |cc 9b |cd 9c |ce 9d |cf 9e |
| d0 9f |d1 a0 |d2 aa |d3 ab |d4 ac |d5 4a |d6 ae |d7 af |
| d8 b0 |d9 b1 |da b2 |db b3 |dc b4 |dd b5 |de b6 |df b7 |
| e0 b8 |e1 b9 |e2 ba |e3 bb |e4 bc |e5 4f |e6 be |e7 bf |
| e8 ca |e9 cb |ea cc |eb cd |ec ce |ed cf |ee da |ef db |
| f0 dc |f1 dd |f2 de |f3 df |f4 ea |f5 eb |f6 ec |f7 ed |
| f8 ee |f9 ef |fa fa |fb fb |fc fc |fd fd |fe fe |ff ff |
```

An alternate character translation table can be used at run-time by calling `Fcodeset`(). The *translation_table* must point to 512 bytes of binary data. The first 256 bytes of data are interpreted as the ASCII to EBCDIC translation table. The second 256 bytes of data are intrepreted as the EBCDIC to ASCII table. Any data after the 512th byte is ignored. If the pointer is NULL, the default translation is used.

Return Values

On success, `Fvstot` returns the length of the target record and `Fvttos` returns the length of the C structure.

These functions return -1 on error and set `Ferror` to indicate the error condition.

Errors

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

`[FEINVAL]`
"invalid argument to function"
One of the arguments to the function invoked was invalid, (for example, specifying a NULL *trecord* or *cstruct* parameter to `Fvttos`) This error is also returned if a value is out of range when converting to or from a target record.

`[FBADACM]`
"ACM contains negative value"
An Associated Count Member cannot be a negative value.

`[FBADVIEW]`
"cannot find or get view"
viewname was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

`[FNOSPACE]`
"no space in buffer"
The target record is not large enough to hold the converted structure.

`[FVFOPEN]`
"cannot find or open view file"
While trying to find *viewname*, the program failed to find one of the files specified by `VIEWDIR` or `VIEWFILES`.

`[FEUNIX]`
"operating system error"
While trying to find *viewname*, the program failed to open one of the files specified by `VIEWDIR` or `VIEWFILES` for reading.

[FVFSYNTAX]
"bad viewfile"
While trying to find *viewname*, one of the files specified by VIEWDIR or VIEWFILES was corrupted or not a view file.

[FMALLOC]
"malloc failed"
While trying to find *viewname*, malloc() failed while allocating space to hold the view information.

Example

```
VIEW test.v

VIEW test
#type   cname   fbname  count   flag    size    null
float   float1  FLOAT1  1       -       -       0.0
double  double1 DOUBLE1 1       -       -       0.0
long    long1   LONG1   1       -       -       0
short   short1  SHORT1  1       -       -       0
int     int1    INT1    1       -       -       0
dec_t   dec1    DEC1    1       -       4,2     0
char    char1   CHAR1   1       -       -       ''
string  string1 STRING1 1       -       20      ''
carray  carray1 CARRAY1 1       -       20      ''
END

Equivalent COBOL Record

02 OUTPUT-REC.
        05 FLOAT1                          USAGE IS COMP-1.
        05 DOUBLE1                         USAGE IS COMP-2.
       05 LONG1                          PIC S9(9) USAGE IS COMP.
       05 SHORT1                         PIC S9(4) USAGE IS COMP.
       05 INT1                           PIC S9(9) USAGE IS COMP.
        05 DEC1                            PIC S9(5)V9(2) COMP-3.
        05 CHAR1                           PIC X(01).
        05 STRING1                         PIC X(20).
        05 CARRAY1                         PIC X(20).

C Program

#include "test.h"
#include "decimal.h"

main()
{

        struct test s1;
        char data[100];
```

```
        s1.float1 = 1.0;
        s1.double1 = 2.0;
        s1.long1 = 3;
        s1.short1 = 4;
        s1.int1 = 5;
        deccvdbl(6.0,s1.dec1);
        s1.char1 = '7';
        (void) strcpy(s1.string1, "eight");
        (void) strcpy(s1.carray1, "nine");

        if (Fvstot((char *)&s1, data, reclen, "test") == -1) {
          printf("Fvstot failed: %sn", Fstrerror(Ferror));
          exit(0);
        }
        /* transfer to target machine and get response */
        ...

        /* translate back */
        if (Fvttos(data, (char *)&s1, "test") == -1) {
          printf("Fvttos failed: %sn", Fstrerror(Ferror));
          exit(0);
        }

        /* use the structure */
        .....
        exit(0);
}
```

See Also `Fintro`(3), `Fvftos`(3), `Fvstof`(3), `decimal`(3), `viewfile`(5)

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.

Portability This function is not supported using the BEA TUXEDO system /WS DLL for OS/2 and Microsoft Windows.

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