

# **Oracle® Tuxedo**

ATMI FML Function Reference

12c Release 2 (12.1.3)

April 2014

**ORACLE®**

Copyright © 1996, 2014, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Contents

## Section 3fml - FML Functions

Introduction to FML Functions . . . . .	5
CFadd, CFadd32(3fml) . . . . .	11
CFchg, CFchg32(3fml) . . . . .	13
CFfind, CFfind32(3fml) . . . . .	15
CFfindocc, CFfindocc32(3fml) . . . . .	17
CFget, CFget32(3fml) . . . . .	19
CFgetalloc, CFgetalloc32(3fml) . . . . .	22
F_error, F_error32(3fml) . . . . .	24
F32to16, F16to32(3fml) . . . . .	26
Fadd, Fadd32(3fml) . . . . .	28
Fadds, Fadds32(3fml) . . . . .	31
Falloc, Falloc32(3fml) . . . . .	33
Fappend, Fappend32(3fml) . . . . .	35
Fboolco, Fboolco32, Fvboolco, Fvboolco32(3fml) . . . . .	38
Fboolev, Fboolev32, Fvboolev, Fvboolev32(3fml) . . . . .	41
Fboolpr, Fboolpr32, Fvboolpr, Fvboolpr32(3fml) . . . . .	44
Fchg, Fchg32(3fml) . . . . .	46
Fchgs, Fchgs32(3fml) . . . . .	49
Fchksum, Fchksum32(3fml) . . . . .	51
Fcmp, Fcmp32(3fml) . . . . .	53
Fconcat, Fconcat32(3fml) . . . . .	55

Fcpy, Fcpy32(3fml) . . . . .	57
Fdel, Fdel32(3fml) . . . . .	59
Fdelall, Fdelall32(3fml) . . . . .	61
Fdelete, Fdelete32(3fml) . . . . .	63
Fextread, Fextread32(3fml) . . . . .	65
Ffind, Ffind32(3fml) . . . . .	68
Ffindlast, Ffindlast32(3fml) . . . . .	70
Ffindocc, Ffindocc32(3fml) . . . . .	72
Ffinds, Ffinds32(3fml) . . . . .	75
Ffloatev, Ffloatev32, Fvfloatev, Fvfloatev32(3fml) . . . . .	77
Ffprint, Ffprint32(3fml) . . . . .	80
Fsprint, Fsprint32(3fml) . . . . .	81
Ffree, Ffree32(3fml) . . . . .	84
Fget, Fget32(3fml) . . . . .	86
Fgetalloc, Fgetalloc32(3fml) . . . . .	88
Fgetlast, Fgetlast32(3fml) . . . . .	90
Fgets, Fgets32(3fml) . . . . .	92
Fgetsa, Fgetsa32(3fml) . . . . .	94
Fidnm_unload, Fidnm_unload32(3fml) . . . . .	96
Fidxused, Fidxused32(3fml) . . . . .	97
Fielded, Fielded32(3fml) . . . . .	99
Findex, Findex32(3fml) . . . . .	100
Finit, Finit32(3fml) . . . . .	102
Fjoin, Fjoin32(3fml) . . . . .	104
Fldid, Fldid32(3fml) . . . . .	106
Fldno, Fldno32(3fml) . . . . .	108
Fldtype, Fldtype32(3fml) . . . . .	109
Flen, Flen32(3fml) . . . . .	110

Fmbpack32(3fml) . . . . .	112
Fmbunpack32(3fml) . . . . .	114
Fmkfldid, Fmkfldid32(3fml) . . . . .	116
Fmove, Fmove32(3fml) . . . . .	118
Fname, Fname32(3fml) . . . . .	120
Fneeded, Fneeded32(3fml) . . . . .	122
Fnext, Fnext32(3fml) . . . . .	124
Fnmid_unload, Fnmid_unload32(3fml) . . . . .	126
Fnum, Fnum32(3fml) . . . . .	127
Foccur, Foccur32(3fml) . . . . .	129
Fjoin, Fjoin32(3fml) . . . . .	131
Fpres, Fpres32(3fml) . . . . .	133
Fprint, Fprint32(3fml) . . . . .	134
Fproj, Fproj32(3fml) . . . . .	136
Fprojcpy, Fprojcpy32(3fml) . . . . .	138
Fread, Fread32(3fml) . . . . .	140
Frealloc, Frealloc32(3fml) . . . . .	142
Frstrindex, Frstrindex32(3fml) . . . . .	144
Fsizeof, Fsizeof32(3fml) . . . . .	147
Fstrerror, Fstrerror32(3fml) . . . . .	149
Ftypecv, Ftypecv32(3fml) . . . . .	150
Ftype, Ftype32(3fml) . . . . .	152
Funindex, Funindex32(3fml) . . . . .	154
Funused, Funused32(3fml) . . . . .	156
Fupdate, Fupdate32(3fml) . . . . .	158
Fused, Fused32(3fml) . . . . .	160
Fvall, Fvall32(3fml) . . . . .	162
Fvals, Fvals32(3fml) . . . . .	164

Fvftos, Fvftos32(3fml) . . . . .	166
Fvneeded, Fvneeded32(3fml). . . . .	168
Fvnull, Fvnull32(3fml). . . . .	169
Fvopt, Fvopt32(3fml) . . . . .	171
Fvrefresh, Fvrefresh32(3fml) . . . . .	173
Fvselinit, Fvselinit32(3fml) . . . . .	174
Fvsinit, Fvsinit32(3fml) . . . . .	176
Fvstof, Fvstof32(3fml) . . . . .	178
Fvstot, Fvttos(3fml) . . . . .	180
Fwrite, Fwrite32(3fml). . . . .	187
tpconvfmb32(3fml) . . . . .	189
tpconvvmb32(3fml) . . . . .	191

# Section 3fml - FML Functions

**Table 1 Oracle Tuxedo ATMI FML Functions**

Name	Description
<a href="#">Introduction to FML Functions</a>	Provides an introduction to the FML functions
<a href="#">CFadd, CFadd32(3fml)</a>	Converts and adds field
<a href="#">CFchg, CFchg32(3fml)</a>	Converts and changes field
<a href="#">CFfind, CFfind32(3fml)</a>	Finds, converts, and returns pointer
<a href="#">CFfindocc, CFfindocc32(3fml)</a>	Finds occurrence of converted value
<a href="#">CFget, CFget32(3fml)</a>	Gets field and converts
<a href="#">CFgetalloc, CFgetalloc32(3fml)</a>	Gets field, allocates space, and converts
<a href="#">F_error, F_error32(3fml)</a>	Prints error message for last error
<a href="#">F32to16, F16to32(3fml)</a>	Converts 16-bit FML to/from 32-bit FML buffer
<a href="#">Fadd, Fadd32(3fml)</a>	Adds new field occurrence
<a href="#">Fadds, Fadds32(3fml)</a>	Converts value from type FLD_STRING and adds to buffer
<a href="#">Falloc, Falloc32(3fml)</a>	Allocates and initializes fielded buffer

**Table 1 Oracle Tuxedo ATMI FML Functions (Continued)**

<b>Name</b>	<b>Description</b>
<code>Fappend, Fappend32(3fml)</code>	Appends new field occurrence
<code>Fboolco, Fboolco32, Fvboolco, Fvboolco32(3fml)</code>	Compiles expression, returns evaluation tree
<code>Fboolev, Fboolev32, Fvboolev, Fvboolev32(3fml)</code>	Evaluates buffer against tree
<code>Fboolpr, Fboolpr32, Fvboolpr, Fvboolpr32(3fml)</code>	Prints Boolean expression as parsed
<code>Fchg, Fchg32(3fml)</code>	Changes field occurrence value
<code>Fchgs, Fchgs32(3fml)</code>	Changes field occurrence - caller presents string
<code>Fchksum, Fchksum32(3fml)</code>	Computes checksum for fielded buffer
<code>Fcmp, Fcmp32(3fml)</code>	Compares two fielded buffers
<code>Fconcat, Fconcat32(3fml)</code>	Concatenates source to destination buffer
<code>Fcpy, Fcpy32(3fml)</code>	Copies source to destination buffer
<code>Fdel, Fdel32(3fml)</code>	Deletes field occurrence from buffer
<code>Fdelall, Fdelall32(3fml)</code>	Deletes all field occurrences from buffer
<code>Fdelete, Fdelete32(3fml)</code>	Deletes list of fields from buffer
<code>Fextread, Fextread32(3fml)</code>	Builds fielded buffer from printed format
<code>Ffind, Ffind32(3fml)</code>	Finds field occurrence in buffer
<code>Ffindlast, Ffindlast32(3fml)</code>	Finds last occurrence of field in buffer
<code>Ffindocc, Ffindocc32(3fml)</code>	Finds occurrence of field value
<code>Ffinds, Ffinds32(3fml)</code>	Returns ptr to string representation
<code>Ffloatev, Ffloatev32, Fvfloatev, Fvfloatev32(3fml)</code>	Returns value of expression as a double
<code>Ffprint, Ffprint32(3fml)</code>	Prints fielded buffer to specified stream
<code>Fsprint, Fsprint32(3fml)</code>	Prints fielded buffer to a buffer

**Table 1 Oracle Tuxedo ATMI FML Functions (Continued)**

<b>Name</b>	<b>Description</b>
<code>Ffree, Ffree32(3fml)</code>	Frees space allocated for fielded buffer
<code>Fget, Fget32(3fml)</code>	Gets copy and length of field occurrence
<code>Fgetalloc, Fgetalloc32(3fml)</code>	Allocates space and gets copy of field occurrence
<code>Fgetlast, Fgetlast32(3fml)</code>	Gets copy of last occurrence
<code>Fgets, Fgets32(3fml)</code>	Gets value converted to string
<code>Fgetsa, Fgetsa32(3fml)</code>	Uses <code>malloc()</code> to allocate space and gets converted value
<code>Fidnm_unload, Fidnm_unload32(3fml)</code>	Recovers space from <code>id-&gt;nm</code> mapping tables
<code>Fidxused, Fidxused32(3fml)</code>	Returns amount of space used
<code>Fielded, Fielded32(3fml)</code>	Returns <code>true</code> if buffer is fielded
<code>Findex, Findex32(3fml)</code>	Indexes a fielded buffer
<code>Finit, Finit32(3fml)</code>	Initializes fielded buffer
<code>Fjoin, Fjoin32(3fml)</code>	Joins source into destination buffer
<code>Fldid, Fldid32(3fml)</code>	Maps field name to field identifier
<code>Fldno, Fldno32(3fml)</code>	Maps field identifier to field number
<code>Fldtype, Fldtype32(3fml)</code>	Maps field identifier to field type
<code>Flen, Flen32(3fml)</code>	Returns <code>len</code> of field occurrence in buffer
<code>Fmbpack32(3fml)</code>	Prepares encoding name and multibyte data information
<code>Fmbunpack32(3fml)</code>	Extracts encoding name and multibyte data information
<code>Fmkfldid, Fmkfldid32(3fml)</code>	Makes a field identifier
<code>Fmove, Fmove32(3fml)</code>	Moves fielded buffer to destination
<code>Fname, Fname32(3fml)</code>	Maps field identifier to field name
<code>Fneeded, Fneeded32(3fml)</code>	Computes size needed for buffer

**Table 1 Oracle Tuxedo ATMI FML Functions (Continued)**

Name	Description
<code>Fnext, Fnext32(3fml)</code>	Gets next field occurrence
<code>Fnmid_unload, Fnmid_unload32(3fml)</code>	Recovers space from <i>nm</i> -> <i>id</i> mapping tables
<code>Fnum, Fnum32(3fml)</code>	Returns count of all occurrences in buffer
<code>Foccur, Foccur32(3fml)</code>	Returns count of field occurrences in buffer
<code>Fojoin, Fojoin32(3fml)</code>	Outer join of source into destination buffer
<code>Fpres, Fpres32(3fml)</code>	True if field occurrence is present in buffer
<code>Fprint, Fprint32(3fml)</code>	Prints buffer to standard output
<code>Fproj, Fproj32(3fml)</code>	Provides projection on buffer
<code>Fprojcpy, Fprojcpy32(3fml)</code>	Provides projection and copy on buffer
<code>Fread, Fread32(3fml)</code>	Reads fielded buffer
<code>Frealloc, Frealloc32(3fml)</code>	Reallocates fielded buffer
<code>Frstrindex, Frstrindex32(3fml)</code>	Restores index in a buffer
<code>Fsizeof, Fsizeof32(3fml)</code>	Returns size of fielded buffer
<code>Fstrerror, Fstrerror32(3fml)</code>	Gets error message string for FML error
<code>Ftypcv, Ftypcv32(3fml)</code>	Converts from one field type to another
<code>Ftype, Ftype32(3fml)</code>	Returns pointer to type of field
<code>Funindex, Funindex32(3fml)</code>	Discards fielded buffer's index
<code>Funused, Funused32(3fml)</code>	Returns number of unused bytes in fielded buffer
<code>Fupdate, Fupdate32(3fml)</code>	Updates destination buffer with source
<code>Fused, Fused32(3fml)</code>	Returns number of used bytes in fielded buffer
<code>Fvall, Fvall32(3fml)</code>	Returns long value of field occurrence
<code>Fvals, Fvals32(3fml)</code>	Returns string value of field occurrence
<code>Fvftos, Fvftos32(3fml)</code>	Copies from fielded buffer to C structure

**Table 1 Oracle Tuxedo ATMI FML Functions (Continued)**

Name	Description
<code>Fvneeded, Fvneeded32(3fml)</code>	Computes size needed for view buffer
<code>Fvnull, Fvnull32(3fml)</code>	Checks if a structure element is NULL
<code>Fvopt, Fvopt32(3fml)</code>	Changes flag options of a mapping entry
<code>Fvrefresh, Fvrefresh32(3fml)</code>	Copies from C structure to fielded buffer
<code>Fvselinit, Fvselinit32(3fml)</code>	Initializes structure element to NULL
<code>Fvsinit, Fvsinit32(3fml)</code>	Initializes C structure to NULL
<code>Fvstof, Fvstof32(3fml)</code>	Copies from C structure to fielded buffer
<code>Fvstot, Fvttos(3fml)</code>	Converts C structure to/from target record type
<code>Fwrite, Fwrite32(3fml)</code>	Writes fielded buffer
<code>tpconvfmb32(3fml)</code>	Converts multibyte characters in an <code>FLD_MBSTRING</code> field in an FML32 typed buffer to a named target encoding.
<code>tpconvvmb32(3fml)</code>	Converts multibyte characters in an <code>MBSTRING</code> field in an <code>VIEW32</code> typed buffer to a named target encoding.

## 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 `fm1.h` which must be included in an application program using the FML16 interface; and functions live in `-lfm1`. 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 `fm132.h`; and functions live in `-lfm132`. All definitions, types, and function names for FML32 have a “32” suffix (for example, `MAXFLEN32`, `FLDID32`, `Fchg32`). Also the environment variables are suffixed with “32” (for example, `FLDTBLDIR32`, `FLDDBLS32`, `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`. The Oracle Tuxedo ATMI system conforms to the following conventions for field numbers:

FML16 Field Numbers		FML32 Field Numbers	
Reserved	Available	Reserved	Available
1-100	101-8191	1-3,999, 6,000-10,000, 30,000,001-33,554,431	4,000-5,999, 10,001-30,000,000

Applications should avoid using the reserved field numbers, although the Oracle Tuxedo ATMI system does not strictly enforce applications from using them.

**Note:** If a field in an FML or FML32 buffer is used for routing, the value of that field must be a number less than or equal to 8191.

The field types can be any of the standard C language types: `short`, `long`, `float`, `double`, and `char`. The following types are also supported: `string` (a series of characters ending with a NULL character), `carray` (a character array), `mbstring` (a multibyte character array—available in Oracle Tuxedo release 8.1 or later), `ptr` (a pointer to a buffer), `fm132` (an embedded FML32 buffer), and `view32` (an embedded VIEW32 buffer). (The `ptr`, `fm132`, and `view32` types are supported only for the FML32 interface.) In `fm1.h`, the supported field types are defined as `FLD_SHORT`, `FLD_LONG`, `FLD_FLOAT`, `FLD_DOUBLE`, `FLD_CHAR`, `FLD_STRING`, and `FLD_CARRAY`. In `fm132.h`, the supported field types are defined as `FLD_SHORT`, `FLD_LONG`, `FLD_FLOAT`, `FLD_DOUBLE`, `FLD_CHAR`, `FLD_STRING`, `FLD_CARRAY`, `FLD_MBSTRING`, `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`.

For FML16, a fielded buffer pointer is of type `FBFR *`, a field length has the type `FLDLN`, 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 `FLDLN32`, 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()` and `mkfldhdr32()` are provided to make header files out of field table files (see [mkfldhdr](#), [mkfldhdr32\(1\)](#) for details). 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`, `FLDLLEN`, 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 `fm1.h` to inclusion of `fm132.h` followed by `fm11632.h`. The `fm11632.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 the ATMI environment that grew out of the FML idea of a fielded buffer. Two of the standard buffer types delivered with the ATMI environment are FML typed buffers and `VIEW` typed buffers. An additional difference of `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()` or `viewc32()`, creating a view object file and can then be used to map data transferred between fielded buffers and C structures in a C program (see [viewc](#), [viewc32\(1\)](#) for details). 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()` or `viewdis32()`, is provided to translate object view descriptions into readable form (that is, back into source view descriptions); the output of the disassembler can be reinput to the view compiler (see [viewdis](#), [viewdis32\(1\)](#) for details).

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 pathnames are searched for through the list of directories specified by the `VIEWDIR` variable (as described later in this section). `VIEWDIR` specifies a colon-separated list of directories to be used to find view object files with relative filenames. 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()` 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(3c)`.

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

## See Also

`CFadd`, `CFadd32(3fml)`, `CFchg`, `CFchg32(3fml)`, `CFfind`, `CFfind32(3fml)`, `CFfindocc`, `CFfindocc32(3fml)`, `CFget`, `CFget32(3fml)`, `CFgetalloc`, `CFgetalloc32(3fml)`, `F_error`, `F_error32(3fml)`, `Fadd`, `Fadd32(3fml)`, `Fadds`, `Fadds32(3fml)`, `Falloc`, `Falloc32(3fml)`, `Fboolco`, `Fboolco32`, `Fvboolco`, `Fvboolco32(3fml)`, `Fboolev`, `Fboolev32`, `Fvboolev`, `Fvboolev32(3fml)`, `Fboolpr`, `Fboolpr32`, `Fvboolpr`, `Fvboolpr32(3fml)`, `Fchg`, `Fchg32(3fml)`, `Fchgs`, `Fchgs32(3fml)`, `Fchksum`, `Fchksum32(3fml)`, `Fcmp`, `Fcmp32(3fml)`, `Fconcat`, `Fconcat32(3fml)`, `Fcpy`,

```

Fcpy32(3fml), Fdel, Fdel32(3fml), Fdelall, Fdelall32(3fml), Fdelete,
Fdelete32(3fml), Fextread, Fextread32(3fml), Ffind, Ffind32(3fml), Ffindlast,
Ffindlast32(3fml), Ffindocc, Ffindocc32(3fml), Ffinds, Ffinds32(3fml),
Ffloatev, Ffloatev32, Fvfloatev, Fvfloatev32(3fml), Ffprint,
Ffprint32(3fml), Ffree, Ffree32(3fml), Fget, Fget32(3fml), Fgetalloc,
Fgetalloc32(3fml), Fgetlast, Fgetlast32(3fml), Fgets, Fgets32(3fml), Fgetsa,
Fgetsa32(3fml), Fidnm_unload, Fidnm_unload32(3fml), Fidxused,
Fidxused32(3fml), Fielded, Fielded32(3fml), Findex, Findex32(3fml), Finit,
Finit32(3fml), Fjoin, Fjoin32(3fml), Fldid, Fldid32(3fml), Fldno,
Fldno32(3fml), Fldtype, Fldtype32(3fml), Flen, Flen32(3fml), Fmkfldid,
Fmkfldid32(3fml), Fmove, Fmove32(3fml), Fname, Fname32(3fml), Fneeded,
Fneeded32(3fml), Fnext, Fnext32(3fml), Fnmid_unload, Fnmid_unload32(3fml),
Fnum, Fnum32(3fml), Foccur, Foccur32(3fml), Fojoin, Fojoin32(3fml), Fpres,
Fpres32(3fml), Fprint, Fprint32(3fml), Fproj, Fproj32(3fml), Fprojcpy,
Fprojcpy32(3fml), Fread, Fread32(3fml), Frealloc, Frealloc32(3fml),
Frstrindex, Frstrindex32(3fml), Fsizeof, Fsizeof32(3fml), Fstrerror,
Fstrerror32(3fml), Ftypcvt, Ftypcvt32(3fml), Ftype, Ftype32(3fml), Funindex,
Funindex32(3fml), Funused, Funused32(3fml), Fupdate, Fupdate32(3fml), Fused,
Fused32(3fml), Fvall, Fvall32(3fml), Fvals, Fvals32(3fml), Fvftos,
Fvftos32(3fml), Fneeded, Fneeded32(3fml), Fvnull, Fvnull32(3fml), Fvopt,
Fvopt32(3fml), Fvselinit, Fvselinit32(3fml), Fvsinit, Fvsinit32(3fml), Fvstof,
Fvstof32(3fml), Fwrite, Fwrite32(3fml), field_tables(5), viewfile(5)

```

*Programming an Oracle Tuxedo ATMI Application Using FML*

## CFadd, CFadd32(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 first converts 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 the type supplied by the user to the type specified 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`.

This function fails if any of the following field types is used: `FLD_PTR`, `FLD_MBSTRING`, `FLD_FML32`, or `FLD_VIEW32`. If one of these field types is encountered when `CFadd()` or `CFadd32()` is being used, `Error` is set to `FEBADOP`.

`CFadd32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `CFadd()` or `CFadd32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `CFadd()` fails and sets `Error` 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()` failed when converting from a carray (or mbstring) 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.

**[FEBADOP]**

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`) is specified.

## See Also

[Introduction to FML Functions](#), [Fadd](#), [Fadd32\(3fml\)](#)

# CFchg, CFchg32(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 (for example, 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).

This function fails if any of the following field types is used: `FLD_PTR`, `FLD_MBSTRING`, `FLD_FML32`, or `FLD_VIEW32`. If one of these field types is encountered when `CFchg()` or `CFchg32()` is being used, `Error` is set to `FEBADOP`.

`CFchg32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `CFchg()` or `CFchg32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `CFchg()` fails and sets `Error` 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()` failed when converting from a `carray` (or `mbstring`) 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.

**[FEBADOP]**

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`) is specified.

## See Also

[Introduction to FML Functions](#), [CFadd](#), [CFadd32\(3fml\)](#), [Fchg](#), [Fchg32\(3fml\)](#)

# CFind, CFind32(3fml)

## Name

`CFind()`, `CFind32()` - find, convert, and return pointer

## Synopsis

```
#include <stdio.h>
#include "fml.h"
char * CFind(FBFR *fbfr, FLDID fieldid, FLDOCC oc, FLDLEN *len,
             int type)
#include "fml32.h"
```

```
char *
CFfind32(FBFR32 *fbfr, FLDID32 fieldid, FLD OCC32 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()`, 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 `Ffins()`, which is guaranteed until the next modification of the buffer. Unlike `Ffind()`, `CFfind()` aligns the converted value for immediate use by the caller.

This function fails if any of the following field types is used: `FLD_PTR`, `FLD_MBSTRING`, `FLD_FML32`, or `FLD_VIEW32`. If one of these field types is encountered when `CFfind()` or `CFfind32()` is being used, `Error` is set to `FEBADOP`.

`CFfind32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `CFfind()` or `CFfind32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `CFfind()` fails and sets `Error` 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()` failed when converting from a carry (or mbstring) 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.

**[FEBADOP]**

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`) is specified.

See Also

[Introduction to FML Functions](#), [Ffind](#), [Ffind32\(3fml\)](#)

## CFfindocc, CFfindocc32(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 `Fffindocc()` 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 `FLD_CARRAY`. *type* is the data type of the field in *value*.

This function fails if any of the following field types is used: `FLD_PTR`, `FLD_MBSTRING`, `FLD_FML32`, or `FLD_VIEW32`. If one of these field types is encountered when `Cffindocc()` or `Cffindocc32()` is being used, `Error` is set to `FEBADOP`.

`Cffindocc32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Cffindocc()` or `Cffindocc32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

If the field value is not found or if other errors are detected, -1 is returned and `Cffindocc()` sets `Error` to indicate the error condition.

## Errors

Under the following conditions, `Cffindocc()` fails and sets `Error` 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()` failed when converting from a carray (or mbstring) 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.

**[FEBADOP]**

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`) is specified.

## See Also

[Introduction to FML Functions](#), [Ffindocc](#), [Ffindocc32\(3fml\)](#)

# CFget, CFget32(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()`. 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.

This function fails if any of the following field types is used: `FLD_PTR`, `FLD_MBSTRING`, `FLD_FML32`, or `FLD_VIEW32`. If one of these field types is encountered when `CFget()` or `CFget32()` is being used, `Error` is set to `FEBADOP`.

`CFget32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `CFget()` or `CFget32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `CFget()` fails and sets `Error` 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()` failed when converting from a carry (or mbstring) 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.

**[FEBADOP]**

"invalid field type"

An invalid field type (such as FLD\_PTR, FLD\_FML32, and FLD\_VIEW32) is specified.

## See Also

[Introduction to FML Functions](#), [Fget](#), [Fget32\(3fml\)](#)

# CFgetalloc, CFgetalloc32(3fml)

## Name

CFgetalloc(), CFgetalloc32() - get field, allocate 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.

This function fails if any of the following field types is used: FLD\_PTR, FLD\_MBSTRING, FLD\_FML32, or FLD\_VIEW32. If one of these field types is encountered when CFgetalloc() or CFgetalloc32() is being used, *Error* is set to FEBADOP.

CFgetalloc32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to CFgetalloc() or CFgetalloc32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

On success, CFgetalloc() returns a pointer to the converted value. On error, the function returns NULL and sets *Error* to indicate the error condition.

## Errors

Under the following conditions, `CFgetalloc()` fails and sets `Error` 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()` 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.

**[FEBADOP]**

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`) is specified.

## See Also

[Introduction to FML Functions](#), `Fgetalloc`, `Fgetalloc32(3fml)`

# F\_error, F\_error32(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()` 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()` is called.

`F_error32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `F_error()` or `F_error32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## See Also

[Introduction to FML Functions](#)

perror(3), Uunix\_err(3) in a UNIX system reference manual

# F32to16, F16to32(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, carray, and mbstring 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 reissued 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 `Error32` on error.

`F32to16()` lives in the FML library or shared object and sets `Error` 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.

`F32to16()` fails with `FBADFLD` for the following field types: `FLD_PTR`, `FLD_MBSTRING`, `FLD_FML32`, or `FLD_VIEW32`. `F16to32()` has no impact when performed on these field types.

A thread in a multithreaded application may issue a call to `F32to16()` or `F16to32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

This function returns -1 on error and sets `Error` 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"

This error occurs only for the `F32to16()` function. The source buffer has a field identifier for which the field type is not one of the eight types supported by 16-bit FML, or the field number is greater than 8191.

## See Also

[Introduction to FML Functions](#)

# Fadd, Fadd32(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 or FLD\_MBSTRING.

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() 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\_PTR, Fadd32() stores the pointer value. The buffer pointed to by a FLD\_PTR field must be allocated using the tmalloc() call. For values of type FLD\_FML32, Fadd32() stores the entire FLD\_FML32 field value, except the index. For values of type FLD\_VIEW32, Fadd() stores a pointer to a structure of type FVIEWFLD, which contains *vflags* (a flags field, currently unused and set to 0), *vname* (a character array containing the viewname), and *data* (a pointer to the view data stored as a C structure). The application provides the *vname* and *data* to Fadd32().

For values of type `FLD_MBSTRING`, the value is the *packed* output argument of the `Fmbpack32()` function, and the *len* argument is the length of the value in the *size* output argument of `Fmbpack32()`.

For values of type `FLD_CARRAY`, the *len* argument is the length of the value. For all types other than `FLD_CARRAY` or `FLD_MBSTRING`, the length of the object referenced by *value* is inferred from its type (for example, 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.

A thread in a multithreaded application may issue a call to `Fadd()` or `Fadd32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fadd()` fails and sets `Error` 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

[Introduction to FML Functions](#), [CFadd](#), [CFadd32 \(3fml\)](#), [Fadds](#), [Fadds32 \(3fml\)](#), [Fchg](#), [Fchg32 \(3fml\)](#)

# Fadds, Fadds32(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.

A thread in a multithreaded application may issue a call to Fadds() or Fadds32() while running in any context state, including TPINVALIDCONTEXT.

## 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()` failed during conversion of `carray` (or `mbstring`) to string.

[FBADFLD]

"unknown field number or type"

A field identifier is specified which is not valid.

## See Also

[Introduction to FML Functions](#), [CFchg](#), [CFchg32\(3fml\)](#), [CFfind](#), [CFfind32\(3fml\)](#), [CFget](#), [CFget32\(3fml\)](#), [Falloc](#), [Falloc32\(3fml\)](#), [Fchgs](#), [Fchgs32\(3fml\)](#), [Ffinds](#), [Ffinds32\(3fml\)](#), [Fgets](#), [Fgets32\(3fml\)](#), [Fgetsa](#), [Fgetsa32\(3fml\)](#)

# Falloc, Falloc32(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()` 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.

A thread in a multithreaded application may issue a call to `Falloc()` or `Falloc32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

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

[`FMALLOC`]

"malloc failed"

Allocation of space dynamically using `malloc()` 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

[Introduction to FML Functions](#), [Ffree](#), [Ffree32\(3fml\)](#), [Fielded](#), [Fielded32\(3fml\)](#), [Finit](#), [Finit32\(3fml\)](#), [Fneeded](#), [Fneeded32\(3fml\)](#), [Frealloc](#), [Frealloc32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#), [Funused](#), [Funused32\(3fml\)](#)

[malloc\(3\)](#) in a UNIX system reference manual

# Fappend, Fappend32(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 the given 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() and Funindex(), or calls to Free(), Fused(), Funused() and Fsizeof(). Calls to other FML routines made before calling Findex() or Funindex() 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 or FLD\_MBSTRING.

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() must be used).

In the "Synopsis" section above the *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_MBSTRING`, the *value* is the *packed* output argument of the `Fmbpack32()` function, and the *len* argument is the length of the value in the *size* output argument of `Fmbpack32()`.

For values of type `FLD_CARRAY`, the *len* argument is the length of the value. For all types other than `FLD_CARRAY` or `FLD_MBSTRING`, the length of the object referenced by *value* is inferred from its type (for example, 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.

A thread in a multithreaded application may issue a call to `Fappend()` or `Fappend32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fappend()` fails and sets `Error` 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.

[FEADFLD]

"unknown field number or type"

A field number is specified which is not valid.

## See Also

[Introduction to FML Functions](#), [Fadd](#), [Fadd32\(3fml\)](#), [Ffree](#), [Ffree32\(3fml\)](#), [Findex](#), [Findex32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#), [Funindex](#), [Funindex32\(3fml\)](#), [Funused](#), [Funused32\(3fml\)](#), [Fused](#), [Fused32\(3fml\)](#)

# Fboolco, Fboolco32, Fvboolco, Fvboolco32(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 grammar can be found in the *Programming an Oracle Tuxedo ATMI Application Using FML*.

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 functionality for views. The *viewname* parameter indicates the view from which the field offsets are taken.

This function fails if any of the following field types is used: FLD\_PTR, FLD\_MBSTRING, FLD\_FML32, or FLD\_VIEW32. If one of these field types is encountered, *Error* is set to FEBADOP.

These functions are not supported on Workstation platforms.

A thread in a multithreaded application may issue a call to any of the functions documented here—Fboolco(), Fboolco32(), Fvboolco(), or Fvboolco32()—while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, `Fboolco()` fails and sets `Error` to:

### [FMALLOC]

"malloc failed"

Allocation of space dynamically using `malloc()` 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 viewfiles.

### [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 viewfile"

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 viewfile.

[FMALLOC]

"malloc failed"

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

[FEBADOP]

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32`) is specified.

## 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");
```

This example 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](#), [Fboolev32](#), [Fvboolev](#), [Fvboolev32\(3fml\)](#), [Fboolpr](#), [Fboolpr32](#), [Fvboolpr](#), [Fvboolpr32\(3fml\)](#), [Fldid](#), [Fldid32\(3fml\)](#)

# Fboolev, Fboolev32, Fvboolev, Fvboolev32(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().

Fboolev32() is used with 32-bit FML.

Fvboolev() and Fvboolev32() provide the same functionality 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.

A thread in a multithreaded application may issue a call to any of the functions documented here—Fboolev(), Fboolev32(), Fvboolev(), or Fvboolev32()—while running in any context state, including TPINVALIDCONTEXT.

## 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 *Error* to indicate the error condition.

## Errors

Under the following conditions, `Fboolev()` fails and sets `Error` 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()` 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 viewfile"

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 viewfile.

```
[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()`:

```
#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

[Introduction to FML Functions](#), [Fboolco](#), [Fboolco32](#), [Fvboolco](#), [Fvboolco32\(3fml\)](#), [Fboolpr](#), [Fboolpr32](#), [Fvboolpr](#), [Fvboolpr32\(3fml\)](#)

# Fboolpr, Fboolpr32, Fvboolpr, Fvboolpr32(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(). *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 functionality 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.

A thread in a multithreaded application may issue a call to any of the functions documented here—Fboolpr(), Fboolpr32(), Fvboolpr(), or Fvboolpr32()—while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

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

### [FBADVIEW]

"cannot find or get view"

*viewname* was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

### [FVFOPEN]

"cannot find or open viewfile"

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 viewfile.

### [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 Oracle Tuxedo System Workstation DLL for Windows.

## See Also

[Introduction to FML Functions](#), [Fboolco](#), [Fboolco32](#), [Fvboolco](#), [Fvboolco32\(3fml\)](#)

# Fchg, Fchg32(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 or FLD\_MBSTRING.

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.

For values of type `FLD_PTR`, `Fchg32()` stores the pointer value. The buffer pointed to by a `FLD_PTR` field must be allocated using the `tpalloc()` call. For values of type `FLD_FML32`, `Fchg32()` stores the entire `FLD_FML32` field value, except the index. For values of type `FLD_VIEW32`, `Fchg()` stores a pointer to a structure of type `FVIEWFLD`, which contains *vflags* (a flags field, currently unused and set to 0), *vname* (a character array containing the viewname), and *data* (a pointer to the view data stored as a C structure). The application provides the *vname* and *data* to `Fchg32()`.

For values of type `FLD_MBSTRING`, the value is the *packed* output argument of the `Fmbpack32()` function, and the *len* argument is the length of the value in the *size* output argument of `Fmbpack32()`.

For values of type `FLD_CARRAY`, the *len* argument is the length of the value. For all types other than `FLD_CARRAY` or `FLD_MBSTRING`, the length of the object referenced by *value* is inferred from its type (for example, a value of type `FLD_FLOAT` is of length `sizeof(float)`), and the contents of *len* are ignored.

`Fchg32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fchg()` or `Fchg32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fchg()` fails and sets `Error` 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

[Introduction to FML Functions](#), [CFchg](#), [CFchg32 \(3fml\)](#), [Fadd](#), [Fadd32 \(3fml\)](#), [Fcmp](#), [Fcmp32 \(3fml\)](#), [Fdel](#), [Fdel32 \(3fml\)](#)

# Fchgs, Fchgs32(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(), 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.

A thread in a multithreaded application may issue a call to Fchgs() or Fchgs32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fchgs() fails and sets `Error` 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

[Introduction to FML Functions](#), [CFchg](#), [CFchg32\(3fml\)](#), [Fchg](#), [Fchg32\(3fml\)](#)

# Fchksum, Fchksum32(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.

For values of type FLD\_PTR, the name of the pointer field (rather than the pointer or the data referenced by the pointer) is included in the checksum calculation.

Fchksum32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fchksum() or Fchksum32() while running in any context state, including TPINVALIDCONTEXT.

## 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

[Introduction to FML Functions](#), [Fread](#), [Fread32\(3fml\)](#), [Fwrite](#), [Fwrite32\(3fml\)](#)

# Fcmp, Fcmp32(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.

For values of type FLD\_PTR, two pointer fields are considered equal if the pointer values (addresses) are equal. For values of type FLD\_FML32, two fields are considered equal if all field occurrences and values are equal. For values of type FLD\_VIEW32, two fields are considered equal if the viewnames are the same, and if all structure member occurrences and values are equal.

Fcmp32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fcmp() or Fcmp32() while running in any context state, including TPINVALIDCONTEXT.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `Fcmp()` fails and sets `Error` 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

[Introduction to FML Functions](#), `Fadd`, `Fadd32(3fml)`, `Fchg`, `Fchg32(3fml)`

# Fconcat, Fconcat32(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.

A thread in a multithreaded application may issue a call to Fconcat() or Fconcat32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fconcat() fails and sets `Error` 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

[Introduction to FML Functions](#), [Fjoin](#), [Fjoin32\(3fml\)](#), [Fupdate](#), [Fupdate32\(3fml\)](#)

# Fcpy, Fcpy32(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.

For values of type FLD\_PTR, Fcpy32() copies the buffer pointer. The application programmer must manage the reallocation and freeing of buffers when the associated pointer is copied.

Fcpy32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fcpy() or Fcpy32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fcpy() fails and sets *Error* 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

[Introduction to FML Functions](#), `Fmove`, `Fmove32 (3fml)`

# Fdel, Fdel32(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() to set the field occurrence value to a NULL value.

For values of type FLD\_PTR, Fdel32() deletes the FLD\_PTR field occurrence without changing the referenced buffer or freeing the pointer. The data buffer is treated as an opaque pointer.

Fdel32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fdel() or Fdel32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fdel() fails and sets `Error` 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

[Introduction to FML Functions](#), [Fadd](#), [Fadd32\(3fml\)](#), [Fchg](#), [Fchg32\(3fml\)](#), [Fdelall](#), [Fdelall32\(3fml\)](#), [Fdelete](#), [Fdelete32\(3fml\)](#)

# Fdelall, Fdelall32(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.

For values of type FLD\_PTR, Fdelall32() deletes the FLD\_PTR field occurrence without changing the referenced buffer or freeing the pointer. The data buffer is treated as an opaque pointer.

Fdelall32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fdelall() or Fdelall32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fdelall() fails and sets `Error` 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

[Introduction to FML Functions](#), [Fdel](#), [Fdel32\(3fml\)](#), [Fdelete](#), [Fdelete32\(3fml\)](#)

# Fdelete, Fdelete32(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 rearranged by Fdelete() (they are sorted, if not already, in numeric order).

For values of type FLD\_PTR, Fdelete32() deletes the FLD\_PTR field occurrence without changing the referenced buffer or freeing the pointer. The data buffer is treated as an opaque pointer.

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

Fdelete32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fdelete() or Fdelete32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fdelete() fails and sets *Error* 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

[Introduction to FML Functions](#), [Fdel](#), [Fdel32\(3fml\)](#), [Fdelall](#), [Fdelall32\(3fml\)](#)

# Fextread, Fextread32(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()`). 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()`, 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 after each completed input buffer.

For values of type `FLD_FML32` and `FLD_VIEW32`, `Fextread32()` generates nested `FML32` buffers and `VIEW32` fields, respectively. This function ignores the `FLD_PTR` field type. No error is returned if a value of type `FLD_PTR` is supplied to the function.

`Fextread32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fextread()` or `Fextread32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fextread()` fails and sets `Error` 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 `Unixerr` (values defined in `Unix.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()` failed.

**[FEINVAL]**

"invalid parameter"

The value of *iop* is NULL.

## Portability

This function is not supported using the Oracle Tuxedo System Workstation DLL for Windows.

## See Also

[Introduction to FML Functions](#), [Fprint](#), [Fprint32\(3fml\)](#)

# Ffind, Ffind32(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, because proper alignment within a buffer is not guaranteed, the locations in which the values of types FLD\_LONG, FLD\_FLOAT, FLD\_DOUBLE, FLD\_PTR, FLD\_FML32, and FLD\_VIEW32 are stored prevents these values from being used directly as their stored type. Such values must be copied first to a suitably aligned memory location. Accessing such fields through the conversion function Cffind() does guarantee the proper alignment of the found converted value. Buffer modification should be done only by the Fadd() or Fchg() function. The values returned by Ffind() and Ffindlast() are valid only so long as the buffer remains unmodified.

Ffind32() does not check for occurrences of the specified field in embedded buffers as provided by the FLD\_FML32 and FLD\_VIEW32 field types.

For a specified *fieldid* of type FLD\_MBSTRING, the value returned by Ffind32() may be analyzed using the Fmbunpack(32) function.

Ffind32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Ffind() or Ffind32() while running in any context state, including TPINVALIDCONTEXT.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `Ffind()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Ffindlast](#), [Ffindlast32\(3fml\)](#), [Ffindocc](#), [Ffindocc32\(3fml\)](#), [Ffinds](#), [Ffinds32\(3fml\)](#)

# Ffindlast, Ffindlast32(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()`. 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, because proper alignment within a buffer is not guaranteed, the locations in which the values of types `FLD_LONG`, `FLD_FLOAT`, `FLD_DOUBLE`, `FLD_PTR`, `FLD_FML32`, and `FLD_VIEW32` are stored prevents these values from being used directly as their stored type. Such values must be copied first to a suitably aligned memory location. Accessing such fields through the conversion function `Cffind()` does guarantee the proper alignment of the found converted value. Buffer modification should be done only by the `Fadd()` or `Fchg()` function. The values returned by `Ffind()` and `Ffindlast()` are valid only so long as the buffer remains unmodified.

`Ffindlast32()` does not check for occurrences of the specified field in embedded buffers as provided by the `FLD_FML32` and `FLD_VIEW32` field types.

For a specified *fieldid* of type `FLD_MBSTRING`, the value returned by `Ffindlast32()` may be analyzed using the `Fmbunpack(32)` function.

Ffindlast32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Ffindlast() or Ffindlast32() while running in any context state, including TPINVALIDCONTEXT.

## 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

[Introduction to FML Functions](#), CFfind, CFfind32(3fml), Fadd, Fadd32(3fml), Fchg, Fchg32(3fml), Ffind, Ffind32(3fml), Ffindocc, Ffindocc32(3fml), Ffinds, Ffinds32(3fml)

# Ffindocc, Ffindocc32(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` or `FLD_MBSTRING`. 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()` pattern matching for `FLD_STRING` fields; it uses `strcmp()` (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.

For values of type `FLD_PTR`, `Ffindocc32()` finds the occurrence of a field that matches a specified pointer value. For values of type `FLD_FML32`, two fields are considered equal if all field occurrences and values are equal. For values of type `FLD_VIEW32`, two fields are considered equal if the viewnames are the same, and if all structure member occurrences and values are equal.

For values of type `FLD_MBSTRING`, the value is the *packed* output argument of the `Fmbpack32()` function, and the *len* argument is the length of the value in the *size* output argument of `Fmbpack32()`.

`Ffindocc32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Ffindocc()` or `Ffindocc32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Ffindocc()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Ffind](#), [Ffind32\(3fml\)](#), [Ffindlast](#), [Ffindlast32\(3fml\)](#), [Ffinds](#), [Ffinds32\(3fml\)](#), [regcmp\(3\)](#) in a UNIX system reference manual

# Ffinds, Ffinds32(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()`, 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()`.

`Ffinds32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Ffinds()` or `Ffinds32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Ffinds()` fails and sets `Error` 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()` failed while converting carray (or mbstring) to string.

## See Also

[Introduction to FML Functions](#), [CFfind](#), [CFfind32\(3fml\)](#), [Ffind](#), [Ffind32\(3fml\)](#)

# Ffloatev, Ffloatev32, Fvfloatev, Fvfloatev32(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(), *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 functionality 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.

A thread in a multithreaded application may issue a call to any of the functions documented here—Ffloatev(), Ffloatev32(), Fvfloatev(), or Fvfloatev32()—while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

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

## Errors

Under the following conditions, `Ffloatev()` fails and sets `Error` 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()` 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 viewfile"

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 viewfile.

[`FMALLOC`]

"malloc failed"

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

Ffloatev, Ffloatev32, Fvfloatev, Fvfloatev32(3fml)

## See Also

[Introduction to FML Functions](#), [Fboolco](#), [Fboolco32](#), [Fvboolco](#), [Fvboolco32\(3fml\)](#),  
[Fboolev](#), [Fboolev32](#), [Fvboolev](#), [Fvboolev32\(3fml\)](#)

# Ffprint, Ffprint32(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()`, 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()` 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.

For values of type `FLD_PTR`, `Ffprint32()` prints the field name or field identifier and the pointer value in hexadecimal. Although this function prints pointer information, the `Fextread32()` function ignores the `FLD_PTR` field type.

For values of type `FLD_FML32`, `Ffprint32()` recursively prints the FML32 buffer, with leading tabs added for each level of nesting. For values of type `FLD_VIEW32`, `Ffprint32()` prints the VIEW32 field name and structure member name/value pairs.

`Ffprint32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Ffprint()` or `Ffprint32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Ffprintf()` 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()` failed.

## Portability

This function is not supported using the Oracle Tuxedo System Workstation DLL for Windows.

## See Also

[Introduction to FML Functions](#), [Fprintf](#), [Fprintf32\(3fml\)](#)

# Fsprintf, Fsprintf32(3fml)

## Name

`Fsprintf()`, `Fsprintf32()` - print fielded buffer to a buffer

## Synopsis

```
#include <stdio.h>

#include "fml.h"

int

Fsprintf(FBFR *fbfr, char *dest)

#include "fml32.h"

int

Fsprintf32(FBFR32 *fbfr, char *dest)
```

## Description

This function requires Oracle Tuxedo 12c Release 2 (12.1.3) Rolling Patch 048 or later.

`Fsprint()` is similar to `Ffprintf()`, except the text is printed to a buffer. `fbfr` is a pointer to a fielded buffer. `dest` is a pointer of char type buffer. Make sure this char buffer is large enough to contain the resulting string.

For each field in the buffer, the output prints the field name and field value separated by a tab. `Fname()` 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.

For values of type `FLD_PTR`, `Fsprint32()` prints the field name or field identifier and the pointer value in hexadecimal. Although this function prints pointer information, the `Fextread32()` function ignores the `FLD_PTR` field type.

For values of type `FLD_FML32`, `Fsprint32()` recursively prints the FML32 buffer, with leading tabs added for each level of nesting. For values of type `FLD_VIEW32`, `Fsprint32()` prints the VIEW32 field name and structure member name/value pairs.

`Fsprint32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fsprint()` or `Fsprint32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fsprint()` fails and sets `Error` 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()` failed.

## Portability

This function is not supported using the Oracle Tuxedo System Workstation DLL for Windows.

## See Also

[Introduction to FML Functions](#), [Ffprint](#), [Ffprint32\(3fml\)](#)

# Ffree, Ffree32(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, it is made non-fielded, and then freed.

`Ffree32()` does not free the memory area referenced by a pointer in a `FLD_PTR` field.

`Ffree()` is recommended as opposed to `free()` (in UNIX system reference manuals), because `Ffree()` invalidates a fielded buffer whereas `free()` does not. It is important to invalidate fielded buffers because `malloc()` (in UNIX system reference manuals) reuses memory that has been freed without clearing it. Thus, if `free()` 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.

A thread in a multithreaded application may issue a call to `Ffree()` or `Ffree32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Ffree()` fails and sets `Error` 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

[Introduction to FML Functions](#), `Falloc`, `Falloc32(3fml)`, `Frealloc`, `Frealloc32(3fml)`

`free(3)`, `malloc(3)` in a UNIX system reference manual

# Fget, Fget32(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 *loc, FLDLEN
    *maxlen)
#include "fml32.h"
int
Fget32(FBFR32 *fbfr, FLDID32 fieldid, FLDOCC32 oc, char *loc,
    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.

For a specified *fieldid* of type FLD\_MBSTRING, the value returned by Fget32() may be analyzed using the Fmbunpack(32) function.

Fget32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fget()` or `Fget32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

When `Fget32()` is used with the `FLD_VIEW32` field type, a pointer to the `FVIEWFLD` structure is returned. This function returns -1 on error and sets `Error` to indicate the error condition.

## Errors

Under the following conditions, `Fget()` fails and sets `Error` 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

[Introduction to FML Functions](#), [CFget](#), [CFget32\(3fml\)](#), [Fgetalloc](#), [Fgetalloc32\(3fml\)](#), [Fgetlast](#), [Fgetlast32\(3fml\)](#), [Fgets](#), [Fgets32\(3fml\)](#), [Fgetsa](#), [Fgetsa32\(3fml\)](#)

# Fgetalloc, Fgetalloc32(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(), Fgetalloc() finds and makes a copy of a buffer field, but it acquires space for the field via a call to malloc() (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 reinsertion 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() space acquired by Fgetalloc(). The buffer will be aligned properly for any field type.

For a specified fieldid of type FLD\_MBSTRING, the value returned by Fgetalloc32() may be analyzed using the Fmbunpack(32) function.

Fgetalloc32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fgetalloc() or Fgetalloc32() while running in any context state, including TPINVALIDCONTEXT.

## 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. When Fgetalloc32() is used with the FLD\_VIEW32 field type, a pointer to the FVIEWFLD structure is returned. This function returns NULL on error and sets Ferror to indicate the error condition.

## Errors

Under the following conditions, `Fgetalloc()` fails and sets `Error` 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()` failed.

## See Also

[Introduction to FML Functions](#), [CFget](#), [CFget32\(3fml\)](#), [Fget](#), [Fget32\(3fml\)](#), [Fgetlast](#), [Fgetlast32\(3fml\)](#), [Fgets](#), [Fgets32\(3fml\)](#), [Fgetsa](#), [Fgetsa32\(3fml\)](#)

[free\(3\)](#), [malloc\(3\)](#) in a UNIX system reference manual

# Fgetlast, Fgetlast32(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 same type as the stored type of the field being retrieved.

For a specified *fieldid* of type FLD\_MBSTRING, the value returned by Fgetlast32() may be analyzed using the Fmbunpack(32) function.

Fgetlast32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fgetlast() or Fgetlast32() while running in any context state, including TPINVALIDCONTEXT.

## 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

[Introduction to FML Functions](#), [Fget](#), [Fget32\(3fml\)](#), [Fgetalloc](#), [Fgetalloc32\(3fml\)](#), [Fgets](#), [Fgets32\(3fml\)](#), [Fgetsa](#), [Fgetsa32\(3fml\)](#)

# Fgets, Fgets32(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()` with an assumed *utype* of `FLD_STRING`, and a *ulen* of 0.

`Fgets32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fgets()` or `Fgets32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fgets()` fails and sets `Error` 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()` failed.

## See Also

[Introduction to FML Functions](#), [CFget](#), [CFget32\(3fml\)](#), [Fget](#), [Fget32\(3fml\)](#), [Fgetalloc](#), [Fgetalloc32\(3fml\)](#), [Fgetlast](#), [Fgetlast32\(3fml\)](#), [Fgetsa](#), [Fgetsa32\(3fml\)](#)

# Fgetsa, Fgetsa32(3fml)

## Name

`Fgetsa()`, `Fgetsa32()` - use `malloc()` to allocate 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()`. *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()` (in UNIX system programmer's reference manuals) to allocate space for the retrieved 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()` (in UNIX system reference manuals) the space `malloc()`'d.

`Fgetsa32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fgetsa()` or `Fgetsa32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

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

## Errors

Under the following conditions, `Fgetsa()` fails and sets `Error` 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()` failed.

## See Also

[Introduction to FML Functions](#), [CFget](#), [CFget32\(3fml\)](#), [Fget](#), [Fget32\(3fml\)](#), [Fgetlast](#), [Fgetlast32\(3fml\)](#), [Fgets](#), [Fgets32\(3fml\)](#)

[free\(3\)](#), [malloc\(3\)](#) in a UNIX system reference manual

# Fidnm\_unload, Fidnm\_unload32(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()` for field identifier to field name mapping tables.

`Fidnm_unload32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fidnm_unload()` or `Fidnm_unload32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## See Also

[Introduction to FML Functions](#), [Fname](#), [Fname32\(3fml\)](#), [Fnmid\\_unload](#), [Fnmid\\_unload32\(3fml\)](#)

# Fidxused, Fidxused32(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.

A thread in a multithreaded application may issue a call to `Fidxused()` or `Fidxused32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `Fidxused()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Findex](#), [Findex32\(3fml\)](#), [Frstrindex](#),  
[Frstrindex32\(3fml\)](#), [Funused](#), [Funused32\(3fml\)](#), [Fused](#), [Fused32\(3fml\)](#)

# Fielded, Fielded32(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.

A thread in a multithreaded application may issue a call to `Fielded()` or `Fielded32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

`Fielded()` returns true if the buffer is fielded. It returns false if the buffer is not fielded and does not set `Error` in this case.

## See Also

[Introduction to FML Functions](#), [Finit](#), [Finit32\(3fml\)](#), [Fneeded](#), [Fneeded32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#)

# Findex, Findex32(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 `FSTDXTINTVL` (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.

A thread in a multithreaded application may issue a call to `Findex()` or `Findex32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Findex()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fidxused](#), [Fidxused32\(3fml\)](#), [Frstrindex](#), [Frstrindex32\(3fml\)](#), [Funindex](#), [Funindex32\(3fml\)](#)

# Finit, Finit32(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 reinitialize a previously used buffer.

`Finit32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Finit()` or `Finit32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Finit()` fails and sets `Error` 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 reinitialize a buffer to have no fields is: `Finit(fbfr, (FLDLLEN) Fsizeof(fbfr)) ;`

## See Also

[Introduction to FML Functions](#), [Falloc](#), [Falloc32\(3fml\)](#), [Fneeded](#), [Fneeded32\(3fml\)](#), [Frealloc](#), [Frealloc32\(3fml\)](#)

# Fjoin, Fjoin32(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. If joining buffers results in the removal of a `FLD_PTR` field, the memory area referenced by the pointer is not modified or freed.

This function may fail due to lack of space if the new values are larger than the old; in this case, the destination buffer is modified. If this happens, however, the destination buffer may be reallocated using `Frealloc()` and repeated calls to the `Fjoin()` function. Even if the destination buffer has been partially updated, the correct results are obtained by repeating the `Fjoin()` function.

`Fjoin32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fjoin()` or `Fjoin32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fjoin()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fconcat](#), [Fconcat32\(3fml\)](#), [Fojoin](#), [Fojoin32\(3fml\)](#), [Fproj](#), [Fproj32\(3fml\)](#), [Fprojcpy](#), [Fprojcpy32\(3fml\)](#), [Frealloc](#), [Frealloc32\(3fml\)](#)

# Fldid, Fldid32(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 run-time 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()` function.

`Fldid32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fldid()` or `Fldid32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

This function returns `BADFLDID` on error and sets `Error` to indicate the error condition.

## Errors

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

[FEADNAME]

"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()` failed.

## See Also

[Introduction to FML Functions](#), [Fldno](#), [Fldno32\(3fml\)](#), [Fname](#), [Fname32\(3fml\)](#),  
[Fnmid\\_unload](#), [Fnmid\\_unload32\(3fml\)](#)

`malloc(3)` in a UNIX system reference manual

# Fldno, Fldno32(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.

A thread in a multithreaded application may issue a call to `Fldno()` or `Fldno32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## See Also

[Introduction to FML Functions](#), [Fldid](#), [Fldid32\(3fml\)](#), [Fldtype](#), [Fldtype32\(3fml\)](#)

# Fldtype, Fldtype32(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.

A thread in a multithreaded application may issue a call to `Fldtype()` or `Fldtype32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

This function returns the field type.

## See Also

[Introduction to FML Functions](#), [Fldid](#), [Fldid32\(3fml\)](#), [Fldno](#), [Fldno32\(3fml\)](#)

# Flen, Flen32(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.

For values of type `FLD_PTR`, `Flen32()` returns a fixed length for a pointer field based on `sizeof(char*)`. For values of type `FLD_FML32`, `Flen32()` returns the value of `Fused32()` for the length of the nested buffer. For values of type `FLD_VIEW32`, `Flen32()` returns the length of the view data plus the length of the viewname.

For values of type `FLD_MBSTRING`, `Flen32()` returns the length of the packed output created by the `Fmbpack32()` function.

`Flen32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Flen()` or `Flen32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

On success, `Flen()` returns the field length.

This function returns -1 on error and sets `Error` 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

[Introduction to FML Functions](#), `Fnum`, `Fnum32(3fml)`, `Fpres`, `Fpres32(3fml)`

# Fmbpack32(3fml)

## Name

Fmbpack32() - prepare encoding name and multibyte data information

## Synopsis

```
#include "fml32.h"

int
Fmbpack32 (char *enc, void *ind, FLDLEN32 indlen, void
*packed, FLDLEN32 *size, long flags)
```

## Description

Fmbpack32() prepares the encoding name and multibyte data information for an FLD\_MBSTRING field input to an FML32 typed buffer. Fmbpack32() is used before the FLD\_MBSTRING field is added to an FML32 buffer via FML32 APIs.

*enc*, if not NULL, is a NULL-terminated ASCII string containing the code-set encoding name for the *ind* code-set multibyte data. If *enc* is NULL and the *flags* argument is 0, the encoding name that is to be included in the *packed* output is obtained from the process TMBENC environment variable. If the *flags* argument is FBUFENC, *enc* is ignored.

*ind* is the code-set multibyte data.

*indlen* is the number of bytes in *ind*.

*packed* is a pointer to an output for Fmbpack32(). It is used as an input value to FML32 APIs that include FLD\_MBSTRING fields in an FML32 buffer. The packed area must be aligned on a FLDLEN32(TM32U) boundary.

*size*, on input, is the size of memory pointed to by *packed*. If the size is not large enough to handle the result of Fmbpack32(), FNOSPACE is returned and *size* is reset to the number of bytes that *packed* should be. After successful execution of Fmbpack32(), *size* is reset to the actual number of bytes used.

*flags* is 0 or FBUFENC. If *flags* is set to FBUFENC, Fmbpack32() ignores the *enc* argument and includes FBUFENC with the input data to *packed*. Without the inclusion of an encoding name, the *packed* output forces the FML32 API processing the FLD\_MBSTRING field to get the encoding name from the FML32 buffer. Therefore, usage of FBUFENC also requires the application developer to use tpsetmbenc() to set the encoding name for the FML32 buffer.

## Return Values

On success, `Fmbpack32()` returns a positive value. On error, `Fmbpack32()` returns -1 and sets `Error32` to indicate the error condition.

## Errors

Under the following conditions, `Fmbpack32()` fails and sets `Error32` to:

[FEINVAL]

*ind*, *packed*, or *size* is NULL. *enc* or *indlen* is not valid.

[FNOSPACE]

The size of *packed* is not sufficient to handle the result of `Fmbpack32()`.

## See Also

[Fmbunpack32\(3fml\)](#), [tpconvfmb32\(3fml\)](#), [tpsetmbenc\(3c\)](#), [tuxgetmbenc\(3c\)](#),  
[tuxsetmbenc\(3c\)](#)

# Fmbunpack32(3fml)

## Name

Fmbunpack32() - extract encoding name and multibyte data information

## Synopsis

```
#include "fml32.h"

int
Fmbunpack32 (void *packed, FLDLEN32 ilen, char *enc, void
*outd, FLDLEN32 *olen, long flags)
```

## Description

Fmbunpack32() extracts the encoding name and multibyte data information from an FLD\_MBSTRING field in an FML32 typed buffer. Fmbunpack32() is used after the FLD\_MBSTRING field is extracted from an FML32 buffer via FML32 APIs (Ffind32(), Fget32(), ...).

*packed* is a pointer to the FLD\_MBSTRING field data output from an FML32 API.

*ilen* is the number of bytes for *packed*.

*enc* is a NULL-terminated ASCII string in *packed* containing the encoding name when the code-set encoding name for the FLD\_MBSTRING field is part of the *packed* information. If the FLD\_MBSTRING field was created by Fmbpack32() with flag FBUFENC, *enc* is set to NULL. For the latter case, the application developer must use tpgetmbenc() on the FML32 buffer to get the encoding name for the FLD\_MBSTRING field.

*outd* contains the multibyte data extracted from *packed* upon successful completion of Fmbunpack32().

*olen*, on input, is the size of memory pointed to by *outd*. If the size is not large enough to handle the result of Fmbunpack32(), FNOSPACE is returned and *olen* is reset to the number of bytes that *outd* should be. After successful execution of Fmbunpack32(), *olen* is reset to the actual number of bytes used.

*flags* is currently not used and should be set to 0.

## Return Values

On success, Fmbunpack32() returns a positive value. On error, Fmbunpack32() returns -1 and sets Ferror32 to indicate the error condition.

## Errors

Under the following conditions, `Fmbunpack32()` fails and sets `Error32` to:

[FEINVAL]

*outd*, *olen*, or *packed* is NULL. *packed* or *ilen* is not valid.

[FNOSPACE]

The size of *outd* is not sufficient to handle the result of `Fmbunpack32()`.

## See Also

[Fmbpack32\(3fml\)](#), [tpconvfmb32\(3fml\)](#), [tpgetmbenc\(3c\)](#), [tuxgetmbenc\(3c\)](#),  
[tuxsetmbenc\(3c\)](#)

# Fmkfldid, Fmkfldid32(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](#), [Fldtype32\(3fml\)](#)). *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.

A thread in a multithreaded application may issue a call to `Fmkfldid()` or `Fmkfldid32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

This function returns `BADFLDID` on error and sets `Error` to indicate the error condition.

## Errors

Under the following conditions, `Fmkfldid()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fldtype](#), [Fldtype32\(3fml\)](#)

# Fmove, Fmove32(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 `Fcopy()` is that `Fcopy()` 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()` bytes of data from the source fielded buffer to the target buffer. The destination buffer must be aligned on a short boundary.

For values of type `FLD_PTR`, `Fmove32()` transfers the buffer pointer. The application programmer must manage the reallocation and freeing of buffers when the associated pointer is moved.

`Fmove32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fmove()` or `Fmove32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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

[Introduction to FML Functions](#), [Fcpy](#), [Fcpy32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#)

# Fname, Fname32(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 run-time 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()`.

`Fname32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fname()` or `Fname32()` while running in any context state, including `TPINVALIDCONTEXT`.

The returned pointer is maintained in an internal FML table. If a multi-threaded application decides to de-reference the returned pointer, thread-safety should be ensured by the application code itself. Specifically, the pointer returned by `Fname` should not be de-referenced, once `Fnmid_unload`/`Fidnm_unload` is issued.

## Return Values

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

## Errors

Under the following conditions, `Fname()` fails and sets `Error` to:

Fname, Fname32(3fml)

[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()` failed.

## See Also

[Introduction to FML Functions](#), [Ffprint](#), [Ffprint32\(3fml\)](#), [Fidnm\\_unload](#),  
[Fidnm\\_unload32\(3fml\)](#), [Fldid](#), [Fldid32\(3fml\)](#), [Fprint](#), [Fprint32\(3fml\)](#)

# Fneeded, Fneeded32(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()` is used to determine the space that must be allocated for a fielded buffer. The *F* argument is the number of fields, and the *V* argument is the space for all field values, in bytes.

`Fneeded32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fneeded()` or `Fneeded32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fneeded()` fails and sets `Error` 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).

Fneeded, Fneeded32(3fml)

## See Also

[Introduction to FML Functions](#), [Falloc](#), [Falloc32\(3fml\)](#), [Fielded](#), [Fielded32\(3fml\)](#), [Finit](#), [Finit32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#), [Funused](#), [Funused32\(3fml\)](#), [Fused](#), [Fused32\(3fml\)](#)

# Fnext, Fnext32(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.

When the type of the field to be retrieved is `FLD_VIEW32`, the *value* parameter points to a `FVIEWFLD` structure. The `Fnext()` function populates the *vname* and *data* fields in the structure.

Fnext32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fnext() or Fnext32() while running in any context state, including TPINVALIDCONTEXT.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, Fnext() fails and sets `Error` 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

[Introduction to FML Functions](#), [Fget](#), [Fget32\(3fml\)](#), [Fnum](#), [Fnum32\(3fml\)](#)

# Fnmid\_unload, Fnmid\_unload32(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()`, the user may unload the files by a call to the `Fnmid_unload()` function.

`Fnmid_unload32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fnmid_unload()` or `Fnmid_unload32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## See Also

[Introduction to FML Functions](#), [Fidnm\\_unload](#), [Fidnm\\_unload32\(3fml\)](#), [Fldid](#), [Fldid32\(3fml\)](#)

# Fnum, Fnum32(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(FBFR32 *fbfr)
```

## Description

Fnum() returns the number of fields contained in the specified buffer. *fbfr* is a pointer to a fielded buffer. The FLD\_FML32 and FLD\_VIEW32 fields are each counted as a single field, regardless of the number of fields they contain.

Fnum32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fnum() or Fnum32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fnum() fails and sets `Error` 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

[Introduction to FML Functions](#), [Foccur](#), [Foccur32\(3fml\)](#), [Fpres](#), [Fpres32\(3fml\)](#)

# Foccur, Foccur32(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 referenced by *fbfr*. Occurrences of a field within an embedded FML32 buffer are not counted, as in the `FLD_FML32` field type.

`Foccur32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Foccur()` or `Foccur32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `Foccur()` fails and sets `Error` 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

[Introduction to FML Functions](#), `Fnum`, `Fnum32(3fml)`, `Fpres`, `Fpres32(3fml)`

# Fojoin, Fojoin32(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(), 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. If joining buffers results in the removal of a FLD\_PTR field, the memory area referenced by the pointer is not modified or freed.

As with Fjoin(), this function can fail for lack of space; it can be reissued to complete the operation after more space is allocated.

Fojoin32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fojoin() or Fojoin32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Fojoin() fails and sets *Error* 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 (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 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

[Introduction to FML Functions](#), [Fconcat](#), [Fconcat32\(3fml\)](#), [Fjoin](#), [Fjoin32\(3fml\)](#), [Fproj](#), [Fproj32\(3fml\)](#)

# Fpres, Fpres32(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 whether a given occurrence (*oc*) of a specified field (*fieldid*) exists in the buffer referenced by *fbfr*. Fpres32() does not check for occurrences of the specified field within an embedded buffer, as in the FLD\_FML32 field type.

Fpres32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fpres() or Fpres32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

Fpres() returns true if the specified occurrence exists and false otherwise.

## See Also

[Introduction to FML Functions](#), [Ffind](#), [Ffind32\(3fml\)](#), [Fnum](#), [Fnum32\(3fml\)](#), [Foccur](#), [Foccur32\(3fml\)](#)

# Fprint, Fprint32(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()` 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.

For values of type `FLD_PTR`, `Fprint32()` prints the field name or field identifier and the pointer value in hexadecimal. Although this function prints pointer information, the `Fextread32()` function ignores the `FLD_PTR` field type.

For values of type `FLD_FML32`, `Fprint32()` recursively prints the FML32 buffer, with leading tabs added for each level of nesting. For values of type `FLD_VIEW32`, `Fprint32()` prints the VIEW32 field name and structure member name/value pairs.

`Fprint32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fprint()` or `Fprint32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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 `Error` 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()` failed.

## See Also

[Introduction to FML Functions](#), [Fextread](#), [Fextread32\(3fml\)](#), [Ffprint](#),  
[Ffprint32\(3fml\)](#), [Fname](#), [Fname32\(3fml\)](#)

# Fproj, Fproj32(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 rearranged. (If they are not already in numeric order, they are sorted.) If updating buffers results in the removal of a `FLD_PTR` field, the memory area referenced by the pointer is not modified or freed.

`Fproj32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fproj()` or `Fproj32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fproj()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fjoin](#), [Fjoin32\(3fml\)](#), [Fojoin](#), [Fojoin32\(3fml\)](#), [Fprojcpy](#), [Fprojcpy32\(3fml\)](#)

# Fprojcpy, Fprojcpy32(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()` 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 rearranged. (If they are not already in numeric order, they are sorted.) If updating buffers results in the removal of a `FLD_PTR` field, the memory area referenced by the pointer is not modified or freed.

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

`Fprojcpy32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fprojcpy()` or `Fprojcpy32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fprojcpy()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fjoin](#), [Fjoin32\(3fml\)](#), [Fojoin](#), [Fojoin32\(3fml\)](#), [Fproj](#), [Fproj32\(3fml\)](#)

# Fread, Fread32(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()` ignores the `FLD_PTR` field type. No error is returned if a value of type `FLD_PTR` is supplied to the function.

`Fread32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fread()` or `Fread32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fread()` fails and sets `Error` 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 Oracle Tuxedo System Workstation DLL for Windows.

## See Also

[Introduction to FML Functions](#), `Findex`, `Findex32(3fml)`, `Fwrite`, `Fwrite32(3fml)`

`stdio(3S)` in a UNIX system reference manual

# Frealloc, Frealloc32(3fml)

## Name

`Frealloc()`, `Frealloc32()` - reallocate 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 reallocate 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.

A thread in a multithreaded application may issue a call to `Frealloc()` or `Frealloc32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

On success, `Frealloc()` returns a pointer to the reallocated FBFR.

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

## Errors

Under the following conditions, `Frealloc()` fails and sets `Error` 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()` failed.

## See Also

[Introduction to FML Functions](#), [Falloc](#), [Falloc32\(3fml\)](#), [Ffree](#), [Ffree32\(3fml\)](#)

# Frstrindex, Frstrindex32(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()` 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()` function.

`Frstrindex32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Frstrindex()` or `Frstrindex32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Frstrindex()` fails and sets `Error` 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()`, 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()` and so has no idea of what the value of the *numidx* argument to `Frstrindex()` should be.
2. The index itself is not available because it was not sent.

The solution is to call `Findex()` 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()` on the receiving side.

## See Also

[Introduction to FML Functions](#), [Findex](#), [Findex32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#),  
[Funindex](#), [Funindex32\(3fml\)](#)

# Fsizeof, Fsizeof32(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.

A thread in a multithreaded application may issue a call to `Fsizeof()` or `Fsizeof32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fsizeof()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fidxused](#), [Fidxused32\(3fml\)](#), [Fused](#), [Fused32\(3fml\)](#), [Funused](#), [Funused32\(3fml\)](#)

# Fstrerror, Fstrerror32(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.

A thread in a multithreaded application may issue a call to `Fstrerror()` or `Fstrerror32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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

[Introduction to FML Functions](#), [tpstrerror\(3c\)](#), [userlog\(3c\)](#), [F\\_error](#), [F\\_error32\(3fml\)](#)

# Ftypecvt, Ftypecvt32(3fml)

## Name

Ftypecvt(), Ftypecvt32() - convert from one field type to another

## Synopsis

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

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

#include "fml32.h"

char *
Ftypecvt32(FLDLLEN32 *tolen, int totype, char *fromval, int fromtype,
           FLDLEN32 fromlen)
```

## Description

Ftypecvt() 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*. Ftypecvt() returns a pointer to the converted value, and sets *\*tolen* to the converted length, upon success. Upon failure, Ftypecvt() returns NULL.

Ftypecvt32() fails if any of the following field types is used: FLD\_PTR, FLD\_MBSTRING, FLD\_FML32, or FLD\_VIEW32. If one of these field types is encountered, *Error* is set to FEBADOP.

Ftypecvt32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Ftypecvt() or Ftypecvt32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, Ftypecvt() fails and sets *Error* to:

**[FMALLOC]**

"malloc failed"

Allocation of space dynamically using `malloc()` failed when converting from a carray (or mbstring) 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.

**[FTYPERR]**

"invalid field type"

An invalid field type (such as `FLD_PTR`, `FLD_FML32`, or `FLD_VIEW32`) is specified.

## See Also

[Introduction to FML Functions](#), [CFadd](#), [CFadd32\(3fml\)](#), [CFchg](#), [CFchg32\(3fml\)](#), [CFfind](#), [CFfind32\(3fml\)](#), [CFget](#), [CFget32\(3fml\)](#), [CFgetalloc](#), [CFgetalloc32\(3fml\)](#)

# Ftype, Ftype32(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.

A thread in a multithreaded application may issue a call to `Ftype()` or `Ftype32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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 `Error` to indicate the error condition.

## Errors

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

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

## See Also

[Introduction to FML Functions](#), [Fldid](#), [Fldid32\(3fml\)](#), [Fldno](#), [Fldno32\(3fml\)](#)

# Funindex, Funindex32(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.

A thread in a multithreaded application may issue a call to `Funindex()` or `Funindex32()` while running in any context state, including `TPINVALIDCONTEXT`.

## 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `Funindex()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Findex](#), [Findex32\(3fml\)](#), [Frstrindex](#),  
[Frstrindex32\(3fml\)](#), [Fsizeof](#), [Fsizeof32\(3fml\)](#), [Funused](#), [Funused32\(3fml\)](#)

# Funused, Funused32(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.

A thread in a multithreaded application may issue a call to `Funused()` or `Funused32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Funused()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fidxused](#), [Fidxused32\(3fml\)](#), [Fused](#), [Fused32\(3fml\)](#)

# Fupdate, Fupdate32(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.

For values of type FLD\_PTR, Fupdate32() stores the pointer value. The buffer pointed to by a FLD\_PTR field must be allocated using the tmalloc() call. For values of type FLD\_FML32, Fupdate32() stores the entire FLD\_FML32 field value, except the index. For values of type FLD\_VIEW32, Fupdate32() stores a pointer to a structure of type FVIEWFLD, which contains *vflags* (a flags field, currently unused and set to 0), *vname* (a character array containing the viewname), and *data* (a pointer to the view data stored as a C structure). The application provides the *vname* and *data* to Fupdate32().

Fupdate32() is used with 32-bit FML.

A thread in a multithreaded application may issue a call to Fupdate() or Fupdate32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, `Fupdate()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fjoin](#), [Fjoin32\(3fml\)](#), [Fojoin](#), [Fojoin32\(3fml\)](#), [Fproj](#), [Fproj32\(3fml\)](#), [Fprojcpy](#), [Fprojcpy32\(3fml\)](#)

# Fused, Fused32(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.

A thread in a multithreaded application may issue a call to `Fused()` or `Fused32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fused()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fidxused](#), [Fidxused32\(3fml\)](#), [Funused](#),  
[Funused32\(3fml\)](#)

# Fvall, Fvall32(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()` 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.

A thread in a multithreaded application may issue a call to `Fvall()` or `Fvall32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

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

## Errors

Under the following conditions, `Fvall()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Ffind](#), [Ffind32\(3fml\)](#), [Fvals](#), [Fvals32\(3fml\)](#)

# Fvals, Fvals32(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()` 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.

A thread in a multithreaded application may issue a call to `Fvals()` or `Fvals32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fvals()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Cffind](#), [Cffind32\(3fml\)](#), [Ffind](#), [Ffind32\(3fml\)](#), [Fvall](#), [Fvall32\(3fml\)](#)

# Fvftos, Fvftos32(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 field identifier and a member. These options are initially specified in the viewfile, but can be changed at run time using `Fvopt()`.

`Fvftos32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fvftos()` or `Fvftos32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fvftos()` fails and sets `Error` 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

[Introduction to FML Functions](#), [Fvopt](#), [Fvopt32\(3fml\)](#), [viewfile\(5\)](#)

# Fvneeded, Fvneeded32(3fml)

## Name

`Fvneeded()`, `Fvneeded32()` - computes size needed for `VIEW` buffer

## Synopsis

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

long
Fvneeded(char *subtype)

#include "fml32.h"

long
Fvneeded32(char *subtype)
```

## Description

`Fvneeded()` returns the size of the `VIEW C` structure. *subtype* is the name of the `VIEW`. You can call `Fvneeded()` to determine the size of a `VIEW` buffer to allocate. `Fvneeded32()` is used with 32-bit `VIEW`s.

## Return Values

`Fvneeded()` returns the size of the `VIEW` in number of bytes. This function returns -1 on error and sets `Error` to indicate the error condition.

## Errors

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

[`FEINVAL`]  
"invalid argument to function"  
The requested `VIEW` cannot be found in the viewfiles specified by `VIEWDIR` and `VIEWFILES` environment variables.

## See Also

[Introduction to FML Functions](#), [viewfile\(5\)](#)

# Fvnull, Fvnull32(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() 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.

A thread in a multithreaded application may issue a call to Fvnull() or Fvnull32() while running in any context state, including TPINVALIDCONTEXT.

## 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 *Error* to indicate the error condition.

## Errors

Under the following conditions, Fvnull() fails and sets *Error* 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

[Introduction to FML Functions](#), [Fvopt](#), [Fvopt32\(3fml\)](#), [viewfile\(5\)](#)

# Fvopt, Fvopt32(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 run time. *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.

A thread in a multithreaded application may issue a call to Fvopt() or Fvopt32() while running in any context state, including TPINVALIDCONTEXT.

## Return Values

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

## Errors

Under the following conditions, `Fvopt()` fails and sets `Error` 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

[Introduction to FML Functions](#), [viewfile\(5\)](#)

# Fvrefresh, Fvrefresh32(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.

A thread in a multithreaded application may issue a call to Fvrefresh() or Fvrefresh32() while running in any context state, including TPINVALIDCONTEXT.

## 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

[Introduction to FML Functions](#)

# Fvselinit, Fvselinit32(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 viewfile.

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

A thread in a multithreaded application may issue a call to `Fvselinit()` or `Fvselinit32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fvselinit()` fails and sets `Error` 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**

[Introduction to FML Functions](#), [Fvsinit](#), [Fvsinit32\(3fml\)](#), [viewfile\(5\)](#)

# Fvsinit, Fvsinit32(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 viewfile.

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

A thread in a multithreaded application may issue a call to `Fvsinit()` or `Fvsinit32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fvsinit()` fails and sets `Error` 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**

[Introduction to FML Functions](#), [Fvselinit](#), [Fvselinit32\(3fml\)](#), [viewfile\(5\)](#)

# Fvstof, Fvstof32(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()`, `Fojoin()`, `Fjoin()`, and `Fconcat()`. 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.

A thread in a multithreaded application may issue a call to `Fvstof()` or `Fvstof32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fvstof()` fails and sets `Error` 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()` failed when converting from a *carray* (or *mbstring*) or string value.

## See Also

[Introduction to FML Functions](#), [Fconcat](#), [Fconcat32\(3fml\)](#), [Fjoin](#), [Fjoin32\(3fml\)](#), [Fojoin](#), [Fojoin32\(3fml\)](#), [Fupdate](#), [Fupdate32\(3fml\)](#), [Fvftos](#), [Fvftos32\(3fml\)](#)

# Fvstot, Fvttos(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.

A thread in a multithreaded application may issue a call to `Fvstot()` or `Fvttos()` while running in any context state, including `TPINVALIDCONTEXT`.

## Default Conversion-IBM/370

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

**Table 2 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
<code>dec_t(m, n)</code>	$S9(2*m-(n+1))V9(n)COMP-3$
ASCII char	EBCDIC char
ASCII string	EBCDIC string
carray	character array
mbstring	multibyte 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 or mbstring 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 (for example, int, long, string, double, and float) using the functions described in `decimal()`.

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 interpreted 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 `Error` to indicate the error condition.

## Errors

Under the following conditions, `Fvttos()` fails and sets `Error` 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 viewfile"

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 viewfile.

[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, reflen, "test") == -1) {
        printf("Fvstot failed: %sn", Fstrerror(Ferror));
        exit(0);
    }

    /* transfer to target machine and get response */
    ...

    /* translate back */

```

```

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

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

```

## See Also

[Introduction to FML Functions](#), [Fvftos](#), [Fvftos32\(3fml\)](#), [Fvstof](#), [Fvstof32\(3fml\)](#), [viewfile\(5\)](#)

[decimal\(3\)](#) in a UNIX system reference manual

# Fwrite, Fwrite32(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.

For the `FLD_PTR` field type, only the pointer, not the data being pointed to, is written to the output stream. For the `FLD_VIEW32` field type, only the `FVIEWFLD` structure, not the data in the `VIEW32` buffer, is written to the output stream.

`Fwrite32()` is used with 32-bit FML.

A thread in a multithreaded application may issue a call to `Fwrite()` or `Fwrite32()` while running in any context state, including `TPINVALIDCONTEXT`.

## Return Values

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

## Errors

Under the following conditions, `Fwrite()` fails and sets `Error` 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 `Unixerr` (values defined in `Unix.h`) is set to the system call that returned the error.

## Portability

This function is not supported using the Oracle Tuxedo System Workstation DLL for Windows.

## See Also

[Introduction to FML Functions](#), `Findex`, `Findex32(3fml)`, `Fread`, `Fread32(3fml)`

`stdio(3S)` in a UNIX system reference manual

# tpconvfmb32(3fml)

## Name

tpconvfmb32() - convert multibyte characters from source encoding to target encoding

## Synopsis

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

extern int tperrno;

int
tpconvfmb32 (FMBR32 **bufp, FLDID32 *ids, char *target_encoding,
long flags)
```

## Description

tpconvfmb32() converts the multibyte characters in an FLD\_MBSTRING field in an FML32 typed buffer to a named target encoding. Specifically, tpconvfmb32() *compares* the source encoding name specified for the FLD\_MBSTRING field *with* the target encoding name defined in *target\_encoding*; if the encoding names are different, tpconvfmb32() converts the FLD\_MBSTRING field data to the target encoding.

tpconvfmb32() is an alternative to system-initiated encoding conversion. System-initiated encoding conversion of FLD\_MBSTRING field data is done automatically when the process TPMBACONV environment variable is set to a non-NULL value.

*bufp* is a pointer to an FML32 typed buffer. It is reallocated internally if the size associated with the pointer is insufficient to handle the converted output data of the FML32 buffer. *bufp* must be defined using the tpalloc() function, not the Falloc() function. If *bufp* contains FLD\_FML32 fields, they are checked recursively for FLD\_MBSTRING fields. If *bufp* contains FLD\_PTR fields, they are skipped.

*ids* contains a pointer to an array of field ids that are to be converted. If *ids* is NULL, all FLD\_MBSTRING fields found in *bufp* are converted to the target encoding if required. The array, if used, must be terminated with 0 (i.e., BADFLDID).

*target\_encoding* is the target code-set encoding name used to convert the FLD\_MBSTRING fields in the *bufp* message. If *target\_encoding* is NULL, tpconvfmb32() uses the encoding name defined in the process TPMBENC environment variable.

*flags* is not used by `tpconvfmb32()`. It is passed to the buffer type switch function for user-defined conversion functions.

## Return Values

On success, `tpconvfmb32()` returns 0. On error, `tpconvfmb32()` returns -1 and sets `tperrno` to indicate the error condition.

## Errors

Under the following conditions, `tpconvfmb32()` fails and sets `tperrno` to:

### [TPEPROTO]

*bufp* translates to a Tuxedo buffer that does not have a buffer type switch conversion function.

### [TPESYSTEM]

A Tuxedo system error has occurred (e.g., *bufp* does not correspond to a valid Tuxedo buffer).

### [TPEINVAL]

*target\_encoding* or *bufp* is NULL.

### [TPEOS]

An operating system error occurred. The external integer `Unixerr` (values defined in `Unix.h`) is set to the system call that returned the error.

## See Also

[Fmbpack32\(3fml\)](#), [Fmbunpack32\(3fml\)](#), [tpalloc\(3c\)](#), [tpsetmbenc\(3c\)](#),  
[tuxgetmbaconv\(3c\)](#), [tuxgetmbenc\(3c\)](#), [tuxsetmbaconv\(3c\)](#), [tuxsetmbenc\(3c\)](#)

# tpconvvmb32(3fml)

## Name

tpconvvmb32() - converts the multibyte characters in an MBSTRING field in an VIEW32 typed buffer to a named target encoding

## Synopsis

```
#include <atmi.h>
#include "fml32.h"
extern int tperrno;

int
tpconvvmb32 (char **bufp, FLDID32 *ids, char *target_encoding,
long flags)
```

## Description

Specifically, tpconvvmb32() compares the source encoding name specified for the MBSTRING field with the target encoding name defined in target\_encoding; if the encoding names are different, tpconvvmb32() converts the MBSTRING field data to the target encoding.

tpconvvmb32() is an alternative to system-initiated encoding conversion. System-initiated encoding conversion of MBSTRING field data is done automatically when the process TPMBACONV environment variable is set to a non-NULL value.

tpconvvmb32() supports the following arguments:

bufp

A pointer to an VIEW32 typed buffer. It is reallocated internally if the size associated with the pointer is insufficient to handle the converted output data of the VIEW32 buffer.

**Note:** bufp must be defined using the tpalloc() function, not Falloc().

ids

Contains a pointer to an array of field ids that are to be converted. If ids is NULL, all MBSTRING fields found in bufp are converted to the target encoding if required. The array, if used, must be terminated with 0 (i.e., BADFLDID).

target\_encoding

The target code-set encoding name used to convert the MBSTRING fields in the bufp message. If target\_encoding is NULL, tpconvvmb32() uses the encoding name defined in the process TPMBENC environment variable.

flags

Not used by `tpconvvmb32()`. It is passed to the buffer type switch function for user-defined conversion functions.

## Return Values

On success, `tpconvvmb32()` returns 0. On error, `tpconvvmb32()` returns -1 and sets `tperrno` to indicate the error condition.

## Errors

Under the following conditions, `tpconvvmb32()` fails and sets `tperrno` to:

[TPEPROTO]

`bufp` translates to a Tuxedo buffer that does not have a buffer type switch conversion function.

[TPESYSTEM]

A Tuxedo system error has occurred (e.g., `bufp` does not correspond to a valid Tuxedo buffer).

[TPEINVAL]

`target_encoding` or `bufp` is NULL.

[TPEOS]

An operating system error occurred. The external integer `Uunixerr` (values defined in `Uunix.h`) is set to the system call that returned the error.

## See Also

[Fmbpack32\(3fml\)](#), [Fmbunpack32\(3fml\)](#), [tpalloc\(3c\)](#), [tpsetmbenc\(3c\)](#),  
[tuxgetmbaconv\(3c\)](#), [tuxgetmbenc\(3c\)](#), [tuxsetmbaconv\(3c\)](#), [tuxsetmbenc\(3c\)](#)