Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'fenv.h.0p' command

$ man fenv.h.0p

fenv.h(0P) POSIX Programmer's Manual fenv.h(0P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fenv.h ? floating-point environment

SYNOPSIS

#include <fenv.h>

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The <fenv.h> header shall define the following data types through type? def:

fenv_t Represents the entire floating-point environment. The float? ing-point environment refers collectively to any floating- point status flags and control modes supported by the imple? mentation.

fexcept_t Represents the floating-point status flags collectively, in? cluding any status the implementation associates with the

flags. A floating-point status flag is a system variable whose value is set (but never cleared) when a floating-point exception is raised, which occurs as a side-effect of excep? tional floating-point arithmetic to provide auxiliary infor? mation. A floating-point control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic.

The <fenv.h> header shall define each of the following macros if and only if the implementation supports the floating-point exception by means of the floating-point functions feclearexcept(), fegetexcept? flag(), feraiseexcept(), fesetexceptflag(), and fetestexcept(). The defined macros shall expand to integer constant expressions with values that are bitwise-distinct.

FE_DIVBYZERO FE_INEXACT FE_INVALID FE_OVERFLOW FE_UNDERFLOW

If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be defined. Additional implementation-defined float? ing-point exceptions with macros beginning with FE_ and an uppercase letter may also be specified by the implementation.

The <fenv.h> header shall define the macro FE_ALL_EXCEPT as the bit? wise-inclusive OR of all floating-point exception macros defined by the implementation, if any. If no such macros are defined, then the macro FE_ALL_EXCEPT shall be defined as zero.

The <fenv.h> header shall define each of the following macros if and only if the implementation supports getting and setting the represented rounding direction by means of the fegetround() and fesetround() func? tions. The defined macros shall expand to integer constant expressions whose values are distinct non-negative values.

FE_DOWNWARD FE_TONEAREST FE_TOWARDZERO FE_UPWARD

If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be defined. Additional implementation-defined round? ing directions with macros beginning with FE_ and an uppercase letter may also be specified by the implementation.

The <fenv.h> header shall define the following macro, which represents

the default floating-point environment (that is, the one installed at program startup) and has type pointer to const-qualified fenv_t. It can be used as an argument to the functions within the <fenv.h> header that manage the floating-point environment.

FE_DFL_ENV

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  feclearexcept(int);
int  fegetenv(fenv_t *);
int  fegetexceptflag(fexcept_t *, int);
int  fegetround(void);
int  feholdexcept(fenv_t *);
int  feraiseexcept(int);
int  fesetenv(const fenv_t *);
int  fesetexceptflag(const fexcept_t *, int);
int  fesetround(int);
int  fetestexcept(int);
int  feupdateenv(const fenv_t *);
```

The FENV_ACCESS pragma provides a means to inform the implementation when an application might access the floating-point environment to test floating-point status flags or run under non-default floating-point control modes. The pragma shall occur either outside external declara? tions or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. If part of an application tests floating-point status flags, sets floating-point control modes,

or runs under non-default mode settings, but was translated with the state for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined. (When execution passes from a part of the application translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the floating-point status flags is unspecified and the floating-point control modes have their default settings.)

The following sections are informative.

APPLICATION USAGE

This header is designed to support the floating-point exception status flags and directed-rounding control modes required by the IEC 60559:1989 standard, and other similar floating-point state infor? mation. Also it is designed to facilitate code portability among all systems.

Certain application programming conventions support the intended model of use for the floating-point environment:

* A function call does not alter its caller's floating-point control modes, clear its caller's floating-point status flags, nor depend on the state of its caller's floating-point status flags unless the function is so documented.
* A function call is assumed to require default floating-point con? trol modes, unless its documentation promises otherwise.
* A function call is assumed to have the potential for raising float? ing-point exceptions, unless its documentation promises otherwise.

With these conventions, an application can safely assume default float? ing-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the application that does so explicitly.

Even though the rounding direction macros may expand to constants cor? responding to the values of FLT_ROUNDS, they are not required to do so.

For example:

```
#include <fenv.h>
void f(double x)
```

```
{
    #pragma STDC FENV_ACCESS ON
    void g(double);
    void h(double);
    /* ... */
    g(x + 1);
    h(x + 1);
    /* ... */
}
```

If the function g() might depend on status flags set as a side-effect of the first x+1, or if the second x+1 might depend on control modes set as a side-effect of the call to function g(), then the application shall contain an appropriately placed invocation as follows:

```
#pragma STDC FENV_ACCESS ON
```

RATIONALE

The fexcept_t Type

fexcept_t does not have to be an integer type. Its values must be ob? tained by a call to fegetexceptflag(), and cannot be created by logical operations from the exception macros. An implementation might simply implement fexcept_t as an int and use the representations reflected by the exception macros, but is not required to; other representations might contain extra information about the exceptions. fexcept_t might be a struct with a member for each exception (that might hold the ad? dress of the first or last floating-point instruction that caused that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an fexcept_t, and so the user cannot inspect it.

Exception and Rounding Macros

Macros corresponding to unsupported modes and rounding directions are not defined by the implementation and must not be defined by the appli? cation. An application might use #ifdef to test for this.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of POSIX.1?2017, feclearexcept(), fegetenv(), fegetexceptflag(), fegetround(), feholdexcept(), feraiseex? cept(), fetestexcept(), feupdateenv()