1 Exceptions

2 Declarations

```
#ifndef COMMON
#define COMMON _System
typedef void (COMMON _vPFN) ();
typedef _vPFN *vPFN;
typedef int (COMMON _iPFN) ();
typedef _iPFN *iPFN;
#endif

typedef struct {
   BYTE                 len1;
   BYTE                 len2;
   BYTE                 len3;
   BYTE                 len4;
   char                 texts[1];
}FAIL_EXCPTXE "FAIL_EXCPT structure"§,
*pFAIL_EXCPT;

typedef struct _link_excpt_ {
   ULONG                exception_code;
   vPFN                 hndlr;
   struct _link_excpt_ *next;
}EXCPT_INFOXE "EXCPT_INFO structure"§,
*pEXCPT_INFO;

typedef struct {
   EXCEPTIONREGISTRATIONRECORD exrr;
   pEXCPT_INFO excpt_info;
}EXCPT_STRUCT,XE "EXCPT_STRUCT structure"§
*pEXCPT_STRUCT;

#define CONTINUABLEXE "CONTINUABLE"§      0
#define NON_CONTINUABLEXE
"NON_CONTINUABLE"§            1

#define EXCEPTION_STRUCTUREXE
"EXCEPTION_STRUCTURE macro"§ EXCPT_STRUCT
excpt_struct
#define INIT_EXCEPTION_HANDLERXE
"INIT_EXCEPTION_HANDLER macro"§
ExcptInitialize(&excpt_struct)
#define FINI_EXCEPTION_HANDLERXE
"FINI_EXCEPTION_HANDLER macro"§
ExcptTerminate(&excpt_struct)
#define ExcptLinkHandlerXE "ExcptLinkHandler()
macro"§(a,b) _ExcptLinkHandler((a),
(b),&excpt_struct.excpt_info)

#define EXCPT_DIO_ERRXE "EXCPT_DIO_ERR"§
                     0x60000001
#define EXCPT_FAILUREXE "EXCPT_FAILURE"§
                     0xE0000002
#define EXCPT_SYS_ERRXE "EXCPT_SYS_ERR"§
                     0xE0000003
#define EXCPT_SET2BIGXE "EXCPT_SET2BIG"§
                     0xE0010001
#define EXCPT_MEMERR1XE "EXCPT_MEMERRx"§
                     0xE0020001
#define EXCPT_MEMERR2        0xE0020002
#define EXCPT_MEMERR3        0xE0020003
#define EXCPT_MEMERR4        0xE0020004
#define EXCPT_MEMERR5        0xE0020005
#define EXCPT_MEMERR6        0xE0020006
#define EXCPT_MEMERR7        0xE0020007
#define EXCPT_MEMERR8        0xE0020008
#define EXCPT_MEMERR9        0xE0020009
#define EXCPT_MEMERRA        0xE002000A
#define EXCPT_MEMERRB        0xE002000B
#define EXCPT_MEMERRC        0xE002000C
#define EXCPT_UNSHADOXE "EXCPT_UNSHADO"§
                     0x60030001
#define EXCPT_XYOUTSDXE "EXCPT_XYOUTSD"§
                     0x60040001
#define EXCPT_CTRE_DFXE "EXCPT_CTRE_DF"§
                     0xE0050001
#define EXCPT_NO_LOGXE "EXCPT_NO_LOG"§
                     0x60050002
#define EXCPT_VAL_CNVXE "EXCPT_VAL_CNV"§
                     0x60060001

void COMMON ExcptInitialize(XE
"ExcptInitialize()"§pEXCPT_STRUCT);
void COMMON ExcptTerminateXE
"ExcptTerminate()"§(pEXCPT_STRUCT);
void COMMON ExcptPost(XE "ExcptPost()"§ULONG,
UINT, const char *, ULONG);
ULONG APIENTRY ExcptHndlr(XE
"ExcptHndlr()"§PEXCEPTIONREPORTRECORD,
                  PEXCEPTIONREGISTRATIONRECORD,
                  PCONTEXTRECORD,
                  PVOID);
void COMMON _ExcptLinkHandler(XE
"ExcptLinkHandler()"§ULONG, void COMMON (*)(const
char *, ULONG), pEXCPT_INFO *);
void COMMON Fail(XE "Fail()"§const char *, const char *,
const char *, const char *);
```

3 Posting exceptions

To post exceptions, use the following call:

```
void COMMON ExcptPost(XE "ExcptPost()"§      ULONG exception_code,
                                             UINT info_block_size,
                                             const char *info_block,
                                             ULONG continuable_flag);
```

Where:

Exception_code is one of the exception codes declared in the headers file.

Info_block_size is the size, in bytes of the additional information block.

Info_block is a pointer to an additional information block, or ***NULL***.

Continuable_flag is either ***CONTINUABLEXE "CONTINUABLE"§*** or ***NON_CONTINUABLEXE "NON_CONTINUABLE"§***. A non-continuable

exception cannot be resumed, and the exception handler may not return.

4 Handling exceptions

A generic exception handler that can be dynamically set to enable or disable exception code handlers is part of the package. Exception handling must be established on a per thread basis. Un-handled exceptions are fielded by the operating system and abort the process with appropriate user notification.
The exception-code handlers simply return if the exception is continuable, or terminate the process if it is not. Each exception code handler is invoked for a particular exception code, and is called with a pointer to the additional information block supplied when the exception was posted, and the exception number.
A handler may also be linked with a zero exception number. Such a handler will be called for all user exceptions that are not specifically handled by another exception handler.
To link or delink exception handlers, use the following call:

ExcptLinkHandler(ULONG exception code, vPFN handler);

Where

Exception code is one of the defined exception codes or zero.

Handler is the address of an exception code handler to be linked, or ***NULL*** to delink.

5 Usage

The following example illustrates how exceptions are managed:

```
#define INCL_DOSEXCEPTIONS
#include <os2.h>
#include "sir32dll.h"
#include <excpt.h>
#include <string.h>
#include <base.h>
#include <stdio.h>
#include <stdlib.h>

static void COMMON Test(const char *, ULONG);

void main(int argc, char **argv)
{  EXCEPTION_STRUCTURE;
   INIT_EXCEPTION_HANDLER;
   ExcptLinkHandler(EXCPT_FAILURE,Test);
   Fail("String 1","String 2","String 3","String 4");
   FINI_EXCEPTION_HANDLER;
}

void COMMON Test(const char *p1, ULONG excpt_num)
{  pFAIL_EXCPT pfe;
   int cum_len;
   pfe = (pFAIL_EXCPT)p1;
   cum_len = 0;
   if(pfe->len1)
   {  printf("%0.*s\n",pfe->len1,pfe->texts+cum_len);
      cum_len += pfe->len1;
   }
   if(pfe->len2)
   {  printf("%0.*s\n",pfe->len2,pfe->texts+cum_len);
      cum_len += pfe->len2;
   }
   if(pfe->len3)
   {  printf("%0.*s\n",pfe->len3,pfe->texts+cum_len);
      cum_len += pfe->len3;
   }
   if(pfe->len4)
   {  printf("%0.*s\n",pfe->len4,pfe->texts+cum_len);
      cum_len += pfe->len4;
   }
   _exit(1);
}
```