eh.cm

```
/*
    ========================================================================

    Copyright (c) 2012-2016 Seppo Laakko
    http://sourceforge.net/projects/cmajor/

    Distributed under the GNU General Public License, version 3 (GPLv3).
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html
        )

    ========================================================================
    */

// Copyright (c) 1994
// Hewlett-Packard Company
// Copyright (c) 1996
// Silicon Graphics Computer Systems, Inc.
// Copyright (c) 2009 Alexander Stepanov and Paul McJones

namespace System.Support
{
    public nothrow bool HandleThisEx(int* exceptionBaseIdTable, int
        thrown, int catched)
    {
        if (thrown == catched)
        {
            return true;
        }
        int baseId = exceptionBaseIdTable[thrown - 1];
        while (baseId != 0)
        {
            if (baseId == catched)
            {
                return true;
            }
            baseId = exceptionBaseIdTable[baseId - 1];
        }
        return false;
    }

    internal class ExceptionTableRecord
    {
        suppress ExceptionTableRecord();
        suppress ExceptionTableRecord(const ExceptionTableRecord&);
        suppress ExceptionTableRecord(ExceptionTableRecord&&);
        suppress void operator=(const ExceptionTableRecord&);
        suppress void operator=(ExceptionTableRecord&&);
        public void* exceptionAddr;
    }
```

```
public nothrow void* GetExceptionAddr(void* exceptionTable, int
    exceptionId)
{
    ExceptionTableRecord* table = cast<ExceptionTableRecord*>(
        exceptionTable);
    ExceptionTableRecord* record = table + (exceptionId - 1);
    return record->exceptionAddr;
}

public nothrow void SetExceptionAddr(void* exceptionTable, int
    exceptionId, void* exception)
{
    ExceptionTableRecord* table = cast<ExceptionTableRecord*>(
        exceptionTable);
    ExceptionTableRecord* record = table + (exceptionId - 1);
    record->exceptionAddr = exception;
}

public class ExDeleter<T>
{
    public nothrow ExDeleter(T* ex_): ex(ex_)
    {
    }
    suppress ExDeleter(const ExDeleter<T>&);
    suppress void operator=(const ExDeleter<T>&);
    suppress ExDeleter(ExDeleter<T>&&);
    suppress void operator=(ExDeleter<T>&&);
    public nothrow ~ExDeleter()
    {
        reset_current_exception();
        if (current_exception_captured() == 0)
        {
            delete ex;
        }
    }
    private T* ex;
}

public const int EXIT_SEGMENTATION_FAULT = 255;
public const int EXIT_UNHANDLED_EXCEPTION = 254;

public nothrow void DestructorUnhandledException()
{
    Write(stderr, "unhandled exception reached a destructor\n");
    exit(EXIT_UNHANDLED_EXCEPTION);
}

public nothrow void MainUnhandledException()
{
    Write(stderr, "got unhandled exception from main\n");
    exit(EXIT_UNHANDLED_EXCEPTION);
}
```

```
    public class ExCodeChecker
    {
        public nothrow ExCodeChecker(int& exCode_): exCode(exCode_)
        {
        }
        suppress ExCodeChecker(const ExCodeChecker&);
        suppress void operator=(const ExCodeChecker&);
        public nothrow ~ExCodeChecker()
        {
            if (exCode != 0)
            {
                DestructorUnhandledException();
            }
        }
        private int& exCode;
    }
}
```