NSException

Inherits From:	NSObject
Conforms To:	NSCoding NSCopying NSObject (NSObject)
Declared In:	Foundation/NSException.h

Class Description

NSException is used to implement exception handling and contains information about an exception. An *exception* is a special condition that interrupts the normal flow of program execution. Each application can interrupt the program for different reasons. For example, one application might interpret saving a file in a directory that's write-protected as an exception. In this sense, the exception is equivalent to an error. Another application might interpret the user's keypress (i.e., Control-C) as an exception: an indication that a long-running process should be aborted.

Raising an Exception

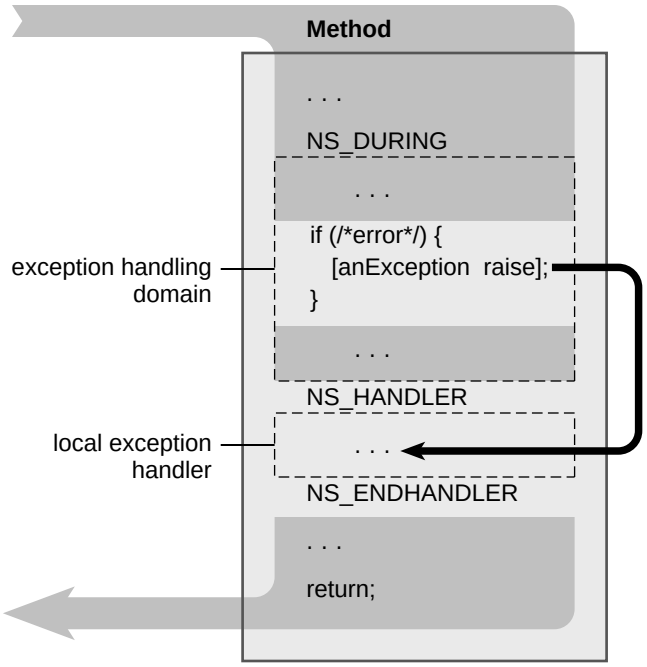
Once an exception is detected, it must be propagated to code that will handle it, called the *exception handler*. This entire process of handling an exception is referred to as "raising an exception." Exceptions are raised by instantiating an NSException object and sending it a **raise** message.

NSException objects provide:

- a name - a short string that is used to uniquely identify the exception.
- a reason - a longer string that contains a "human-readable" reason for the exception.
- userInfo - a dictionary used to supply application-specific data to the exception handler. For example, if the return value of a method causes an exception to be raised, you could pass the return value to the exception handler through **userInfo**.

Handling an Exception

Where and how an exception is handled depends on the context where the exception was raised. In general, a **raise** message is sent to an NSException object within the domain of an exception handler. An exception handler is contained within a control structure created by the macros NS_DURING, NS_HANDLER, and NS_ENDHANDLER, as shown in the following illustration.



The section of code between NS_DURING and NS_HANDLER is the *exception handling domain*; the section between NS_HANDLER and NS_ENDHANDLER is the *local exception handler*. The normal flow of program execution is marked by the gray arrow; the code within the local exception handler is executed only if an exception is raised. Sending a **raise** message to an exception object causes program control to jump to the first executable line following NS_HANDLER.

Although you can raise an exception directly within the exception handling domain, **raise** is more often invoked indirectly from a method invoked from the domain. No matter how deep in a call sequence the exception is raised, execution jumps to the local exception handler (assuming there are no intervening exception handlers, as discussed in the next section). In this way, exceptions raised at a low level can be caught at a high level.

For example, in the following program excerpt, the local exception handler displays an attention panel after detecting an exception having the name MyAppException. The local exception handler has access to the raised exception object through a local variable **localException**.

```
NS_DURING
    ...
    if (someError)
       [anException raise];
     ...
NS_HANDLER
    if ([[localException name] isEqualToString:MyAppException]) {
       NSRunAlertPanel(@"Error Panel", @"%@", @"OK", nil, nil,
           localException);
     }
    [localException raise];/* Re-raise the exception. */
NS_ENDHANDLER
```

You may leave the exception handling domain (the section of code between NS_DURING and NS_HANDLER) by:

- Raising an exception.
- Calling **NS_VALUERETURN()**
- Calling **NS_VOIDRETURN**
- “Falling off the end”

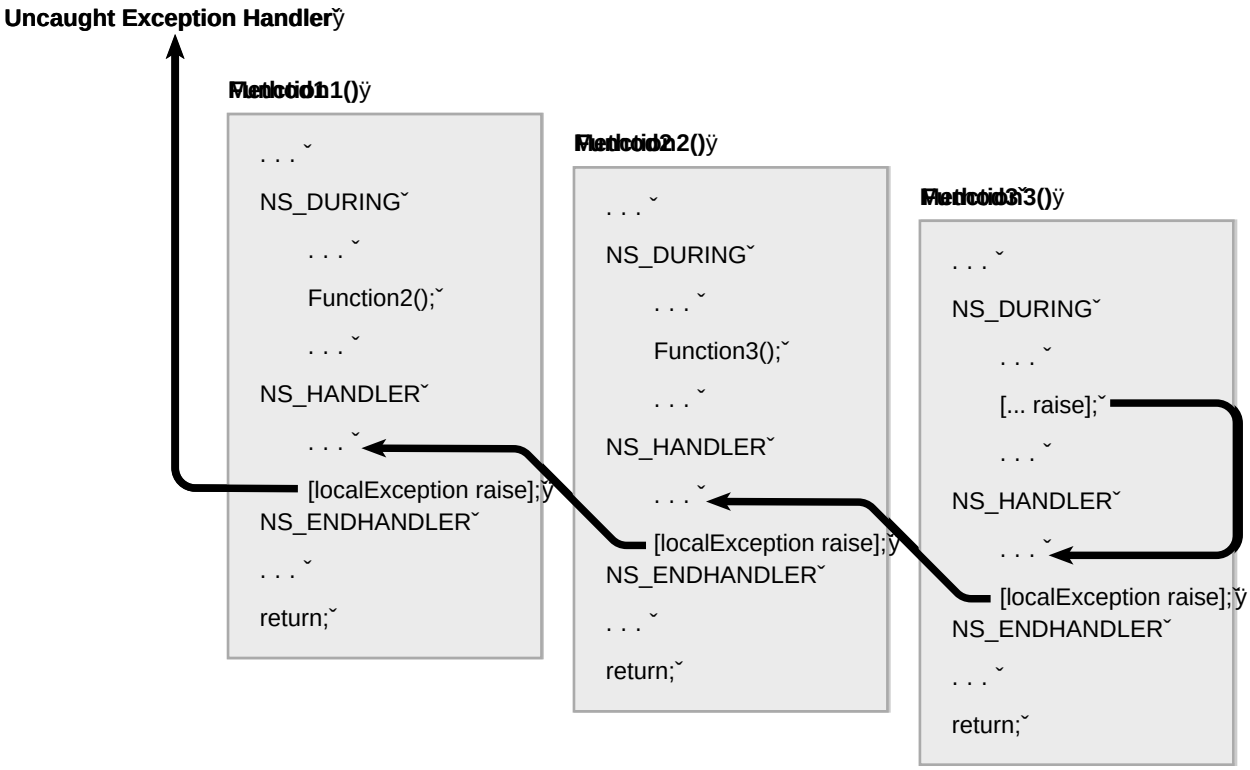
The above example raises an exception when **someError** is YES. Alternatively, you can return control to the caller from within the exception handling domain by calling either **NS_VALUERETURN()** or **NS_VOIDRETURN**. "Falling off the end” is simply the normal path of execution—after all statements in the exception handling domain are executed, execution continues on the line following NS_ENDHANDLER.

Note: You can’t use **goto** or **return** to exit an exception handling domain—errors will result. Nor can you use **setjmp()** and **longjmp()** if the jump entails crossing an NS_DURING statement. Since the NEXTSTEP code that your program calls may have exception handling domains within it, avoid using **setjmp()** and **longjmp()** in your application.

Similarly, you can leave the local exception handler (the section of code between NS_HANDLER and NS_ENDHANDLER) by raising an exception or simply "falling off the end".

Nested Exception Handlers

In the code example above, the same exception, **localException**, is raised again at the end of the local handler, allowing an encompassing exception handler to take some additional action. Exception handlers can be nested so that an exception raised in an inner domain can be treated by the local exception handler and any number of encompassing exception handlers. The following diagram illustrates the use of nested exception handlers, and is discussed in the text that follows.



An exception raised within Method3's domain causes execution to jump to its local exception handler. In a typical application, this exception handler checks the object **localException** to determine the nature of the exception. For exception types that it recognizes, the local handler responds and then may send **raise** to **localException** to pass notification of the exception to the handler above, the handler in Method2. (An exception that's re-raised appears to the next higher handler just as if the initial exception had been raised within its own exception handling domain.) Method2's exception handler does the same and then re-raises the exception to Method1's handler. Finally, Method1's handler re-raises the exception. Since there's no exception handling domain above Method1, the exception is transferred to the uncaught exception handler as described below.

Uncaught Exceptions

If an exception is not caught by any handler, it's intercepted by the uncaught exception handler, a function set by **NSSetUncaughtExceptionHandler()** and returned by **NSGetUncaughtExceptionHandler()**.

The default uncaught exception handler logs a message in the console and exits the program. However, for Application Kit programs, the message is logged with the Workspace Manager's console window (if the application was launched by the Workspace Manager) or to a Terminal window (if the application was launched from the shell).

You can change the default behavior by changing the uncaught exception handler using **NSSetUncaughtExceptionHandler()**.

Predefined Exceptions

NEXTSTEP predefines a number of exception names. These exception names are defined in **NSException.h**. For example:

```
NSGenericException
NSRangeException
NSInvalidArgumentException
NSMallocException
```

You can catch any of these exceptions from within your exception handler by comparing the exception's name with these predefined names. Note that all predefined exceptions begin with the prefix "NS", so you should avoid using the same prefix when creating new exception names.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:
NSCopying	– copyWithZone:

Method Types

Creating and raising an NSException	+ exceptionWithName:reason:userInfo: + raise:format: + raise:format:arguments: – initWithName:reason:userInfo: – raise
Querying an NSException	– name – reason – userInfo

exceptionWithName:reason:userInfo:

+ (NSException *)**exceptionWithName:**(NSString *)*name*
  reason:(NSString *)*reason*
  userInfo:(NSDictionary *)*userInfo*

Creates and returns an exception object using a predefined *name*, a human-readable *reason*, and arbitrary *userInfo*.

See also: **– initWithName:reason:userInfo:**, **– name**, **– reason**, **– userInfo**

raise:format:

+ (void)**raise:**(NSString *)*name* **format:**(NSString *)*format*,...

A convenience method that creates and raises an exception with name *name* and a reason constructed from *format* and the arguments that follow in the manner of **printf()**. The user-defined information is **nil**.

See also: **+ raise:format:arguments:**, **– raise**

raise:format:arguments:

+ (void)**raise:**(NSString *)*name* **format:**(NSString *)*format* **arguments:**(va_list)*argList*

Creates and raises an exception with name *name* and a reason constructed from *format* and the arguments in *argList*, in the manner of **vprintf()**. The user-defined information is **nil**.

See also: **+ raise:format:**, **– raise**

nstance Methods

description

– (NSString *)**description**

Overridden to return the receiver’s reason, so that "%@" used in formatted strings produces a meaningful description of the exception.

See also: – **reason**

initWithName:reason:userInfo:

– (id)**initWithName:**(NSString *)*name*
reason:(NSString *)*reason*
userInfo:(NSDictionary *)*userInfo*

Initializes a newly allocated exception object using the predefined *name*, human-readable *reason*, and user-defined *userInfo*. This is the designated initializer.

See also: **+ exceptionWithName:reason:userInfo:**, **– name**, **– reason**, **– userInfo**

name

– (NSString *)**name**

Returns an NSString used to uniquely identify the exception.

See also: **+ exceptionWithName:reason:userInfo:**, **– initWithName:reason:userInfo:**

raise

– (void)**raise**

Raises the exception, causing program flow to jump to the local exception handler. All other methods that raise an exception call this method, so set a breakpoint here if you are debugging exceptions.

See also: **+ raise:format:**, **+ raise:format:arguments:**

reason

– (NSString *)**reason**

Returns an NSString containing a “human-readable” reason for the exception.

See also: **– description**, **+ exceptionWithName:reason:userInfo:**, **– initWithName:reason:userInfo:**

userInfo

– (NSDictionary *)**userInfo**

Returns an NSDictionary that contains application-specific data pertaining to the receiver. Returns **nil** if no application-specific data exists. As an example, if a method’s return value caused the exception to be raised, the return value might be available to the exception handler through this method.

See also: + **exceptionWithName:reason:userInfo:**, **– initWithName:reason:userInfo:**