OPENSTEP 4.2 Release Notes: Foundation Framework

The Foundation Framework is a library of Objective-C classes providing the infrastructure for non-user-interface object-based applications and other user-interface- and non-user-interface-based frameworks. The OPENSTEP Foundation is compliant with the Foundation portion of the OpenStep specification, and enriches OpenStep with a few extensions.

Within the Foundation are basic classes used by any Objective-C program: collection classes, Distributed Objects, persistence frameworks, Unicode and internationalization support, the foundations of event-driven programming, and facilities to insulate code from the underlying operating system.

These release notes are divided in three main sections:

- Comments on compatibility with OPENSTEP 4.0 and 4.1
- General comments on often-encountered problem areas and interesting things
- Known problems in this release

In the document below, "4.0" and "OPENSTEP 4.0" refer to all products that shipped with the 4.0 Foundation: OPENSTEP 4.0, D'OLE 4.0, and PDO 4.0 for HP-UX™ and Solaris™. "4.1" and "OPENSTEP 4.1" refer to OPENSTEP 4.1 for Mach and OPENSTEP Enterprise 4.1. "4.2" and "OPENSTEP 4.2" refer to OPENSTEP 4.2 for Mach and OPENSTEP Enterprise 4.2. All trademarks are property of their respective owners.

Changes and Compatibility Between OPENSTEP 4.0, 4.1,

and 4.2

There have been a number of changes to Foundation since the shipment of OPENSTEP 4.0 and a few since 4.1, both defect fixes and a few API additions. These are enumerated in the section below.

Note that if your application uses API which was added after 4.0, your application might not run under 4.0 (and similarly with 4.1 and API added after 4.1). (If you do not want to deploy your application on 4.0 or 4.1, then this is not an issue.) You can catch some of the potential problems by defining the macros **STRICT_40** and/or **STRICT_41** at compile time. This #define is similar to **STRICT_OPENSTEP** and is used in the Application and Foundation Frameworks to mark new API. Similarly, if you rely (perhaps unintentionally) on a bug fixed in 4.2, your application may not function correctly under 4.0 or 4.1. Make sure you test on your intended deployment platform.

A number of performance improvements have also made to Foundation for each release. Customers developing on 4.2 may find performance to be significantly worse in some cases when an application is deployed on 4.0, for example.

Here are the feature changes to Foundation in 4.1 (from 4.0):

- Some methods in 4.0 were not surrounded by **#if !defined(STRICT_OPENSTEP)** when they weren't, in fact, in OpenStep.

CLASS	HEADER	METHOD
NSCharacterSet	NSCharacterSet.h	+characterSetWithContentsOfFile:
NSConnection	NSConnection.h	-connection:shouldMakeNewConnection:
NSArray	NSPathUtilities.h	-pathsMatchingExtensions:

- These methods were made public in 4.1. They were not public in 4.0, but existed in the 4.0 Foundation binary so there are no compatibility issues in using them. They are not in OpenStep, however.

CLASS	HEADER	METHOD
NSArray	NSArray.h	-makeObjectsPerformSelector:
NSArray	NSArray.h	-makeObjectsPerformSelector:withObject:
NSSet	NSSet.h	-makeObjectsPerformSelector:
NSSet	NSSet.h	-makeObjectsPerformSelector:withObject:

- These methods and functions were added to Foundation in 4.1 and are not present in the 4.0 Foundation binary. Applications which do not take some precaution in using them will not run correctly on OPENSTEP 4.0. These methods and functions are also not in OpenStep.

CLASS	HEADER	METHOD
NSBundle	NSBundle.h	+allBundles
NSBundle	NSBundle.h	+allFrameworks
NSFileHandle	NSFileHandle.h	+fileHandleWithNullDevice
NSFileHandle	NSFileHandle.h	-initWithNativeHandle:closeOnDealloc:
NSFileHandle	NSFileHandle.h	-initWithFileDescriptor:closeOnDealloc:
NSObject	NSObject.h	+instanceMethodSignatureForSelector:

FUNCTION	HEADER
NSFrameAddress	NSDebug.h
NSReturnAddress	NSDebug.h
NSCountFrames	NSDebug.h

Here are the feature changes to Foundation in 4.2 (from 4.1):

- Some methods in 4.1 were not surrounded by **#if !defined(STRICT_OPENSTEP)** when they weren't, in fact, in OpenStep. They were guarded with **#if !defined(STRICT_40)**.

CLASS	HEADER	METHOD
NSBundle	NSBundle.h	+allBundles
NSBundle	NSBundle.h	+allFrameworks
NSObject	NSObject.h	+instanceMethodSignatureForSelector:

- This method was made public in 4.2. It was not public in 4.0 or 4.1, but existed in the 4.0 and 4.1 Foundation binaries so there is no compatibility issues in using it. It is not in OpenStep, however.

CLASS	HEADER	METHOD
NSDictionary	NSDictionary.h	-initWithDictionary:copyItems:

- This method was made public in 4.2. It was not public in 4.1, but existed in the 4.1 Foundation binary so there is no compatibility issues in using it for 4.1 deployment. It does not exist in the 4.0 binary. It is also not in OpenStep.

CLASS	HEADER	METHOD
NSBundle	NSBundle.h	-load

- These global variables were added to Foundation in 4.2 and are not present in the 4.0 nor the 4.1 Foundation binaries. Applications which use these global variables will not run on OPENSTEP 4.0 or 4.1. These global variables are not in OpenStep.

VARIABLE	HEADER
NSFileHandleDataAvailableNotification	NSFileHandle.h
NSPositiveCurrencyFormatString	NSUserDefaults.h
NSNegativeCurrencyFormatString	NSUserDefaults.h

- These methods were added to Foundation in 4.2 and are not present in the 4.0 nor the 4.1 Foundation binaries. Applications which do not take some precaution in using them will not run correctly on OPENSTEP 4.0 or 4.1. These methods are not in OpenStep.

CLASS	HEADER	METHOD
NSFileHandle	NSFileHandle.h	-waitForDataInBackgroundAndNotifyForModes:
NSFileHandle	NSFileHandle.h	-waitForDataInBackgroundAndNotify
NSRunLoop	NSRunLoop.h	-configureAsServer
NSTask	NSTask.h	-interrupt

- Some API was added to **NSDebug.h**. Production programs should not be referring to symbols declared in **NSDebug.h**.

General Comments

NSDates and the Year 2000

Dates are represented in Foundation with NSDate (and NSCalendarDate) objects and NSTimeInterval values. An NSTimeInterval is essentially a C-language double. NSDates (and NSCalendarDates) store the date that they represent with an NSTimeInterval, which represents the time delta, in seconds, from the Foundation reference date, 00:00:00 1 January 2001 GMT. This time delta is negative until the reference date. There is no special significance to the year 2000 to NSDate objects or NSTimeInterval values. However, refer to the topic *Risks in Parsing and Displaying Dates* in this section for more information on

issues in date presentation and the year 2000.

Since the internal time representation will transition across zero (at the reference date) at some point in the future, unlike some other schemes which set the epoch to a date "always" in the past, there will be a brief instant, on the order of microseconds in length, in which an NSDate object created to represent the current time will have an NSTimeInterval of 0.0. The length of this instant is dependent on the resolution of the system time provided by the particular operating system and platform an application is running on. An NSTimeInterval of 0.0 has no special significance to NSDate or NSCalendarDate, but applications which use the NSTimeInterval of a date object in division should protect against division by zero. Such applications are vanishingly rare. More common might be to have an NSTimeInterval which represents the delta between two arbitrary date objects (such as when timing events), and applications should take appropriate caution in using such values (this is always a good idea apart from date usage).

Risks in Parsing and Displaying Dates

As with any communication between a program and a user, there can be miscommunication in showing dates to the user and reading dates the user types in. When a user types a date into a text field in an application they may intend one thing, and the program may interpret the date as something else. When a date is displayed to a user, the user may interpret the date incorrectly. While an application cannot control the user, there are certain things that an application can avoid doing, with respect to dates, to reduce the potential for confusion and ambiguity and data corruption.

- Avoid two-digit years and all uses of the "%y" date format specifier.
- Do not use natural language date parsing.
- Avoid time zone abbreviations.

Each of these points is explained in greater detail below. Whenever it is important (or critical) that the user get accurate information from the application, or the application get accurate information from the user, an application should avoid the ambiguous behaviors below and require strict conformance to a fully specified input format (for user input) and provide an equally verbose output description of dates. Alternatively an application could ask for confirmation of input dates, but this seems to be rather more heavyweight.

Avoid two-digit years and all uses of the "%y" date format specifier. Per POSIX and ANSI C (refer to

section 7.12, ANSI/ISO 9899-1990 for ANSI C definitions), a two-digit year *YY* means 1900 + *YY.* The Foundation follows the lead of ANSI C here. Thus, when a user types "5/10/02" in to a text field with a date formatter with format "%d/%m/%y", a date of 5 October 1902 is successfully created. If the user meant 5 October 2002, they haven't entered that. Since %y means "two-digit year" it is not possible for the user to type anything that will get them a date in the year 2002 (for example, if the user types "5/10/2002" in this example, a date of 5 October 1920 will be successfully parsed). The date parsing routines parse at most two-digits for %y, in order to support dates formatted like "980409" (also not a good idea). Conversely, a user has no way of knowing whether "28/10/33" refers to 1933 or 2033 if they see that displayed on the screen or in printed output. If a program uses "%Y" in the format string instead, this problem is avoided.

Do not use natural language date parsing. The natural language parser is pretty sophisticated, but it is after all just sophisticated guessing. The natural language parser is also usually successful at creating a date from input, as long as the input is moderately close to something that looks like a date. However, as with the previous problem, this may not be the date the user intended. Relatedly, don't enable natural language parsing for date formatters. If it is enabled, and the user's input does not match the date formatter's format, the input string is given to the natural language parser.

Here is an example where the natural language parser may fail to parse the date the user intended: suppose the date formatter has a format "%d/%m/%Y" and the user types in "5-10-2002". This does not match the format (which has slashes), so the date formatter gives the input to the natural language parser. Now, what the "5" means and what the "10" means is ambiguous±it could equally be month-day-year or day-month-year. The natural language parser prefers a particular ordering, given by the NSDateTimeOrdering preference. If the user's preferred language is French, this is DMYH (denoting an ordering: day month year hour). The built-in default, if the user has no preferred language, is MDYH (denoting an ordering: month day year hour). NSDateTimeOrdering could also be set explicitly to something else by the user. If the user's preferred language is French, the date will be "correctly" parsed as 5 October 2002. By default it will be parsed as 10 May 2002. In general, an application writer can't know *a priori* what the preferences of the user will be. If natural language parsing is desired, this and other issues (such as use of 12- or 24-hour hour designations) should be taken into account and dealt with by the application.

Avoid time zone abbreviations. Time zone abbreviations are inherently ambiguous. Many time zones around the world map to the same abbreviations, by local convention, and there is usually very little

information in the abbreviations (most are of acronyms of a phrase such as "*Something* Standard Time" or "*Something* Daylight Time", where the *Something* is the only differentiator). A time of 11:00 CST means something different to a person in New York, New York, U.S.A. (indicating time in Chicago) than someone in Sydney, Australia (indicating time in Darwin) (that hour by itself is also ambiguous±is it *ante meridiem* or *post meridiem*?). Use of time zone names of the form GMT+hhmm and GMT-hhmm (as in GMT+0100) can also be problematic, since the user cannot know what which convention (simple offset, or the POSIX minutes-west) is being used. Thus, GMT+0200 can indicate either Helsinki or the mid-Atlantic.

Foundation's Time Zone Data

The Foundation now uses its own time zone data files (which describe when changes to the standard time zone corrections from GMT occur) on all platforms. This provides for greater consistency of behavior, and the ability to unarchive and transport time zones to any Foundation platform. However, it's possible that the time zone data may become out-of-date relative to a particular platform's data as, for example, new operating system updates are installed, but the new versions of PDO or OPENSTEP are not. The Foundation's time zone data comes from a freely available source at **elsie.nci.nih.gov**, and is current with that source as of November, 1996. The **Resources/TimeZoneInfo** directory inside the Foundation.framework contains a file **README.TimeZoneInfo** which explains how the time zone data for Foundation can be updated from this source, if required. Part of the reason for choosing the data from **elsie.nci.nih.gov** is that it is an easily accessible source of current information (it is updated about once per month). It is also quite complete.

The time zone data that Foundation uses in 4.2 is compatible with the data used in 4.0 and 4.1. However, to save disk space on machines with disks with large allocation-block sizes, all of the data is bundled up into a serialized property list (**TimeZones.splist**). The top-level property list is a dictionary where the keys are time zone names (the same as the file names in 4.1) and the values are NSData objects containing the bytes of the files. To use the data to update a 4.1 installation (for example), the data objects in this file would need to be extracted into files with the file names given by the keys. It is fairly straightforward to write a Foundation-based program to do this. An example of the opposite process, taking a directory hierarchy and making the serialized property list is given in the **Resources/TimeZoneInfo/README.TimeZoneInfo** file described above.

With the adoption of the time zone data from **elsie.nci.nih.gov**, Foundation has also adopted the zone

naming conventions of that data, where zone names usually have the form Area/Location, as in America/Sao_Paulo or Europe/Paris. The previous, and conventional UNIX, time zone names are supported for backwards compatibility.

The set of time zone abbreviations has also changed somewhat. How abbreviations map to time zones is an arbitrary choice of the Foundation, since time zone abbreviations are ambiguous. Use of time zone abbreviations is not encouraged (see *Risks in Parsing and Displaying Dates* in this section for more information). Sites can customize this to some extent±see the file **Resources/TimeZoneInfo/README.TimeZoneInfo** in the Foundation.framework.

Using performSelector: with Selectors Which Don't Return `id'

The NSObject methods **-performSelector:**, **-performSelector:withObject:**, and **-performSelector:withObject:withObject:** have an "id" return value. This is something of a holdover from pre-4.0 conventions where methods without otherwise interesting return values returned **self**. Much of the time one can "get away with" performing a selector to an object with these methods, where the target method does not have a return value or has a non-object return value. Such use is semantically suspect, however, and is not valid. Any situation where the **-performSelector:...** message may be forwarded will quickly illustrate this (with an abnormal program termination), as the NSInvocation object attempts to retain the return value from after invocation, because the selector for **-performSelector:...** is typed to return an object. Sending **-performSelector:** to a faulted object with a selector that does not return id is an example of this. This behavior extends also to various other "perform" methods such as the **-makeObjectsPerformSelector:** of some collection classes and the "delayed perform" and "ordered perform" methods in **NSRunLoop.h**.

Where Do NSLog()'d Messages Go?

NSLog() and **NSLogv()** do not log an error message to **stderr**, as stated in the documentation.

On HP-UX, Solaris, and Mach, it writes the log to STDERR_FILENO if the file descriptor is open. If that fails, the message is sent to the syslog subsystem, if it exists on a platform, with the LOG_USER facility (or default facility if LOG_USER doesn't exist on a platform), with priority LOG_ERR (or similar, depending on what the platform supports). If both of these attempts to write the message fail, the message is

discarded.

On Windows platforms, the message is written to the STD_ERROR_HANDLE, if that handle is valid, on Windows platforms that support that standard handle. It is also written to the Windows Event Log on Windows platforms that support the Event Log, or to the file **c:\fndation.log** on Windows platforms that do not, if that file can be opened. If all of these attempts fail, the message is discarded. On some Windows platforms, the message to the Event Log may be truncated if there is a limit to the size of a message that the Event Log can accept. On Windows platforms that support an application discovering whether or not it is running under a debugger, **NSLog()** and **NSLogv()** may only send the message to the debugger for its handling, via standard WIN32 mechanisms, and not also write the message to STD_ERROR_HANDLE and the Event Log. Note that a debugger may choose to not display messages thus sent to it, or may choose not to display all of the message±**NSLog()** and **NSLogv()** have no control over that.

Output from **NSLog()** and **NSLogv()** is serialized, in that only one thread in a process can be doing the writing/logging described above at a time. All attempts at writing/logging a message complete before the next thread can begin its attempts.

The effects of **NSLog()** and **NSLogv()** are not serialized with other subsystems than those discussed above (such as the standard I/O package) and do not produce side effects on those subsystems (such as causing buffered output to be flushed, which may be undesirable).

The format specification allowed by **NSLog()** and **NSLogv()** is that which is understood by NSString's formatting capabilities. That is not necessarily the set of format escapes and flags understood by **printf()**, as mentioned in the documentation.

Foundation Examples

There are several Foundation examples in **/NextDeveloper/Examples/Foundation**.

Various examples illustrating the use of Distributed OLE and the NeXT ORB can be found on NeXTanswers. NeXTanswers can be accessed through **http://www.next.com/**.

I/O Functionality

Two Foundation classes, NSFileHandle and NSPipe, provide objects that represent open files or communications channels and support basic I/O operations. They make it easy for developers to write code that is directly portable to other platforms. The interfaces for these classes are declared in **NSFileHandle.h**. These classes replace the NSPosixFileDescriptor and NSPosixPipeDescriptor classes made available in OPENSTEP 4.0 prereleases. The differences from the predecessor classes for the most part involve naming and a refinement of semantics.

Note that the new (in 4.2) **-waitForDataInBackgroundAndNotify** functionality only works for NSFileHandles which refer to sockets.

Archiving

NSUnarchivers own the objects that they decode. When an NSUnarchiver is deallocated, so are the objects that it has decoded, unless they have been retained. The following code, for example, will probably result in a decoded object being used after being freed:

```
    NSAutoreleasePool *pool = [[NSAutoreleasePool allocWithZone:NULL] init];
    NSUnarchiver *unarchiver = [[NSUnarchiver allocWithZone:NULL]
initForReadingWithData:myData];
    id object = [unarchiver decodeObject];
    [unarchiver release];
    [pool release];
    ... use or return object here ...
```

`object` should be retained before the NSUnarchiver is released, and probably also autoreleased if `object` is returned from the function or method. An exception to this are the **-decodeValueOfObjCType:at:** and **-decodeValueOfObjCTypes:...** methods. Objects "returned" from these two methods, like those returned from **+allocWithZone:** and **-copyWithZone:**, are not autoreleased, and must be explicitly released.

User Defaults

Foundation contains API (NSUserDefaults) which replaces the old defaults database suite of functions. User defaults with 4.0 applications are stored differently and in a different place than 3.3 user defaults. There is a new command-line utility called **defaults** that allows you to manipulate the new-style defaults just as **dread**, **dwrite**, and **dremove** allowed you to manipulate old-style defaults.

On Windows NT, defaults are stored in the Windows registry. There is a program shipped with Windows NT, **REGEDT32**, for viewing and editing registry entries. NSUserDefaults stores its domains and their key-value pairs under the entry **HKEY_CURRENT_USER\Software\NeXT\UserDefaults**. The values are currently stored as strings in the property list format.

The command **/usr/bin/defconvert**, in OPENSTEP 4.x for MACH, can be used to convert a user's 3.3 defaults database to 4.x-style defaults±simply run it from the command line with no parameters. This program converts all old-style defaults, even those of non-OpenStep applications, adding them to the user's new-style database. Since NEXTSTEP 3.3 applications may not use the same default names after conversion to OPENSTEP 4.x, and 3.3 applications do not use the new defaults database, and increasing the size of the defaults database increases the amount of memory needed by each OpenStep application, converting an old defaults database to the new style usually provides little utility.

Errors in ASCII Property Lists

When a method like NSDictionary's +**dictionaryWithContentsOfFile:** is used to read and parse an ASCII property list from a file, parse errors and exceptions are suppressed, and **nil** is returned upon error. Sometimes a program wants to know about syntax errors in a property list, or wants finer-grained information about why a property list cannot be read and parsed. Another method, NSString's -**propertyList**, can be used for this purpose. The following example illustrates this:

```
- (id)readPropertyListFromFile:(NSString *)path mustBeOfType:(Class)targetClass logErrors:
(BOOL)showErrors {
    NSString *string;
    id plist = nil;

    string = [[NSString allocWithZone:NULL] initWithContentsOfFile:path];
    if (showErrors && nil == string) {
        NSLog(@"%@: string could not be read from '%@'", NSStringFromSelector(_cmd), path);
    }
    NS_DURING
        plist = [string propertyList];
    NS_HANDLER
        if (showErrors) {
            NSLog(@"%@: received exception while parsing: %@", NSStringFromSelector(_cmd),
localException);
```

```
        }
        plist = nil;
    NS_ENDHANDLER
    [string release];
    if (Nil != targetClass && ![plist isKindOfClass:targetClass]) {
        if (showErrors) {
            NSLog(@"%@: property list is not of desired type '%@'. It's an '%@'.",
                  NSStringFromSelector(_cmd), NSStringFromClass(targetClass),
NSStringFromClass([plist class]));
        }
        [plist release];
        plist = nil;
    }
    return plist;
}
```

Grammars for Serialization and ASCII Property List Formats

Grammars and description of the serialization (NSSerializer) and ASCII property list representation formats are available in the appendix to the OpenStep specification.

Strings which contain non-alphanumeric characters must be double quoted in property lists to ensure the entire string is parsed as one string object. Parse failures will result, otherwise.

Retain Cycles and Invalidation

The retain/release strategy used in Foundation allows an incautious programmer to create reference cycles in an object graph (that is, the graph of objects that retain one another in some manner). These cycles are self-sustaining and constitute a memory leak. The simplest example is to create a NSMutableArray instance and add it to itself. Since an array retains its objects, it retains itself independent of its normal usage. Retains are essentially distributed across the network via the Distributed Objects mechanism.

If retain cycles cannot be avoided by careful design, but are known *a priori*, the following technique can be used to recover the memory. For the objects in the cycle(s) whose retain counts are solely due to the cycle, all should be retained (perhaps in an array), each one told to release its retained objects, then all

released. The action of releasing its retained objects would undoubtedly cause an object to become dysfunctional. Some objects already implement such a method by the name "-invalidate". It is a bug that neither NSObject nor the collections implement this method to facilitate cycle recovery.

More on Autoreleasing and Retaining

The following statements are false:

Returned objects are guaranteed to be valid for the scope of the current method.
Returned objects are guaranteed to be valid until the current autorelease pool is released.
Returned objects are guaranteed to be valid until the end of the current event loop.

The Foundation's retain count mechanism operates via **-retain** and **-release**. A code fragment such as:

```
id object = [collection returnObject];
```

creates a reference (in `object`) to the returned object, but does not increment the retain count of `object`. If you don't let the system know about your reference, by incrementing the retain count with **-retain**, the system can't ensure that your reference remains valid for any length of time. For example:

```
object = [array objectAtIndex:7];      // get object
[array removeObjectAtIndex:7];         // removes object and releases it
[object dosomething];                  // object may be invalid here
```

In that example, if `array` had the last retain on `object`, the second line will cause `object` to be deallocated and the third line will cause the application to crash. Here is another example with the AppKit:

```
id oldTitle = [myWindow title];        // get current title
[myWindow setTitle:@"Calculating..."]; // set temporary title
... do calculation here ...
[myWindow setTitle:oldTitle];          // restore previous title
```

Here, the **-setTitle:** method gets rid of its reference to the existing title string and takes a reference to the new title string. If the window had the only reference to the previous title string object, the last line of the example will attempt to **setTitle:** to a freed object. If the AppKit in its implementation happens to send **-autorelease** to the old title instead of **-release**, then the object referenced in `oldTitle` may not by freed by the time of the last line, but in general you cannot know whether such a method will release or autorelease the previous object. The Foundation almost exclusively uses **-release**, since that does not have the performance overhead of autoreleasing, and Foundation objects are so common in applications.

In practice, you can get away with such "weak references" much of the time. But the safest approach formalizes your reference to the object:

```
id object = [[collection returnObject] retain];
... do operations, some of which may be on object ...
/* Don't need object any longer */
[object release];
object = nil;
```

The -hash and -isEqual: Invariant

It is worthwhile to highlight an obscure requirement documented in NSObject class specification for the **-hash** method: The **-hash** and **-isEqual:** methods of a class must be defined so that if two objects are equal, their hash value is the same ([x isEqual:y] implies [x hash] == [y hash], but not the converse). This is easy to forget, but especially important to ensure when putting custom objects into some collections (such as NSSet and NSDictionary).

Forwarding Messages in OpenStep

Objects wishing to forward messages with the **-forwardInvocation:** method must also (re)implement the **-methodSignatureForSelector:** method. This is a result of the compiler not providing call stack descriptive information at each call site. Consequently, the runtime needs a source of information for how the stack frame is constructed. It needs this to build the invocation that will be supplied to the **-forwardInvocation:** method call. The **-methodSignatureForSelector:** method will need to be reimplemented to provide a method signature for selectors to which the object does not respond.

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)sel {
    NSMethodSignature *result = [super methodSignatureForSelector:sel];

    // if result is nil, self must not respond to the selector
    if (nil == result)
        result = [target methodSignatureForSelector:sel];
    return result;
}
```

```
- (void)forwardInvocation:(NSInvocation *)invocation {
    [invocation invokeWithTarget:target];  // let target implement methods that we do not
}
```

Faulted objects in EOF which wish to forward messages should also override **+instanceMethodSignatureForSelector:**.

See the ForwardInvocation example in **/NextDeveloper/Examples/Foundation/ForwardInvocation**.

Debugging Aids

The *OpenStep Conversion Guide* describes several techniques for debugging Foundation and AppKit applications. The header file **NSDebug.h** also has some global flags, functions, and methods which may be useful. But be warned±although this header file is public, its contents are mostly unsupported, largely undocumented, and are subject to change without warning. Do not depend on API within this header for the operation of production programs.

Object Allocation Analysis

One of the useful flags defined in **NSDebug.h** is **NSKeepAllocationStatistics**, which allows you to analyze the object allocation activity in a program. The **AnalyzeAllocation** command line program can be used to analyze the output generated by running with this option enabled. You can also use the **ObjectAlloc** demo application which comes OPENSTEP for Mach and Windows to graph the object allocation activity in real time.

The +load Method

The **+load** message is sent to classes when they are added to the Objective C runtime. **+load** is usually received before **+initialize**. **+load** is not inherited by subclasses.

The order in which **+load** messages are sent to classes is unspecified (and specifically, a class's superclass is not guaranteed to have its **+load** method called before the class receives the message). Therefore, you should not use any subsystems or classes which are loaded at the same time, or more generally, which you do not know have received the **+load** message. If you do, you will probably cause

execution of code which assumes that **+load** (and anything that *had* to be done in **+load**) has been previously executed.

There are two situations typically when classes are added to the runtime:

- when loaded dynamically from a bundle or framework by NSBundle
- at application launch

The restrictions on **+load** are perhaps most serious at launch time. The Foundation has classes and subsystems (as might any other library) which depend on **+load** being called before they are fully functional. For example, you should not use **NSZoneMalloc()** to allocate memory in **+load** methods which are statically linked into an application. You certainly should not create any objects, or do anything with creates objects.

Note that calling **+load** on a class isn't a solution (and is a generally bad idea), because that will cause **+initialize** to be sent first, which the class may not handle. Not to mention that doing so places into your code dependencies on which classes implement **+load** in a particular version of a library. In the case of Foundation, for convenience sake, **+load** methods which must use Foundation API can call `[NSObject self]` first to make sure Foundation has performed the initialization it needs.

+load is not intended as a general initialization mechanism, and should only be used when absolutely necessary. Use **+initialize** for early initialization whenever possible.

Performance and NSNotificationCenter

NSNotificationCenter, although heavily optimized, by its nature can become a significant performance bottleneck if heavy use of notifications is made. The notification mechanism should be used judiciously.

NSAutoreleasePool's -addObject: Method

You should never use the NSAutoreleasePool instance method **-addObject:**. Use the **-autorelease** method, or **[NSAutoreleasePool addObject:]** to autorelease an object. Using the instance method incurs the risk of adding an object to a pool which is not the top autorelease pool, a semantically suspect operation, since the autorelease pool model is one of a (per-thread) stack of pools. OPENSTEP 4.x does allow you to add objects to an autorelease pool that is not the top pool, but be warned that it is a much

more expensive operation (and always will be) than adding an object to the top pool.

Frameworks as NSBundles

Frameworks are a specialized form of bundle. An application can statically link against a framework at compile time (like a library), or dynamically load a framework (like a bundle) at runtime. However, as a specialized type of bundle, there are some constraints placed on frameworks:

- A framework must have a **.framework** extension
- The name of the executable code of a framework must have the same name as the framework directory, minus the **.framework** extension. For example, the file name of the executable code for the framework **MyFramework.framework** must be **MyFramework** (or on Windows, **MyFramework.dll**).
- On Windows only: The framework directory must be stored in a directory called **Frameworks**. The executable code (.dll) must be stored in a directory called **Executables** that has the same parent as the **Frameworks** directory. If the framework is statically linked into applications, the **Executables** directory (full path) should be in the PATH environment variable. (On other OPENSTEP platforms, the executable code file is stored within the framework directory, like a bundle.)

How NSBundles Search for Resources

The algorithm that NSBundles use to find resources has changed somewhat from that of NXBundles, and there are changes to the structure of bundles themselves. A bundle's resources are now stored in a directory named **Resources** within the bundle directory. Within the **Resources** directory are the non-localized resources and the localized resource directories (**English.lproj**, **Swedish.lproj**, etc., as with NXBundle). The addition of the **Resources** directory is primarily a way to simplify Framework versioning. Other changes have been made to reduce the amount of computation required to find resources.

Suppose we are searching for the resource with name **Main** and type **nib**. The bundle's resource path, the "top level", is searched first, for the file **Main.nib**. If the file is not found there, each of the language subdirectories are each searched, in the user's preference order. This is a change from NXBundle's behavior, where the language subdirectories were searched first. This change means that the localized version of a resource that also exists at the top level will not be found, but the top level one will be. All non-localized resources should be placed in the top level, and no localized resources should exist at the top level.

When a resource is found, the algorithm checks to see if a resource of the name **Main-*$(PLATFORM_OS)*** of type **nib** exists. ***$(PLATFORM_OS)*** represents the **make** variable of the same name, and takes on the same values that the **make** variable takes on at compile time. For example, on Windows, during a project build, ***$(PLATFORM_OS)*** in the **Makefile.postamble** will have the value "winnt". When **Main.nib** is found in a directory, the search algorithm does one final check to see if **Main-winnt.nib** exists in that same directory. If it does, the search algorithm returns the path to that resource; if it does not, the search algorithm returns the path to **Main.nib**. Note that for **Main-winnt.nib** to be found, a file named **Main.nib** must exist in the same directory; this also applies within each language resource directory. It is expected that a developer will choose one of the platform-specific versions to name without this ***$(PLATFORM_OS)*** suffix, and give the other platform-specific versions of the resource extended names. The values that ***$(PLATFORM_OS)*** can take on are currently "winnt" (on OPENSTEP for Windows), "nextstep" (on OPENSTEP for MACH), "hpux"(on PDO for HP-UX), and "solaris" (on PDO for Solaris).

Another way to accommodate platform-specific resources is by using the **inDirectory:** parameter of the resource-searching methods. The **inDirectory:** parameter is primarily intended to facilitate the collection of resources of a common type or purpose into a single directory; for example, all images could be put into a directory called **Images** within the resource directory. (Note that the AppKit's **+imageNamed:** method in NSImage does not support searching arbitrary directories for images.) As such, the **inDirectory:** parameter can be used to manage platform-specific resources. This is less "automatic" however, and requires the developer to specify the name of the platform-specific directory as well as replicate the required resources and language projects within that directory. The ***$(PLATFORM_OS)*** mechanism described above is simpler to use, particularly if the number of platform-specific resources is small.

Using API not in OpenStep

Many new methods and classes were added for 4.0 and some new methods for 4.1 and 4.2, many outside the purview of the OpenStep specification. To restrict those that are defined to the methods and classes in OpenStep, add the flag **-DSTRICT_OPENSTEP** to the **OTHER_CFLAGS** variable in the project's **Makefile.preamble**.

Foundation Defect and Change List

This section documents many of the known problems with Foundation in OPENSTEP 4.0, 4.1, and 4.2, and the fixes and changes that have been made in OPENSTEP 4.1 and 4.2. Not all of the problems apply to the current release±refer to the *Releases* and *Disposition* fields in each note. Defects not present in the current release are of interest to those developing software on the current release with plans to deploy on a previous release. (However, we do not recommend that developers develop on a version newer than that planned for deployment.)

Reference: 36156

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSBundle **-localizedStringForKey:value:table:** caches default values

Description: NSBundle's **-localizedStringForKey:value:table:** method incorrectly caches the default value passed as the second parameter if the key isn't found in the table. On subsequent calls with the same key this first default value is returned. Sometimes, however, the default value may need to be different, even with the same key, in different locations in code.

Workaround: None

Reference: 41245

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSArray **-componentsJoinedByString:** extended

Description: NSArray **-componentsJoinedByString:** sends **-description** to each of the array's objects and concatenates the results of that message together, rather than using the objects directly. This means that this method will now operate on arrays which are composed of objects that are not NSStrings.

Workaround: None

Reference: 47347

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSArchiver and NSUnarchiver do not understand bitfields

Description: Structures containing bitfields cannot be archived and unarchived.

Workaround: Encode the members of a structure containing bitfields individually, encoding the bitfields as chars, shorts, or ints (or unsigned chars, shorts, or ints as appropriate).

Reference: 48447

Releases: OPENSTEP 4.0

Platforms: HP-UX

Disposition: Fixed in OPENSTEP 4.1

Problem: D.O. crashes sending some structures containing doubles

Description: D.O. would crash when attempting to send a parameter or return value which was a structure larger than 8 bytes and contained a double field.

Workaround: None

Reference: 49001

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Extended formatting for NSCalendarDates

Description: The number-formatting sequences for NSCalendarDates (%d, %e, %H, %I, %j, %m, %M, %S, %w, %y, %Y), during the construction of a description, can now contain formatting flags between the `%' and format code, to the extent that the string methods **+stringWithFormat:**, **-appendFormat:**, etc. understand them. Format flags are not understood during parsing from a format, however. Note that per 70220 [below], NSDateFormatters use their format string both to display the date and to parse a date from a string (which the user entered in a text field, for example), so this modest extension should not be used in the format string of NSDateFormatters. Setting the date formatter to allow natural language may obviate the problem (but may be undesirable for application reasons)

Workaround: None

Reference: 49084

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSInvocation always retains the return value

Description: NSInvocation always retains object return values. This can cause problems if the return value of a method is not a fully initialized object. Only **+alloc** and **+allocWithZone:** return uninitialized objects, so this is only a problem if a program uses invocations to send either of those methods to a class object. If an application contains its own methods which return objects which are not fully initialized, it should avoid sending those messages with an NSInvocation or over Distributed Objects.

Workaround: None

Reference: 50670

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSMethodSignature's **-getArgumentTypeAtIndex:** and **-methodReturnType** return too much information

Description: These NSMethodSignature methods return a string which contains the type descriptors for the requested argument plus all the type descriptors for the arguments following.

Workaround: Use **NSGetSizeAndAlignment()** to parse the first type. **NSGetSizeAndAlignment()** returns the first parameter pointer, advanced to the next type in the type string. Pass the return value of either of those methods as the first parameter to **NSGetSizeAndAlignment()** and pass NULL for the second and third parameters (or pass pointers to unsigned integers if either of those values is interesting); record the return value of that function in another variable. The difference between the return value and the first parameter is the length of the first type in the first parameter to **NSGetSizeAndAlignment()**. The single type string can then be **strncpy()**'d into its own buffer.

Reference: 51108

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: Kanji text in **NSLog()** messages shows as Unicode hex on Japanese systems

Description: Strings that contain non-ASCII characters are displayed with the non-ASCII characters translated to a hex representation (\uNNNN). This makes strings with Kanji essentially unreadable.

Workaround: None

Reference: 55579

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Collections that contain themselves don't unarchive well

Description: Unarchiving an object graph with a collection which contains itself, or contains an object

which archived a reference to the collection, may cause a program either to crash immediately when the archiving system attempts to retain it, or to crash later when a freed object is sent a message. The problem here is that the collection gets retained when the recursive reference is unarchived (which is normal) before it has completed its initialization in **-initWithCoder:**, which can be a problem for objects which keep track of their own retain count (as the collection classes do).

Workaround: None

Reference: 56330

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSConnection doesn't let go on NSRunLoop when it has nothing to do

Description: NSConnections do not register/un-register their ports with the run loop lazily, as they have something to serve. Specifically, an NSConnection with no root object and no local objects vended over the wire (no local proxies) should not have its ports registered with a run loop, since it can never receive any messages. This causes code like the following to never terminate:

```
id server = [NSConnection rootProxyForConnectionWithRegisteredName:@"name"
host:@"*"];
... do stuff with server ...
[server release];
... do stuff which adds a one-shot timer to the current run loop ...
... and run the run loop ...
[[NSRunLoop currentRunLoop] run];
/* NOTREACHED */
```

Workaround: None

Reference: 56644

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSArchiver tracks **char *** pointers as equivalent to their contents

Description: During archiving, NSArchiver keeps track of the **char *** strings it has seen by the address passed in for encoding. If a program uses the stack or **malloc()** to construct a **char ***, then encodes it using the "*" type primitive (or **@encode(char *)**), the archiver only remembers the stack/malloc address, assuming it has constant contents.

Workaround: Do not archive **char *** strings from the stack. Do not mutate or free malloc'd **char *** strings during archiving.

Reference: 56659

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: Proxies cannot be notification observers unless they are retained

Description: Because NSNotificationCenter does not retain observers, registering an observer with a remote NSNotificationCenter (the observer would be a proxy in the remote process) will eventually cause a crash of the remote process when it tries to message a freed object, unless the remote process explicitly retains the proxy. This situation is a bit arcane, but the same type of problem can happen with other objects that don't retain objects that they know about.

Workaround: Use an explicit protocol to tell the server (the process vending the notification center) to register and unregister an observer. The server then retains and registers the observer with the vended notification center, and unregisters and releases the observer when the client tells it to do so. (If an unregister mechanism is not part of the protocol, and used, the object will never be released.)

Reference: 57984

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: No way to get list of non-defaults command-line arguments

Description: In NEXTSTEP 3.3 (and earlier releases), command-line arguments that looked like defaults were removed from NXArgv. The array returned by **[[NSProcessInfo processInfo] arguments]**, NXArgv's replacement, however, contains all of the command-line arguments.

Workaround: Code that expected defaults options to have been stripped from NXArgv will have to be rewritten to skip default option names and their values, or the arguments array will need to be processed to remove them before using it. The **NSArgumentDomain** dictionary can be retrieved from the standard user defaults instance to find out what NSUserDefaults interpreted as a default option.

Reference: 58137

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: No way to create an NSData which does not free the bytes

Description: NSData and NSMutableData instances always take ownership of the byte pointer with which they are initialized, and then call **free()** to free the storage when they are deallocated. This applies to all existing creation and initialization methods. Only pointers to storage which can be passed to **free()** should be used to initialize data objects. As a corollary problem, there is no way to create a data object which refers to VM allocated storage.

Workaround: None

Reference: 59230

Releases: OPENSTEP 4.0, 4.1

Platforms: Mach, HP-UX, Solaris

Disposition: Fixed in OPENSTEP 4.2

Problem: The main bundle can be allocated with the wrong directory

Description: If an application is not launched with a full path, NSBundle attempts to create the main bundle instance with the current directory. If the application is launched from the command-line without a full path, and the application is not in the current directory (but found via the shell $PATH environment variable), the main bundle will be created with the wrong path. The Mach Workspace Manager always launches applications as a full path, so this is not a problem for applications launched with Workspace Manager.

Workaround: Launch the application with the full path, or launch the application from the directory in which it exists.

Reference: 61520

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Handler is called twice when deleting a directory

Description: NSFileManager's **-removeFileAtPath:handler:** method, when deleting a directory, calls the handler method **-fileManager:shouldProceedAfterError:** twice on each subdirectory or file error, and fails to call the handler for the top-level directory itself.

Workaround: None

Reference: 62019

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Character tables are not Unicode 2.0 compliant

Description: The tables used by NSCharacterSet and shipped as resources with the 4.0 Foundation are compliant with Unicode 1.1. The Unicode 2.0 tables were not yet available at the time of release.

Workaround: None

Reference: 62044

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Default uncaught exception handler changed

Description: The default uncaught exception handler now includes the name of the exception and a "numeric backtrace" in the logged message it generates. The backtrace consists only of the return addresses on the execution stack. Additionally, on Windows, an application exception is raised (via **RaiseException()**), which can be handled by a debugger.

Workaround: None

Reference: 62235

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: Can't search for characters case-insensitively

Description: NSString's **-rangeOfCharactersFromSet:** ignores the **NSCaseInsensitiveSearch** flag. This also affects NSScanner's **-scanCharactersFromSet:intoString:** method when the scanner has been set to be case insensitive.

Workaround: None

Reference: 62576

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSNumber **-compare:** results odd results with NaN

Description: Two NSNumbers containing with NaN (created, say, via **+numberWithDouble:**) compare NSOrderedSame and are **-isEqual:**. However, in C, NaN is *not* equal to NaN.

Workaround: None

Reference: 62812

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: Changed defaults don't appear sometimes to have taken affect

Description: The default NSUserDefaults domain search order places the NSArgumentDomain before the application's domain. This is usually desirable for reading defaults, but may not be if you set/change default values. If you set a value for a default that was passed as an option on the command-line, the value is correctly set/changed in the application's domain, but **-objectForKey:** will still return the value found in the NSArgumentDomain, because by default it is searched first.

Workaround: There are various workarounds depending upon the application and its use of user defaults. The following options are relative to a particular NSUserDefaults instance (that is, if you create multiple user defaults instances, you'd have to do the workaround for each individually).

1. Remove the NSArgumentDomain from the search list, perhaps after adding any keys it contains to the registration domain; command-line specified defaults will not be found, or not have priority over same-keyed defaults in the application's domain, however.
2. Move the NSArgumentDomain after the application's domain; command-line specified defaults will not have priority over same-keyed defaults in the application's domain, however.
3. Each time you set a default value, set it in the argument domain (difficult and not worth the trouble).
4. Best solution: Create a subclass of NSUserDefaults. For each defaults instance created, create a new volatile domain, tell the instance about it (**-setVolatileDomain:forName:**)

perhaps calling it "ChangedDefaults" (don't give it a name beginning with "NS"!), and insert it first in the domain search list of the new defaults instance. Override the **-setObject:forKey:** and **-removeObjectForKey:** methods to set/remove the value in the "ChangedDefaults" volatile domain, and then call **[super ...]** to do the same for the application domain. The following code illustrates what these override methods might look like:

```
- (void)setObject:(id)value forKey:(NSString *)defaultName {
    NSDictionary *changedDomain;
    id oldValue;
    [super setObject:value forKey:defaultName];
    changedDomain = [self volatileDomainForName:@"ChangedDefaults"];
    oldValue = [changedDomain objectForKey:defaultName];
    if (![oldValue isEqual:value]) {     // Something to do
        NSMutableDictionary *newDomain;
        newDomain = [[NSMutableDictionary allocWithZone:[self zone]]
                                          initWithDictionary:changedDomain];
        [newDomain setObject:value forKey:defaultName];
        [self setVolatileDomain:newDomain forName:@"ChangedDefaults"];
        [newDomain release];
    }
}

- (void)removeObjectForKey:(NSString *)defaultName {
    NSDictionary *changedDomain;
    id oldValue;
    [super removeObjectForKey:defaultName];
    changedDomain = [self volatileDomainForName:@"ChangedDefaults"];
    oldValue = [changedDomain objectForKey:defaultName];
    if (oldValue) {                      // Something to do
        NSMutableDictionary *newDomain;
        newDomain = [[NSMutableDictionary allocWithZone:[self zone]]
                                          initWithDictionary:changedDomain];
        [newDomain removeObjectForKey:defaultName];
        [self setVolatileDomain:newDomain forName:@"ChangedDefaults"];
        [newDomain release];
    }
}
```

Reference: 63974

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Time zone information not up-to-date

Description: The time zone information in Foundation's resources directory on Windows was quite a bit out-of-date. It has been updated for 4.2. The files contained in the directory and the way in which they are used has also changed internally in Foundation, so it is not possible to directly update a 4.1 deployment by copying the new resource files into the Foundation's resource directory. The Foundation now always uses its time zone data on all platforms, rather than any system data that may exist. For more information, see the section *Foundation's Time Zone Data* in the *General Comments* section of the Foundation release notes.

Workaround: None

Reference: 64605

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: HP-UX

Disposition: None

Problem: NSBundles cannot dynamically load code

Description: Unlike NeXT's implementations of OpenStep on other platforms, Foundation in PDO for HP-UX cannot dynamically load code. This is due to limitations in the compiler and Objective C runtime, which we hope to resolve in a future release.

Workaround: None

Reference: 66411

Releases: OPENSTEP 4.0, 4.1

Platforms: Solaris

Disposition: Fixed in OPENSTEP 4.2

Problem: Formatted floats sometimes loose sign

Description: Formatted strings created with the NSString methods like **+stringWithFormat:** lose the sign of floats formatted with **%f** (and **%g**) in some cases. For instance, `[NSString stringWithFormat:@"%f", -2.3e-2]` results in @"0.023000".

Workaround: Use **sprintf()** to create the character string with the desired format, then create the NSString from that with **+stringWithCString:**.

Reference: 66538

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSString **+stringWithFormat:** doesn't work with **%p** or some formatting flags

Description: Unlike what the documentation seems to say, NSString does **not** understand all ANSI C **printf()**-style formatting escapes and flags, and never has.

Workaround: Use **sprintf()** to create the character string with the desired format, then create the NSString from that with **+stringWithCString:**.

Reference: 66725

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: Workaround **CreateProcess()** bug in NSTask

Description: There is a bug in **CreateProcess()** on Windows NT 3.51, and possibly 4.0, such that when an environment is provided that is of size n*2048+1 bytes (n >= 1), the call fails with error ERROR_MORE_DATA. NSTask now works around this bug.

Workaround: None

Reference: 66766

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSBundle doesn't load _profile and _debug binaries in bundles

Description: NSBundle does not attempt to load code from debug or profile versions of their binary code, which the makefiles give the _debug and _profile suffixes, respectively. If the non-debug/non-profile binary doesn't exist in the bundle, no code will be loaded. NSBundle doesn't care how the binary has been compiled, it just only looks for the file with the name given by the **NSExecutable** key in the **Info.plist** file in the **Resources** directory.

Workaround: [Mach, Solaris, HP-UX]: Create a symbolic link named without the _profile or _debug suffix to the _profile or _debug binary at the top level of the bundle directory. [All platforms]: Move the _profile or _debug binary to a file of the same name without that suffix. [All platforms]: Modify the value of the **NSExecutable** key in the **Info.plist** file to include the desired _profile or _debug suffix.

Reference: 67201

Releases: OPENSTEP 4.0

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.1

Problem: Files mapped with **-initWithContentsOfMappedFile:** cannot be removed

Description: On Windows, you cannot remove a file that a process has mapped; this is a feature of the operating system. However, even after an NSData initialized with **-initWithContentsOfMappedFile:** is deallocated, a program may not be able to immediately remove the file. The problem is that an internal file-mapping object was autoreleased at the time the data was initialized.

Workaround: The autorelease pool which was active at that time must be released, and then the mapping will be undone, and the file can be removed. Surround the initialization of the NSData with

an autorelease pool so that when the data is deallocated, the mapping is immediately undone:

```
{
    id tmpPool = [[NSAutoreleasePool allocWithZone:NULL] init];
    mappedData = [[NSData allocWithZone:NULL]
initWithContentsOfMappedFile:@"/temp/foo"];
    [tmpPool release];
}
```

Reference: 67382

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: NSTask **-terminate** does not terminate child processes of the subprocess

Description: The **-terminate** method in NSTask does not recursively terminate any and all child processes of the launched process on Windows, as it does on other platforms.

Workaround: None

Reference: 68041

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSString's -**stringByStandardizingPath** raises an exception in certain circumstances

Description: When passed an absolute path that ends with ª..º, NSString's -**stringByStandardizingPath** method raises an exception.

Workaround: None.

Reference: 68155

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: Windows

Disposition: None

Problem: NSUserDefaults does not reflect Windows Registry case semantics

Description: The Windows Registry is case-preserving but case-insensitive with respect to keys (registry keys, not default keys). NSUserDefaults doesn't reflect this in its behavior, so, for example, if a persistent domain "MyApp" exists, and a persistent domain "MYAPP" is created, the new domain will eradicate the existing one and all defaults in the existing one will be lost. This does not affect default keys or values, only domain names.

Workaround: None

Reference: 68223

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: **-stringByAbbreviatingWithTildeInPath** sometimes incorrectly abbreviated

Description: If the first part of a path matched the user's home directory, '~' would be substituted for that prefix, regardless of whether or not the partial component at the end of the prefix was a complete component or not. For example,

```
    [@"/me-local/foo/bar" stringByAbbreviatingWithTildeInPath] =>
@"~/local/foo/bar"
```

if the example was executed by the "me" user whose account is "/me".

Workaround: None

Reference: 68251

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Various minor defects in the semantics of some path utility methods in corner cases

Description: **-stringByAppendingPathComponent:** doesn't compress extra slashes when receiver is the empty string

"" + "///x/y//z" -> "///x/y//z"

-stringByAppendingPathExtension: will append an extension beginning with a slash

"abc" + "/a" -> "abc./a"

-stringByAppendingPathExtension: will append an extension beginning with a drive

"abc" + "c:foo" -> "abc.c:foo"

-stringByAppendingPathExtension: does not compress extra slashes in the appended extension (although it is very unusual to do this)

"abc" + "de///fg////h" -> "abc.de///fg////h"

-stringByDeletingPathExtension does not compress extra slashes when there is no extension

"///a/b//c" -> "///a/b//c"

Workaround: None

Reference: 68447, 68781

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: D.O. and archiving type signature checks don't take structure tags into account

Description: The NeXT compiler does not include structure names in type encoding strings produced from typedefs of structures (for example, @encode(NSRange)); instead, the structure tag is given as "?". This is a bug in the compiler, but one that cannot be fixed for backwards compatibility reasons. To achieve compatibility with SunSoft's compiler (which correctly emits all structure tags), a "?" as a structure tag now acts as a "wildcard", matching anything. This is unlikely to affect user applications, but developers who construct their own type encoding strings should note this, as 4.1 is more permissive than OPENSTEP 4.0 was, a possible backwards-deployment issue.

Workaround: None

Reference: 68449

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: Windows

Disposition: None

Problem: Cannot get a proxy for a distributed OLE object using OPENSTEP D.O.

Description: When trying to get a proxy for an OLE object, you may get the following error: "deserializeObjectAt: class `NSDistantIDispatchProxy' not loaded".

Workaround: Include **nxorb.m** (from **NextDeveloper/Libraries**) in your client. Although the *D'OLE Developer's Guide* states that **nxorb.m** is only needed for use with NXConnections, it's needed in some circumstances for NSConnections as well.

Workaround: None

Reference: 68675

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: HP-UX

Disposition: None

Problem: Hang while looking up connection after fork

Description: If you have a DO client which tries to look up a connection using NSConnection's **connectionWithRegisteredName:host:**, then forks a process, and then tries to look up the connection again, the client will hang. If the fork is omitted, the client won't hang.

Workaround: None

Reference: 68709

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Deadlock between NSTask and NSNotificationQueue

Description: A process can sometimes deadlock in NSTask's **-waitUntilExit** method. The triggering conditions are not fully understood.

Workaround: None

Reference: 68790

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: **NSConnectionDidDieNotification** not sent in some circumstances

Description: If a client attempts to send a message to a server which had died before the client had processed the port death notification (which can take quite a while to arrive), an **NSInvalidSendPortException** is raised rather than the **NSConnectionDidDieNotification** being sent.

Workaround: None

Reference: 69019

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: byref ignored by NSCalendarDates

Description: The *byref* keyword is no longer ignored when transporting NSCalendarDates (as a parameter or return value) over D.O. Thus, a parameter or return value which is declared with *byref* now results in a proxy to the calendar date instance, rather than a real calendar date instance. By default, NSCalendarDates still go over the wire bycopy. NSDates always go over the wire bycopy.

Workaround: None

Reference: 69056

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: HP-UX

Disposition: None

Problem: Multi-threaded DO example occasionally hangs on HP-UX

Description: Compiling and running the multi-threaded DO example in **/NextDeveloper/Examples/Foundation/MultiThreadedDO** occasionally results in a hang.

Workaround: None

Reference: 69110

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: (unsigned char)45 and (long long)-1 are ordered ascending

Description: This is actually correct behavior, in that NSNumber follows the C type promotion rules. Thus, the (long long)-1 is promoted to (unsigned long long)18446744073709551615 and (unsigned char)45 is promoted to (unsigned long long)45, and then the comparison makes sense. This is, however, not all that desirable.

Workaround: None

Reference: 69152

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: HP-UX

Disposition: None

Problem: HP-UX nmserver has problems when passing large structures

Description: On HP-UX systems, passing large structures over D.O. without having first defined a protocol can cause the nmserver to become non-functional.

Workaround: Use protocols when passing large structures over D.O. on HP-UX.

Reference: 69174

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Lossy conversion to **NSASCIIStringEncoding** more lossy than intended

Description: When NSString finds a non-ASCII character during lossy conversion to **NSASCIIStringEncoding**, NSString converts it to an underscore. In 4.0, when NSString finds a non-ASCII character during such conversion, NSString converts the character and all the remaining characters in the string to underscores, even if they were ASCII.

Workaround: None

Reference: 69293

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Change to **+pathForResource:ofType:inDirectory:**

Description: The NSBundle class method **+pathForResource:ofType:inDirectory:** now first searches a **Resources** subdirectory of the third parameter if such a subdirectory exists. It used to examine only the directory specified in the third parameter.

Workaround: None

Reference: 69321

Releases: OPENSTEP 4.0

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.1

Problem: D.O. leaks every messaged object

Description: On Windows, every object that is messaged via D.O. is leaked due to an extraneous retain.

Workaround: None

Reference: 69471

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSPipe leaks file descriptors

Description: NSPipes and NSFileHandles do not take ownership of the file descriptor by default. If such an object was released without an explicit close of the descriptors, the descriptors were not recovered. The new methods **-initWithNativeHandle:closeOnDealloc:** and **-initWithFileDescriptor:closeOnDealloc:** allow for the creation of file handles which own the descriptors.

Workaround: None

Reference: 69926

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSNotificationCenter sometimes sent duplicate notifications

Description: In some situations, NSNotificationCenter sent observers two (or more) copies of a single notification. This only happened to observers which were registered to receive any notification name (perhaps with restriction to particular object).

Workaround: None

Reference: 70028

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Fault objects could not forward messages

Description: EOFault would cause an unrecognized selector exception rather than causing **-forwardInvocation:** to be invoked on the faulted object when the fault was sent a message that the faulted object did not understand (in other words, an object faulted would not get an opportunity for forward an unknown selector). Faulted objects which wish to forward a message which may cause them to be faulted should override the new NSObject class method **+instanceMethodSignatureForSelector:** to return an NSMethodSignature for the eventual target of the forwarded message. See the subsection *Forwarding Messages in OpenStep* in the *General Comments* section below.

Workaround: None

Reference: 70172

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSCalendarDate **-setCalendarFormat:** doesn't handle nil format properly

Description: NSDate's **-dateWithCalendarFormat:timeZone:** and NSCalendarDate's **-setCalendarFormat:** now have the documented behavior with respect to the first parameter being **nil**. In 4.0, they would simply set the format to **nil**, which resulted in the various **-description** methods returning an empty string.

Workaround: None

Reference: 70220

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSDateFormatter didn't convert input in the same way as output

Description: NSDateFormatters did not use the output format string to parse user-entered strings, so in most cases strings could not be entered in the same format in which they were displayed.

Now a date displayed with "%m/%y" (for example) can also be entered in that format.

Workaround: None

Reference: 70323

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSDateFormatter does not allow empty string

Description: Once the user had begun typing in a text field that had a date formatter, the user could not leave the field without entering a valid date; in particular, the user could not leave the field empty. Empty is now valid, and the date formatter produces **nil** for an empty string in its **-getObjectValue:forString:errorDescription:** method.

Workaround: None

Reference: 70334

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Time zones generated with wrong offset

Description: The **+timeZoneForSecondsFromGMT:** was incorrectly using the negative of the supplied parameter to generate the returned time zone. This also affected the parsing of dates where the time zone was specified with a raw GMT offset, like "-0700", but not where the time zone was a name, such as "GMT-0800" or "PST".

Workaround: None

Reference: 70440

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSRunLoop incorrectly computed whether or not there was work to be done

Description: In some situations using background monitoring of file handles, NSRunLoop would not take into account the file handles in deciding whether or not it had any more input sources.

Workaround: None

Reference: 70463

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: Cannot **NSLog()** large messages

Description: Due to Event Log limitations on NT 3.51, NSLog()'d messages greater than ~32K are dropped and do not appear in the Event Log. Also, messages greater than ~60K stopped appearing in the console for console based processes. In 4.2, messages longer than ~32000 bytes are truncated and the message:

```
[NSLog log message too long; remainder deleted]
```

is appended before being sent to the Event Log. Multiple writes are done to the console for large messages.

Workaround: None

Reference: 70565

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSNumberFormatter is not handling nil and NaN correctly

Description: If you tab into a cell with an NSNumberFormatter, and the cell has a nil object value, if you tab out of the cell, the number formatter inserts NaN into the cell. This value can be persistent and hard to get rid of, even when a valid number is typed into the field.

Workaround: None

Reference: 70580

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSNumber hashed long long numbers incorrectly

Description: NSNumber hashed long longs by simply casting the value to (unsigned int). This caused some equal numbers to have different hash values, breaking the hash±isEqual: invariant.

Workaround: None

Reference: 70612, 70718

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: NSDecimalNumber inaccuracies

Description: An error in the text from which the division algorithm was taken caused inaccuracies in the less significant end of the result of **-decimalNumberByDividingBy:withBehavior:** (in the fractional part, the ones, and tens digits, depending on the size of the numbers). Also, digits of precision were being lost during initialization on the less significant end of the number, and the decimal point was being incorrectly positioned in the result of **-decimalNumberBySubtracting:withBehavior:**.

Workaround: None

Reference: 70955

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Cannot append path components beginning with tilde

Description: NSString **-stringByAppendingPathComponent:** would refuse to append a component that began with `~' in 4.0. This was to prevent the accidental concatenation of "~*username*" onto a path, but `~' is a valid (but perhaps uncommon) first character of a file name, and that behavior was of marginal benefit, so it was eliminated.

Workaround: None

Reference: 70975

Releases: OPENSTEP 4.0, 4.1

Platforms: Mach

Disposition: Fixed in OPENSTEP 4.2

Problem: Loading bundles with duplicate or undefined symbols crashes applications

Description: In NEXTSTEP 3.3 and prior versions, attempting to load a bundle with symbols which duplicated already-existing symbols, or had undefined symbols which were still undefined after the bundle was loaded into the executable image, simply caused the bundle load to fail. In OPENSTEP, this causes an application to crash.

Workaround: By default, this is not fixed to restore the 3.3 behavior, since the fix is somewhat expensive. The 3.3 behavior can be enabled by setting the default **NSSafeBundleLoading** to YES in a user's defaults database. The success or failure of the operation can be tested with the NSBundle **-load** method.

Reference: 71118

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: Collections that contain themselves don't describe

Description: Sending the **-description** method to collections which contain themselves, or contain objects that contain the collection, causes infinite recursion.

Workaround: None

Reference: 71288

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Problem popping initial autorelease pool

Description: A program could crash if it repeatedly created and released the top autorelease pool.

Workaround: None

Reference: 71646

Releases: OPENSTEP 4.0

Platforms: All

Disposition: Fixed in OPENSTEP 4.1

Problem: Formatter.strings missing

Description: A missing strings file has been added to the Foundation, so in 4.1 error strings returned from the formatters (NSDateFormatter and NSNumberFormatter) will be localized.

Workaround: None

Reference: 71911

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSMutableData **-setData:** method does not reduce length of receiving data

Description: The NSMutableData **-setData:** method will increase the length of the receiving data object to accommodate all the bytes in the parameter, but does not shrink the size of the receiving data if the parameter contains fewer bytes than the receiver.

Workaround: Do an explicit **-setLength:** after **-setData:**.

Reference: 71962

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: NSBundle cannot identify the main bundle over Samba

Description: NSBundle is not able to identify the main bundle for applications launched from a directory mounted in Windows via Samba, where the Samba server has been configured to be case sensitive. Applications will terminate immediately on launch.

Workaround: Set the Samba server to be case insensitive (but this can cause other problems, such as files being inaccessible).

Reference: 72144

Releases: OPENSTEP 4.0, 4.1

Platforms: Mach

Disposition: Fixed in OPENSTEP 4.2

Problem: NSData's **-writeToFile:atomically:** always fails on DOS-formatted disks

Description: The temporary file that NSData creates when the second parameter to **-writeToFile:atomically:** is YES is not a valid file name on DOS-formatted disks (it's too long).

Workaround: None

Reference: 72464

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: Domains seem to be missing

Description: NSUserDefaults sometimes fails to identify the longest-named domain in the registry as a domain, and it will be omitted from lists, such as that produced by the command `defaults domains'.

Workaround: None

Reference: 72479

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSScanner's **-scanDouble:** and **-scanFloat:** too eager

Description: In cases where NSScanner encounters a lone 'E' or a lone '.', it currently puts 0.0 into the variable and returns YES. In some future release, it will probably return NO in such a situation.

Workaround: None

Reference: 73292

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSMutableDictionary **-setDictionary:** crash

Description: The **-setDictionary:** method of NSMutableDictionary will cause an application to crash when the parameter is an empty immutable dictionary.

Workaround: None

Reference: 73320

Releases: OPENSTEP 4.1, 4.2

Platforms: All

Disposition: None

Problem: In NSNumberFormatter, specifying different formats for negative and positive values causes problems

Description: When the formats you specify for positive and negative values aren't parallel in their treatment of separators, for example:

```
[numForm setFormat:@"###,##0.00;-(##0.00)"];
```

the positive format takes on the separator characteristics of the negative format. For example, given the above statement, positive values will be displayed without thousand separators even though their format includes separators.

Workaround: None

Reference: 73352

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: **-stringByResolvingSymlinksInPath** now does more

Description: On Windows, **-stringByResolvingSymlinksInPath** now fixes the cases of characters so that the case of the name in the returned string matches the on-disk case, and it also converts 8.3 names to the full names.

Workaround: None

Reference: 73686

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSFileHandle error notification not handled correctly

Description: If an error occurs in attempting a background monitoring operation, NSFileHandle may send an empty notification to observers. This may occur when a native file handle has not been

configured correctly for use, for example.

Workaround: None

Reference: 73689

Releases: OPENSTEP 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: NSCalendarDate won't archive correctly unless TZFILE is set

Description: In 4.1, NSTimeZone attempted to use information from the registry to choose the default time zone, so that the TZFILE environment variable, required in 4.0, was no longer necessary. However, the time zone object that was created in the absence of TZFILE was not transportable over Distributed Objects, and could not be unarchived from an object archive on any platform (so NSCalendarDates which use the default time zone could not be unarchived either).

Workaround: Set the TZFILE environment variable for a user to the name of a time zone, like "US/Eastern" or "Japan".

Reference: 73751

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Incorrect behavior when copying NSMutableDictionarys

Description: According to the OpenStep specification, when you send **-copy** or **-copyWithZone:** to an NSMutableDictionary you should get back a deep immutable copy. This was the behavior implemented in OPENSTEP 4.0. However, in OPENSTEP 4.1 (all platforms) what you get back is a shallow immutable copy--the objects in the dictionary are merely retained instead of being copied.

This change in behavior can have subtle effects within any code that copies

NSMutableDictionarys. For example, suppose you have a mutable dictionary populated with objects, and you create a copy (which should be a deep immutable copy). If you then modify one of the objects that you had put into the original dictionary, the modified object will appear in the copy of the dictionary as well as in the original. This bug means that a copy of a dictionary can change out from under you unexpectedly. In an application, you might see no change, incorrect operation, or a crash as a result. Because it doesn't always crash your applications, and because it manifests itself in a number of different ways, this bug can be particularly difficult to detect.

Workaround: While you may not be able to alter the way that libraries you link against are written, in your own code you can work around this bug simply by avoiding the use of either **-copy** or **-copyWithZone**: with NSMutableDictionarys. The easiest way to do this is to use the undocumented (in 4.1) **-initWithDictionary:copyItems:** method, as shown here:

```
[[NSDictionary allocWithZone:NULL] initWithDictionary:dict copyItems:YES]
```

This method only makes sense when you know you have a dictionary object, but that's often the case (**-isKindOfClass:** can be used for testing this). Note that **-initWithDictionary:copyItems:** isn't part of the OpenStep specification.

Reference: 73880

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: **NSFullUserName()** always returned **nil**

Description: Note that this DLL is not implemented on Windows 95, so this function will always return **nil** on Windows 95. Applications should be prepared for a **nil** return value.

Workaround: None

Reference: 74192

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSTimeZones are not equal

Description: Unless the receiver and parameter are the same identical object, NSTimeZone's **-isEqual:** method returns NO for identical time zone objects.

Workaround: None

Reference: 74198

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Misbehavior in initializing NSArrays

Description: In 4.1, the following code doesn't cause a nil-object exception, but it should.

```
id ids[5] = {[NSObject new], nil, nil, [NSObject new], [NSObject new]};
id a = [NSMutableArray arrayWithObjects:ids count:5];
```

Workaround: None

Reference: 74246

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Time zone offsets not parsed well

Description: The date can be misparsed from a string with a bare time zone offset (as opposed to a time zone name). For example:

```
dateWithNaturalLanguageString:@"1996-10-22 13:44:03 -0700"
```

returns the date

```
700-10-22 13:44:03 -0800
```

Workaround: None

Reference: 74271

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSUserDefaults **+initialize** causes crash on second invocation

Description: If NSUserDefaults' **+initialize** method was invoked a second time, say by the presence of a subclass, an application would crash.

Workaround: A subclass can provide an empty implementation of **+initialize** so that NSUserDefaults implementation does not get called multiple times.

Reference: 74349

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSAutoreleasePool crashes with a memory fault

Description: NSAutoreleasePool can crash with a memory violation during an autorelease if the following sequence of events occurs:

1. the non-top pools get a sum of exactly (2019 - 4 * P + 2048 * N) objects in them, where:
   N is any integer N >= 2,
   M is the maximum number of pools there have ever been at any instant, up to that point, and
   P = { 0 if M <= 16; ceil(M - 13) / 4) if M > 16 }
2. the top pool grows to be large (having more than 4096 objects in it), and is then released

Upon the next autorelease, the autorelease pool will shrink its storage, shrinking it to exactly the amount required for the (2019 - 4 * P + 2048 * N) objects in the remaining pools, and then try to add the newly autoreleased object one slot past the end of the new storage. The shrink in this case is not taking into account the extra storage that will immediately be needed.

Workaround: None

Reference: 74560

Releases: OPENSTEP 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: Prefix "\\" stripped by path utilities

Description: The path utilities go to a lot of trouble to preserve a prefix "\\" on Windows, which indicates a UNC file name. However, due to a bug, this was being stripped in one common code path.

Workaround: None

Reference: 74725

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSMutableDictionary **-initWithDictionary:copyItems:** crash

Description: NSMutableDictionary's **-initWithDictionary:copyItems:** method can crash a program on a nil dictionary parameter.

Workaround: None

Reference: 74787

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSNumberFormatter may not use the default attributes

Description: **-attributedStringForObjectValue:withDefaultAttributes:** doesn't always use the default text attributes passed in as the second parameter. Specifically, they weren't used when

formatting zero, nil, and NaN.

Workaround: None

Reference: 74791

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSFileHandle shouldn't raise if it doesn't write all the data

Description: The **-writeData:** method raises if it can't write all the data in one attempt.

Workaround: None

Reference: 74823

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: **NSWeekDayNameArray** is incorrect for French, German, Spanish, Swedish, & Italian.

Description: The **NSWeekDayNameArray** default in the French, German, Spanish, Swedish, & Italian language files in the Foundation resources start with Monday. The array should start with Sunday (even though Monday may be the conventional first-day-of-the-week).

Workaround: None

Reference: 74849

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSNumberFormatter's **-attributedStringForObjectValue:withDefaultAttributes:** doesn't work with nil second parameter

Description: NSNumberFormatter's **-attributedStringForObjectValue:withDefaultAttributes:** doesn't handle the case where nil is passed as the default attributes dictionary.

Workaround: None

Reference: 75011

Releases: OPENSTEP 4.0, 4.1

Platforms: Windows

Disposition: Fixed in OPENSTEP 4.2

Problem: Language preference in registry is inserted into NSLanguages default

Description: In 4.1, if the user did *not* have an NSLanguages default, NSUserDefaults looked into the Windows registry and created an NSLanguages default in the registration domain from the user's preferred language stored by the Control Panel. Now, the language from the registry is appended to the list if it doesn't already occur in the NSLanguages default (with the default being created in the registration domain as before if it doesn't exist). Also, whenever the language "English" is not present in the list, it is now appended to the list.

Workaround: None

Reference: 75359

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSData **-hash** algorithm is too simple

Description: NSData used to return the length of the data as its hash value. For EOF, where fixed-length data objects may be used as primary keys, this results in terrible hashing performance. NSData now uses a better algorithm that depends on the byte contents of the data.

Workaround: None

Reference: 75568

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSCalendarDate %F format flag truncates rather than rounds

Description: The %F format flag for NSCalendarDates indicates where milliseconds are to be indicated in a formatted string. However, it was truncating a series of nines in the result, rather than rounding. For example, if the millisecond value was 120, it might be stored as .119999999999..., and the formatting for %F would truncate the excess fractional part and result in "119" in the formatted string.

Workaround: None

Reference: 75881

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSNumberFormatter can use wrong formats

Description: If an NSNumberFomatter is asked to format a number after the positive format or negative format has been changed from the default, but before the **-positiveFormat** or **-negativeFormat** methods have been called (such as by **-getObjectValue:forString:errorDescription:**), the number formatter will use the previous positive or negative format(s) rather than the new ones.

Workaround: Call **-positiveFormat** after setting the positive format or **-negativeFormat** after setting the negative format.

Reference: 75947

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Problems with NSDecimalNumber addition and subtraction

Description: When two NSDecimalNumber objects have a large difference in their precision (for example, 1070464.1 and 10458008.858880000582251216050000055879), the numbers can be added or subtracted incorrectly.

Workaround: None

Reference: 76051

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Path utilities don't handle multi-byte strings

Description: The path utility methods can return corrupted strings for receiving strings which have multi-byte characters.

Workaround: None

Reference: 76181

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSValues and NSNumbers in 3.3 archives can't be unarchived

Description: In the 3.3 Foundation, NSValues and NSNumbers were archived in such a way that a helper class, NSValueDecoder, was needed to unarchive them. This class does not exist in the Foundation in 4.0 or 4.1, so such archives cannot be read in.

Workaround: None

Reference: 76716

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Date parsing not handling 1-digit numbers well

Description: If an application parsed the string "96/5" with the format "%y/%m", the application would raise an out-of-bounds exception. If the string was "96/05", things worked fine.

Workaround: None

Reference: 77004

Releases: OPENSTEP 4.0, 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: Attempting to transport or unarchive the local time zone object fails

Description: The object returned by [NSTimeZone localTimeZone] cannot be unarchived if archived, nor transported over the wire with Distributed Objects.

Workaround: None

Reference: 77079

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSArray methods not working with fault parameters

Description: The NSArray methods **-indexOfObject:**, **-indexOfObject:inRange:**, and **-containsObject:** cause a raise of a selector-not-recognized exception when the first parameter is a faulted object. The **-firstObjectCommonWithArray:** has the same problem when the array parameter is an array of faults.

Workaround: None

Reference: 77202

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: NSFileManager methods don't work on strict 8.3 DOS file systems

Description: The NSFileManager methods **-createFileAtPath:contents:attributes:** and **-movePath:toPath:handler:** can fail on FAT file systems which are strict about 8.3 file names, such as available on Mach. The FAT file systems of Windows NT and Windows 95 are more forgiving.

Workaround: None

Reference: 77309

Releases: OPENSTEP 4.0, 4.1, 4.2

Platforms: All

Disposition: None

Problem: Local proxies not immediately released when remote proxy released

Description: For performance reasons, Distributed Objects batches, on the remote side, release methods destined for local proxies. Whenever a synchronous method (i.e., non-*oneway*) is sent to a remote object, all pending releases on local objects are returned with the return value of the method (which may be void of course). Since the local proxy retains the object it represents (part of its function is to make sure the local object which has been vended remains valid at least as long as there is an outstanding remote proxy for it), a local object which has been vended over the wire, but released on the server side, will persist until the client sends a synchronous message to the server. If a client and server communicate only with *oneway* messages, local objects which are sent remotely by proxy (as a parameter for example) will persist "forever", or until the local connection is invalidated. This can be a problem for long-lived clients which maintain a persistent connection to a server.

Workaround: Occasionally send a synchronous message from the client to the server. The message does not have to have a return value or parameters, or do anything, but will serve to flush pended releases back to the client. Another workaround would be to occasionally invalidate the

connection to the server and reconnect.

Reference: 77559

Releases: OPENSTEP 4.2

Platforms: Mach, HP-UX, Solaris

Disposition: None

Problem: Process name is empty string

Description: If argv[0] of a process is not an absolute path, and the Foundation can't find argv[0] by searching the PATH environment variable to compute the absolute path, the process name computation may result in the empty string. This can only happen if a process uses **fork()** and **execve()** (or related function) to create a Foundation-based process, and the first argument to **execve()** specifies an executabe which does not exist in the PATH list and the argv[0] passed into **execve()** is not that same full path, but a relative one. There isn't anything the Foundation can do in this case about computing the full path to the executable, for use in computing the main bundle for example. However, the process name should not be empty string, but rather the last component of argv[0] unchanged. An empty string for the process name causes trouble for other Foundation subsystems, like user defaults.

Workaround: Pass the same string in argv[0] as the first argument, to **execve()** or related function.

Reference: 77678

Releases: OPENSTEP 4.1

Platforms: All

Disposition: Fixed in OPENSTEP 4.2

Problem: NSNumber's **-decimalValue** method broken for non-English usage

Description: The **-decimalValue** method of NSNumber incorrectly computes its result for users whose defaults specify a decimal point other than period (`.'). This may cause NSNumberFormatters to not accept any input other than "2" in a text field. This affects the scale field in the AppKit's print panel in 4.1, for example.

Workaround: None