3.3 Release Notes: Objective€C Run-Time System and Common Classes and Functions

This file contains release notes for the 3.2, 3.1, and 3.0 releases of the Objective C run-time system, and also of the common classes and functions. There were no changes for Release 3.3, 3.2, or 3.1.

Notes Specific to Release 3.0

This section contains information about the following topics:

- New features
- Implementation changes
- Incompatible changes
- Organizational changes

New Features

New Protocol and NXBundle Classes

Protocol objects are used by the run-time system to keep track of formal protocols employed by the program. Documentation for this new class can be found in:

/NextLibrary/Documentation/NextDev/GeneralRef/15_RunTime/Classes/Protocol.rtf

NXBundle objects manage bundles of resources required by an application. The NXBundle class is documented in:

/NextLibrary/Documentation/NextDev/GeneralRef/03_Common/Classes/NXBundle.rtf

New Run-Time Functions

The function **objc_lookUpClass()** has been added to the Objective C run-time functions. Just like **objc_getClass()**, this function returns the class for a

given name. However, if the class is not present, **objc_lookUpClass()** returns **nil** rather than considering this to be an error condition.

The function **sel_registerName()** has also been added to allow new selectors to be registered with the Objective C run-time system. Like **sel_getUid()**, this function gets the selector associated with a given name. However, if there is no selector for that name already present in the system, **sel_registerName()** allocates a new selector rather than returning **nil**.

Making Objective C Thread Safe

The Objective C run-time system has been made safe for use in multi-threaded programs. Since complete thread-safety requires that a lock be acquired every time a message is sent (which increases the time required to send a message by a factor of approximately three), the thread safety features must be explicitly enabled using the new function **objc_setMultithreaded()**.

Note: The Application Kit and other NEXTSTEP kits don't make use of this Objective-C feature; thus kit classes (and their subclasses) can't be used outside the main thread of execution.

Thread-Safe Exception Handling

The exception handling system provided in `objc/error.h` has been made unconditionally thread safe.

Debugging Interface

The Object class now implements a method to assist in debugging when using **gdb** (see **Debugger.rtf** for information on how to use this feature from **gdb**). The **printForDebugger:** method in the Object class prints the name of the receiver's class and the address of the receiver. Classes should override this method to print a more useful description of themselves for debugging purposes.

Improved Detection of Messaging Freed Objects

The Objective C run-time system can now detect messages which are sent to objects which have been freed. An error will be reported when this problem is detected. Note that this detection is possible only in the interval between when the object is freed and when the freed space is reused for some new purpose.

Implementation Changes

Copy Methods

In release 2.0, Object's **copy** and **copyFromZone:** methods were implemented independently. In order to support copying, an object had to implement both methods. In release 3.0, Object's **copy** method has been changed to call **copyFromZone:** from the object's zone:

```
- copy
{
  return [self copyFromZone: [self zone]];
}
```

With this change, an object need only implement the **copyFromZone:** method to support copying using either method.

The **copy** methods of the List, Storage and HashTable classes have been removed since they were identical to the **copy** method which they now inherit from Object.

New Objective C Keyword

The release 3.0 Objective C compiler implements the type **id** as a keyword of the language. For this reason, `objc/objc.h` no longer provides a typedef for **id** when compiling Objective C programs. However, in order to allow `objc/objc.h` to be included by C programs without parse errors,

`objc/objc.h` does still provide a typedef for **id** when the macro **__OBJC__** is not predefined by the compiler.

Incompatible Changes

Renamed Methods

The following methods in the Object class were renamed:

```
+ superClass
- superClass
- isKindOfGivenName:
- isMemberOfGivenName:
- methodDescFor:
- instanceMethodDescFor:
```

The new names are:

```
+ superclass
- superclass
- isKindOfClassNamed:
- isMemberOfClassNamed:
- descriptionForMethod:
- descriptionForInstanceMethod:
```

These new names conform to standard NEXTSTEP spelling conventions and

are more descriptive of the method's function.

The following methods in the Storage class were renamed:

- `insert:(void *)anElement at:(unsigned)index`
- `replace:(void *)anElement at:(unsigned)index`
- `removeAt:(unsigned)index`

The new names are:

- `insertElement:(void *)anElement at:(unsigned)index`
- `replaceElementAt:(unsigned)index with:(void *)anElement`
- `removeElementAt:(unsigned)index`

These new names are more compatible with the corresponding methods in the List class:

- `insertObject:anObject at:(unsigned)index`
- `replaceObjectAt:(unsigned)index with:newObject`
- `removeObjectAt:(unsigned)index`

Existing programs which use the old methods will continue to work correctly, and source code will still compile (although with warnings).

Renamed Functions

The functions **class_addInstanceMethods()** and **class_addClassMethods()** have been replaced by the single function **class_addMethods()**, which better matches the single function **class_removeMethods()**.

Objective C Type Encoding Strings

The Objective C type encodings have been expanded to contain additional information for Distributed Objects. One effect of these changes is that the type strings produced by **@encode** are no longer compatible with typed streams when the encoded type is a struct. The new encoding for `struct foo {int x, y;};` is `"{foo=ii}"`, which is not recognized by a typed stream. It expects simply `"{ii}"`.

Known Problems

The data allocated by **NXAllocErrorData()** may be invalidated by subsequent calls to **NXAllocErrorData()**. Do not rely on this data remaining valid across multiple calls. It is safe to use **NXAllocErrorData()** so long as the data will only be used prior to the next call to **NXAllocErrorData()**. If you need more complex allocation patterns, you should use **malloc()** to allocate the data and implement your own mechanism to free this data. One possibility is to register the data to be freed in your main event loop.

Organizational Changes

The header file `zone.h` has been moved to `objc/zone.h`.