Objective-C support in GCC 4.6

GNUSTEP www.gnustep.org

Nicola Pero, Fosdem 2011

History: GCC and "Objective-C 1.0"

The NeXT era

- Objective-C compiler originally donated by NeXT (1992)
- GNU Objective-C runtime (1993)
- GNUstep project started (1994)

The Fall of NeXT

- Objective-C slowly drifted into obscurity over the years
- GCC Objective-C compiler started to fall apart (2000)
- GCC Objective-C testsuite started

The golden Apple years (2001-2005)

- Resurgence of Apple interest in Objective-C (2001)
- Apple kept improving the GCC Objective-C compiler
- Apple contributed changes back to GCC mainline

History: GCC and "Objective-C 2.0"

Apple develops "Objective-C 2.0" (2005-2009)

- Apple starts working on new language features on their own fork (2005)
- Apple stops contributing changes to mainline GCC (2006)
- Mainline GCC Objective-C compiler remains perfectly functional, but gets no improvements or new features (2006-2009)

GCC 4.6 (2010-2011)

- New impetus to bring the new Objective-C feature to mainline GCC
- Chris Lattner @ Apple states that Apple GCC Objective-C 2.0 changes can not be merged back into mainline GCC because "Apple does not have an internal process to assign code to the FSF anymore" (September 2010)
- A handful of Apple changes up to early 2006 could be merged anyhow because they had already been donated by Apple, just not merged from one branch to the other (September – October 2010)
- "Objective-C 2.0" features are then implemented from scratch by Nicola and Iain in mainline GCC (September – December 2010)
- GCC 4.6 will be released in Spring 2011

GCC 4.6

Objective-C compiler improvements and bug fixes:

- fixed about 50 bugs between objc compiler, objc++ compiler and libobjc
- a better compiler even for "traditional" Objective-C

Objective-C 2.0:

- Syntax changes (declared and synthesized properties, dot syntax, fast enumeration, optional protocol methods, method/protocol/class attributes, class extensions)
- No ABI changes (no non-fragile instance variables, no exposure of properties and protocol optional methods to runtime)

GNU Objective-C runtime:

- Runtime support for the new Objective-C 2.0 features
- New Apple-like API

Objective-C 2.0: dot-syntax

An alternative syntax for using getters and setters.

```
@interface MyClass
- (int) count;
- (void) setCount: (int)value;
@end

void function (MyClass *object)
{
  /* The following means "[object setCount: 40]" */
  object.count = 40;

 /* The following means "if ([object count] != 40)" */
  if (object.count != 40)
    abort ();
}
```

Objective-C 2.0: dot-syntax for classes

An alternative syntax for using class getters and setters

```
@interface MyClass
+ (int) count;
+(void) setCount: (int)value;
@end

void function (void)
{
  /* The following means "[MyClass setCount: 40]" */
  MyClass.count = 40;

 /* The following means "if ([MyClass count] != 40)" */
  if (MyClass.count != 40)
    abort ();
}
```

Objective-C 2.0: dot-syntax

Confusing not only for users, but also for the compiler. The code produced by “object.count” heavily depends on context.

```
 if (object.count++) { … }
```

is equivalent to something like

```
if ({int x, x = [object count], [object setCount: (x + 1)], x}) { … }
```

Objective-C 2.0: declared properties

@property originates as a different way of declaring a setter and getter

```
@interface MyClass
{
  int x;
}
@property int a;
@end

@implementation MyClass
- (int) a
{
  return x;
}
- (void) setA: (int)x
{
  x = a;
}
@end
```

Objective-C 2.0: declared properties

@property can have property attributes

```
@interface MyClass
{}
@property (copy) id a;
@property (retain) id b;
@property (setter = writeA:, nonatomic) int a;
@end
```

Objective-C 2.0: synthesized properties

@synthesize automatically generates @property getters/setters as appropriate

```
@interface MyClass
{
  id x;
  int b;
}
@property (copy) id x;
@property (setter = writeA:, nonatomic) int a;
@end

@implementation MyClass
@synthesize x;
@synthesize a = b;
@end
```

Objective-C 2.0: dynamic properties

@dynamic disables all warnings about unimplemented property setters/getters

```
@interface MyClass
{ }
@property int x;
@end

@implementation MyClass
@dynamic x;
- (void) test
{
   self.x = 44; /* No warnings, as if the method -setX: actually existed.  */
}
@end
```

Objective-C 2.0: properties

The Objective-C 2.0 “specification” only describes the simple cases.

More complicated cases have behaviour that is unspecified. Apple implemented a number of ad-hoc behaviours which were carefully replicated in GCC 4.6.

The complicated cases arise by the combination of:

- readonly properties (only have a setter)
- properties declared readonly in public but with a private setter
- properties inherited from protocols, potentially marked as @optional
- properties added or overridden by categories
- properties overridden in subclasses, sometimes with different types
- properties with types that do not match the instance variable type
- properties with a mixture of hand-written and @synthesize setter/getter implementations

Objective-C 2.0: fast enumeration

A new syntax to enumerate objects in collections. Compiles to fairly efficient code that requests objects in batches.

```
void function (NSArray *array)
{
  NSObject *object;
  for (object in array)
    {
        /* do something with object */
    }

  /* c99-style syntax  */
  for (NSObject *object in array)
     {
        /* do something with object */
     }
}
```

Objective-C 2.0: @optional protocol methods

A new syntax to mark some protocol methods are optional. Intended to replace informal protocols.

```
@protocol Delegate
- (int)requiredMethod;
@optional
- (int)optionalMethod;
@required
-(int)requiredMethod2;
@end
```

Objective-C 2.0: deprecation attributes

Ability to mark classes, protocols and methods as deprecated.

A compiler warning is generated if they are used.

```
__attribute__ ((deprecated))
@interface MyClass
@end

__attribute__ ((deprecated))
@protocol MyProtocol
@end

@interface MyClass2
- (void) method __attribute__((deprecated));
@end
```

Objective-C 2.0: unused attribute

Ability to mark method arguments as unused.

No compiler warning is generated if the argument is not used.

```
@interface MyClass
- (id) method: (id) __attribute__((unused)) argument;
@end

@implementation MyClass
- (id) method: (id) __attribute__((unused)) argument
{
  return nil; /* no warning that 'argument' is unused.  */
}
@end
```

Objective-C 2.0: method attributes

Ability to mark methods with deprecated, sentinel, noreturn and format attributes.

```
@interface NSArray
{ }
+ (id) arrayWithObjects: (id)firstObject, ... __attribute__ ((sentinel));
@end

void test (id object)
{
  NSArray *array;

  array = [NSArray arrayWithObjects: object, object, nil]; /* Ok */
  array = [NSArray arrayWithObjects: object, object]; /* warning:  "missing sentinel" */
}
```

Objective-C 2.0: class extensions

A category with no name. Methods are added directly to the main interface, and the compiler checks that they are implemented in the main implementation.

```
/* Public header (MyClass.h)  */
@interface MyClass
- (void) test1;
@end
```

```
/* Private implementation (MyClass.m) */
@interface MyClass ()
- (int) test2;
@end

@implementation MyClass
- (int) test1 { return; }
- (int) test2 { return; }
@end
```

Controlling the language dialect

GCC 4.6 default

Objective-C 2.0 syntax, but without exception and synchronization syntax (@try, @catch, @throw, @finally, @synchronized)

-fobjc-exceptions

Enable Objective-C exception and synchronization syntax

-fobjc-std=objc1

Conform to the Objective-C 1.0 language as implemented by GCC 4.0

GNUstep-make default

-fobjc-exceptions: Objective-C 2.0 syntax and exception and synchronization syntax.

GNU Objective-C Runtime: New API

#include <objc/objc.h>

basic Objective-C header file, defining basic Objective-C types such as id, Class and BOOL.

#include <objc/runtime.h>

All of the Objective-C runtime API, almost identical to the Apple/NeXT Objective-C runtime API. All functions and types documented extensively in the header file.

#define __GNU_LIBOBJC__ 20100911

Allow you to recognize if the GNU Objective-C runtime is being used, and which version.

GCC 4.6

GCC 4.6 is in **stage 4** ("open just for regression bugfixes and documentation fixes").

So, no more changes. But please try it out!

It will be released "when ready" - sometime in the next few months.

You need **gnustep-base and gnustep-gui from trunk** to use it

- A new GNUstep release will be available before GCC 4.6 is finally released.

You can also use it on Apple (Objective-C 32-bit only).

Thank you!

For more information

http://gcc.gnu.org