XCode ⁞ Build Settings

For Mac OS X / iOS

Linking

1. Dead Code Stripping

DEAD_CODE_STRIPPING = YES

Activating this setting causes the -dead_strip flag to be passed to ld(1) via cc(1) to turn on dead code stripping. If this option is selected, -gfull (not -gused) must be used to generate debugging symbols in order to have them correctly stripped.

[DEAD_CODE_STRIPPING, -dead_strip]

2. Link With Standard Libraries

LINK_WITH_STANDARD_LIBRARIES = YES

If this setting activated, then the compiler driver will automatically pass its standard libraries to the linker to use during linking. If desired, this flag can be used to disable linking with the standard libraries, and then individual libraries can be passed as Other Linker Flags.

[LINK_WITH_STANDARD_LIBRARIES, -nostdlib]

Packaging

1. Info.plist Output Encoding

INFOPLIST_OUTPUT_FORMAT = binary

Specifies the output encoding for the output Info.plist (by default, the output encoding will be unchanged from the input). The output endcodings can be 'binary' or 'XML'.

[INFOPLIST_OUTPUT_FORMAT]

GCC 4.2 - Code Generation

1. Generate Debug Symbols

GCC_GENERATE_DEBUGGING_SYMBOLS = NO
(*Permitted for «Debug» configuration)*

Enables or disables generation of debug symbols. When debug symbols are enabled, the level of detail can be controlled by the build 'Level of Debug Symbols' setting.

[GCC_GENERATE_DEBUGGING_SYMBOLS]

2. Generate Position-Dependant Code

GCC_DYNAMIC_NO_PIC = YES
(*No permitted for «Debug» configuration, or for shared libraries)*

Faster function calls for applications. Not appropriate for shared libraries (which need to be position-independent).

[GCC_DYNAMIC_NO_PIC, -mdynamic-no-pic]

3. Inline Methods Hidden

GCC_INLINES_ARE_PRIVATE_EXTERN = YES

When enabled, out-of-line copies of inline methods are declared 'private extern'.

[GCC_INLINES_ARE_PRIVATE_EXTERN, -fvisibility-inlines-hidden]

4. Make Strings Read-Only

GCC_REUSE_STRINGS = YES

Reuse string literals.

[GCC_REUSE_STRINGS, -fwritable-strings]

5. Objective-C Garbage Collection

GCC_ENABLE_OBJC_GC = Unsupported

Compiles code to use Garbage Collector write-barrier assignment primitives within the Objective-C runtime. Code is marked as being GC capable. An application marked GC capable will be started by the runtime with Garbage Collection enabled. All Objective-C code linked or loaded by this application must also be GC capable. Code compiled as GC Required is presumed to not use traditional Cocoa retain/release methods and may not be loaded into an application that is not running with Garbage Collection enabled. Code compiled as GC Supported is presumed to also contain traditional retain/release method logic and can be loaded into any application. Garbage Collection is only supported on Mac OS X 10.5 and later.

[GCC_ENABLE_OBJC_GC, -fobjc-gc | -fobjc-gc-only]

6. Optimization Level

GCC_OPTIMIZATION_LEVEL = Fastest, Smallest [-Os]

None: Do not optimize. [-O0]
With this setting, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Fast: Optimizing compilation takes somewhat more time, and a lot more memory for a large function. [-O, -O1]
With this setting, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time. In Apple's compiler, strict aliasing, block reordering, and inter-block scheduling are disabled by default when optimizing.

Faster: The compiler performs nearly all supported optimizations that do not involve a space-speed tradeoff. [-O2]
With this setting, the compiler does not perform loop unrolling or function inlining, or register renaming. As compared to the 'Fast' setting, this setting increases both compilation time and the performance of the generated code.

Fastest: Turns on all optimizations specified by the 'Faster' setting and also turns on function inlining and register renaming options. This setting may result in a larger binary. [-O3]

Fastest, smallest: Optimize for size. This setting enables all 'Faster' optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size. [-Os]

[GCC_OPTIMIZATION_LEVEL]

GCC 4.2 - Language

1. C Language Dialect

GCC_C_LANGUAGE_STANDARD = C89
(C99 permitted if absolutely necessary, especially on iOS)

Choose a standard or non-standard C language dialect.

ANSI C: Accept ISO C90 and ISO C++, turning off GNU extensions that are incompatible. [-ansi]
Incompatible GNU extensions include the 'asm', 'inline', and 'typeof' keywords
(but not the equivalent __asm__, __inline__, and __typeof__ forms), and the '//' syntax for comments.
This setting also enables trigraphs.
C89: Accept ISO C90, but not GNU extensions. [-std=c89]
GNU89: Accept ISO C90 and GNU extensions. [-std=gnu89]
C99: Accept ISO C99, but not GNU extensions. [-std=c99]
GNU99: Accept ISO C99 and GNU extensions. [-std=gnu99]

Compiler Default: Tells the compiler to use its default C language dialect. This is normally the best choice unless you have specific needs. (Currently equivalent to GNU89.)

Please see the full GCC manual for the full definition of all these settings on the C dialect:
http://developer.apple.com/documentation/DeveloperTools/gcc-4.2.1/gcc/C-Dialect-Options.html

[GCC_C_LANGUAGE_STANDARD]

GCC 4.2 - Warnings

1. Check Switch Statements

GCC_WARN_CHECK_SWITCH_STATEMENTS = YES

Warn whenever a switch statement has an index of enumeral type and lacks a case for one or more of the named codes of that enumeration. The presence of a default label prevents this warning. Case labels outside the enumeration range also provoke warnings when this option is used.

[GCC_WARN_CHECK_SWITCH_STATEMENTS, -Wswitch]

2. Effective C++ Violations

GCC_WARN_EFFECTIVE_CPLUSPLUS_VIOLATIONS = YES

Warn about violations of the following style guidelines from Scott Meyers' Effective C++ book:

- Item 11: Define a copy constructor and an assignment operator for classes with dynamically allocated memory.
- Item 12: Prefer initialization to assignment in constructors.
- Item 14: Make destructors virtual in base classes.
- Item 15: Have operator= return a reference to *this .
- Item 23: Don't try to return a reference when you must return an object.

and about violations of the following style guidelines from Scott Meyers' More Effective C++ book:

- Item 6: Distinguish between prefix and postfix forms of increment and decrement operators.
- Item 7: Never overload &&, ||, or, .

If you use this option, you should be aware that the standard library headers do not obey all of these guidelines; you can use grep -v to filter out those warnings.

[GCC_WARN_EFFECTIVE_CPLUSPLUS_VIOLATIONS, -Weffc++]

3. Four Character Literals

GCC_WARN_FOUR_CHARACTER_CONSTANTS = YES

Warn about four-char literals (e.g., MacOS-style OSTypes: 'APPL').

[GCC_WARN_FOUR_CHARACTER_CONSTANTS, -Wfour-char-constants]

4. Global Construction or Destruction Required

GCC_WARN_ABOUT_GLOBAL_CONSTRUCTORS = YES

Warn about namespace scope data that requires construction or destruction, or functions that use the constructor attribute or the destructor attribute. Additionally warn if the Objective-C GNU runtime is used to initialize various metadata.

[GCC_WARN_ABOUT_GLOBAL_CONSTRUCTORS, -Wglobal-constructors]

5. Hidden Local Variables

GCC_WARN_SHADOW = YES

Warn whenever a local variable shadows another local variable, parameter or global variable or whenever a built-in function is shadowed.

[GCC_WARN_SHADOW, -Wshadow]

6. Implicit Conversion to 32 Bit Type

GCC_WARN_64_TO_32_BIT_CONVERSION = YES

Warn if a value is implicitly converted from a 64 bit type to a 32 bit type.

[GCC_WARN_64_TO_32_BIT_CONVERSION, -Wshorten-64-to-32]

7. Incomplete Objective-C Protocols

GCC_WARN_ALLOW_INCOMPLETE_PROTOCOL = YES

Warn if methods required by a protocol are not implemented in the class adopting it. Only applies to Objective-C and Objective-C++.

[GCC_WARN_ALLOW_INCOMPLETE_PROTOCOL, -Wno-protocol]

8. Inhibit All Warnings

GCC_WARN_INHIBIT_ALL_WARNINGS = NO

Inhibit all warning messages.

[GCC_WARN_INHIBIT_ALL_WARNINGS, -w]

9. Initializer Not Fully Bracketed

GCC_WARN_INITIALIZER_NOT_FULLY_BRACKETED = YES

Warn if an aggregate or union initializer is not fully bracketed.
Example, Here initializer for a is not fully bracketed, but that for b is fully bracketed.

```
int a[ 2 ][ 2 ] = { 0, 1, 2, 3 };
int b[ 2 ][ 2 ] = { { 0, 1 }, { 2, 3 } };
```

[GCC_WARN_INITIALIZER_NOT_FULLY_BRACKETED, -Wmissing-braces]

10. Mismatched Return Type

GCC_WARN_ABOUT_RETURN_TYPE = YES

Causes warnings to be emitted when a function with a defined return type (not void) contains a return statement without a return-value. Also emits a warning when a function is defined without specifying a return type.

[GCC_WARN_ABOUT_RETURN_TYPE, -Wreturn-type]

11. Missing Braces and Parentheses

GCC_WARN_MISSING_PARENTHESES = YES

Warn if parentheses are omitted in certain contexts, such as when there is an assignment in a context where a truth value is expected, or when operators are nested whose precedence people often get confused about.

Also warn about constructions where there may be confusion to which if statement an else branch belongs. Here is an example of such a case:

```
{
   if( a )
      if( b )
         foo();
      else
         bar();
}
```

In C, every else branch belongs to the innermost possible if statement, which in this example is if (b) . This is often not what the programmer expected, as illustrated in the above example by indentation the programmer chose. When there is the potential for this confusion, GCC will issue a warning when this flag is specified. To eliminate the warning, add explicit braces around the innermost if statement so there is no way the else could belong to the enclosing if . The resulting code would look like this:

```
{
   if( a )
   {
      if( b )
         foo();
      else
         bar();
   }
}
```

[GCC_WARN_MISSING_PARENTHESES, -Wparentheses]

12. Missing Fields in Structure Initializers

GCC_WARN_ABOUT_MISSING_FIELD_INITIALIZERS = YES

Warn if a structure's initializer has some fields missing. For example, the following code would cause such a warning, because "x.h" is implicitly zero:

```
struct s { int f, g, h; };
struct s x = { 3, 4 };
```

This option does not warn about designated initializers, so the following modification would not trigger a warning:

```
struct s { int f, g, h; };
struct s x = { .f = 3, .g = 4 };
```

[GCC_WARN_ABOUT_MISSING_FIELD_INITIALIZERS, -Wmissing-field-initializers]

13. Missing Function Prototypes

GCC_WARN_ABOUT_MISSING_PROTOTYPES = YES

Causes warnings to be emitted about missing prototypes.

[GCC_WARN_ABOUT_MISSING_PROTOTYPES, -Wmissing-prototypes]

14. Missing Newline At End Of File

GCC_WARN_ABOUT_MISSING_NEWLINE = YES

Warn when a source file does not end with a newline.

[GCC_WARN_ABOUT_MISSING_NEWLINE, -Wnewline-eof]

15. Multiple Definition Types for Selector

GCC_WARN_MULTIPLE_DEFINITION_TYPES_FOR_SELECTOR = NO

Warn if multiple methods of different types for the same selector are found during compilation. The check is performed on the list of methods in the final stage of compilation. Additionally, a check is performed for each selector appearing in a "@selector(...)" expression, and a corresponding method for that selector has been found during compilation. Because these checks scan the method table only at the end of compilation, these warnings are not produced if the final stage of compilation is not reached, for example because an error is found during compilation, or because the -fsyntax-only option is being used.

[GCC_WARN_MULTIPLE_DEFINITION_TYPES_FOR_SELECTOR, -Wselector]

16. Nonvirtual Destructor

GCC_WARN_NON_VIRTUAL_DESTRUCTOR = YES

Warn when a class declares an nonvirtual destructor that should probably be virtual, because it looks like the class will be used polymorphically.

[GCC_WARN_NON_VIRTUAL_DESTRUCTOR, -Wnon-virtual-dtor]

This is only active for C++ or Objective-C++ sources.

17. Other Warning Flags

WARNING_CFLAGS = -Wall -Wextra -Werror -Wbad-function-cast -Wmissing-declarations -Wmissing-prototypes -Wnested-externs -Wold-style-definition -Wstrict-prototypes -Wdeclaration-after-statement

Space-separated list of additional warning flags to pass to the compiler. Use this setting if Xcode does not already provide UI for a particular compiler warning flag.

[WARNING_CFLAGS]

18. Overloaded Virtual Functions

GCC_WARN_HIDDEN_VIRTUAL_FUNCTIONS = YES

Warn when a function declaration hides virtual functions from a base class. [GCC_WARN_HIDDEN_VIRTUAL_FUNCTIONS, -Woverloaded-virtual]

For example, in:

```
struct A
{
   virtual void f();
};
struct B: public A
{
   void f( int );
};
```

the A class version of f() is hidden in B, and code like this:

```
B * b;
b->f();
```

will fail to compile. This setting only applies to C++ and Objective-C++ sources.

19. Pedantic Warnings

GCC_WARN_PEDANTIC = YES

Issue all the warnings demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any -std option used.

[GCC_WARN_PEDANTIC, -pedantic]

20. Pointer Sign Comparison

GCC_WARN_ABOUT_POINTER_SIGNEDNESS = YES

Warn when pointers passed via arguments or assigned to a variable differ in sign.

[GCC_WARN_ABOUT_POINTER_SIGNEDNESS, -Wno-pointer-sign]

21. Prototype Conversion

GCC_WARN_PROTOTYPE_CONVERSION = NO

Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed point argument except when the same as the default promotion.

Also, warn if a negative integer constant expression is implicitly converted to an unsigned type. For example, warn about the assignment "x = -1" if "x" is unsigned. But do not warn about explicit casts like "(unsigned) -1".

[GCC_WARN_PROTOTYPE_CONVERSION, -Wconversion]

22. Sign Comparison

GCC_WARN_SIGN_COMPARE = YES

Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. This warning is enabled by -W, and by -Wall in C++ only.

[GCC_WARN_SIGN_COMPARE, -Wsign-compare]

23. Strict Selector Matching

GCC_WARN_STRICT_SELECTOR_MATCH = YES

Warn if multiple methods with differing argument and/or return types are found for a given selector when attempting to send a message using this selector to a receiver of type "id" or "Class". When this setting is disabled, the compiler will omit such warnings if any differences found are confined to types which share the same size and alignment.

[GCC_WARN_STRICT_SELECTOR_MATCH, -Wstrict-selector-match]

24. Treat Missing Function Prototypes as Errors

GCC_TREAT_IMPLICIT_FUNCTION_DECLARATIONS_AS_ERRORS = YES

Causes warnings about missing function prototypes to be treated as errors. Only applies to C and Objective-C.

[GCC_TREAT_IMPLICIT_FUNCTION_DECLARATIONS_AS_ERRORS, -Werror-implicit-function-declaration]

25. Treat Nonconformant Code Errors as Warnings

GCC_TREAT_NONCONFORMANT_CODE_ERRORS_AS_WARNINGS = NO

Enabling this option will downgrade messages about nonconformant code from errors to warnings. By default, G++ effectively sets -pedantic-errors without -pedantic; this option reverses that. This behavior and this option are superseded by -pedantic, which works as it does for GNU C.

[GCC_TREAT_NONCONFORMANT_CODE_ERRORS_AS_WARNINGS, -fpermissive]

26. Treat Warnings as Errors

GCC_TREAT_WARNINGS_AS_ERRORS = YES

Enabling this option causes all warnings to be treated as errors.

[GCC_TREAT_WARNINGS_AS_ERRORS, -Werror]

27. Typecheck Calls to printf/scanf

GCC_WARN_TYPECHECK_CALLS_TO_PRINTF = YES

Check calls to printf and scanf, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense.

[GCC_WARN_TYPECHECK_CALLS_TO_PRINTF, -Wno-format]

28. Undeclared Selector

GCC_WARN_UNDECLARED_SELECTOR = YES

Warn if a "@selector(...)" expression referring to an undeclared selector is found. A selector is considered undeclared if no method with that name has been declared before the "@selector(...)" expression, either explicitly in an @interface or @protocol declaration, or implicitly in an @implementation section. This option always performs its checks as soon as a "@selector(...)" expression is found, while -Wselector only performs its checks in the final stage of compilation. This also enforces the coding style convention that methods and selectors must be declared before being used.

[GCC_WARN_UNDECLARED_SELECTOR, -Wundeclared-selector]

29. Uninitialized Automatic Variables

GCC_WARN_UNINITIALIZED_AUTOS = YES

Warn if a variable might be clobbered by a setjmp call or if an automatic variable is used without prior initialization.

Detection of uninitialized automatic variable requires data flow analsys that is only enabled during optimized compilation.

Note that GCC cannot detect all cases where an automatic variable is initialized or all usage patterns that may lead to use prior to initialization.

[GCC_WARN_UNINITIALIZED_AUTOS, -Wuninitialized]

30. Unknown Pragma

GCC_WARN_UNKNOWN_PRAGMAS = YES

Warn when a #pragma directive is encountered which is not understood by GCC. If this command line option is used, warnings will even be issued for unknown pragmas in system header files. This is not the case if the warnings were only enabled by the -Wall command line option.

[GCC_WARN_UNKNOWN_PRAGMAS, -Wunknown-pragmas]

31. Unused Functions

GCC_WARN_UNUSED_FUNCTION = YES

Warn whenever a static function is declared but not defined or a non-inline static function is unused.

[GCC_WARN_UNUSED_FUNCTION, -Wunused-function]

32. Unused Labels

GCC_WARN_UNUSED_LABEL = YES

Warn whenever a label is declared but not used.

[GCC_WARN_UNUSED_LABEL, -Wunused-label]

33. Unused Parameters

GCC_WARN_UNUSED_PARAMETER = YES

Warn whenever a function parameter is unused aside from its declaration.

[GCC_WARN_UNUSED_PARAMETER, -Wunused-parameter]

34. Unused Values

GCC_WARN_UNUSED_VALUE = YES

Warn whenever a statement computes a result that is explicitly not used.

[GCC_WARN_UNUSED_VALUE, -Wunused-value]

35. Unused Variables

GCC_WARN_UNUSED_VARIABLE = YES

Warn whenever a local variable or non-constant static variable is unused aside from its declaration.

[GCC_WARN_UNUSED_VARIABLE, -Wunused-variable]

36. Warn About Deprecated Functions

GCC_WARN_ABOUT_DEPRECATED_FUNCTIONS = YES

Warn about the use of deprecated functions, variables, and types (as indicated by the 'deprecated' attribute).

[GCC_WARN_ABOUT_DEPRECATED_FUNCTIONS, -Wno-deprecated-declarations]

37. Warn About Undefined Use of offsetof Macro

GCC_WARN_ABOUT_INVALID_OFFSETOF_MACRO = YES

Unchecking this setting will suppress warnings from applying the offsetof macro to a non-POD type. According to the 1998 ISO C++ standard, applying offsetof to a non-POD type is undefined. In existing C++ implementations, however, offsetof typically gives meaningful results even when applied to certain kinds of non-POD types. (Such as a simple struct that fails to be a POD type only by virtue of having a constructor.) This flag is for users who are aware that they are writing non-portable code and who have deliberately chosen to ignore the warning about it.

The restrictions on offsetof may be relaxed in a future version of the C++ standard.

[GCC_WARN_ABOUT_INVALID_OFFSETOF_MACRO, -Wno-invalid-offsetof]