Xcode 4 Template Documentation

Synopsis

This document explains how to create File and Project Templates for Xcode 4. Xcode 4 uses a template format significantly different from the one used in Xcode 3, which renders all previous information on Template creation useless. The new format is also much more complex and allows for greater flexibility, for example a template can now consist of multiple inherited templates.

This Xcode 4 Template documentation first gives you a quick overview what File and Project templates are. Step-by-step tutorials show you how to create your own File and Project Templates. In the reference section the format of the TemplateInfo.plist and related files are meticulously documented. Finally there's the Frequently Asked Questions (FAQ) section with answers to common questions. The documentation is fully hyperlinked so that you can quickly look up keywords in the reference section.

Several example files for File and Project Templates help you get started and can be used as the basis for your own templates.

Disclaimers

Please note that this documentation is not complete. This is owed to the fact that a lot of experimentation was involved in creating this documentation. There are still many unknowns to be discovered - you will find annotations regarding these throughout the document. Nevertheless, this document contains substantially more information than what is currently available on the Internet.

The document has not been reviewed by a professional copy editor nor proofread. I'm sorry for any grammatical and syntactical errors that will show up here and there because I'm not a native english speaker.

License

Quality documentation takes a lot of time and effort to produce. This document is only available for a nominal fee.

Copyright

Acknowledgements

I have found little information on the web about the new Xcode 4 template format. The little I found helped me getting started. The following list contains all the articles that helped me getting started with this document:

- Minimal Project Template with example:
  http://blog.boreal-kiss.net/2011/03/11/a-minimal-project-template-for-xcode-4/
- List of replacement placeholder strings:
  http://gallantgames.com/post/3771670972/xcode-4-templates
- Solution on how to place files in subgroups:
  http://stackoverflow.com/questions/5445640/creating-sub-groups-in-xcode-4-templates

In addition I would like to thank Mohammad Azam and Nate Weiss for reviewing this documentation.

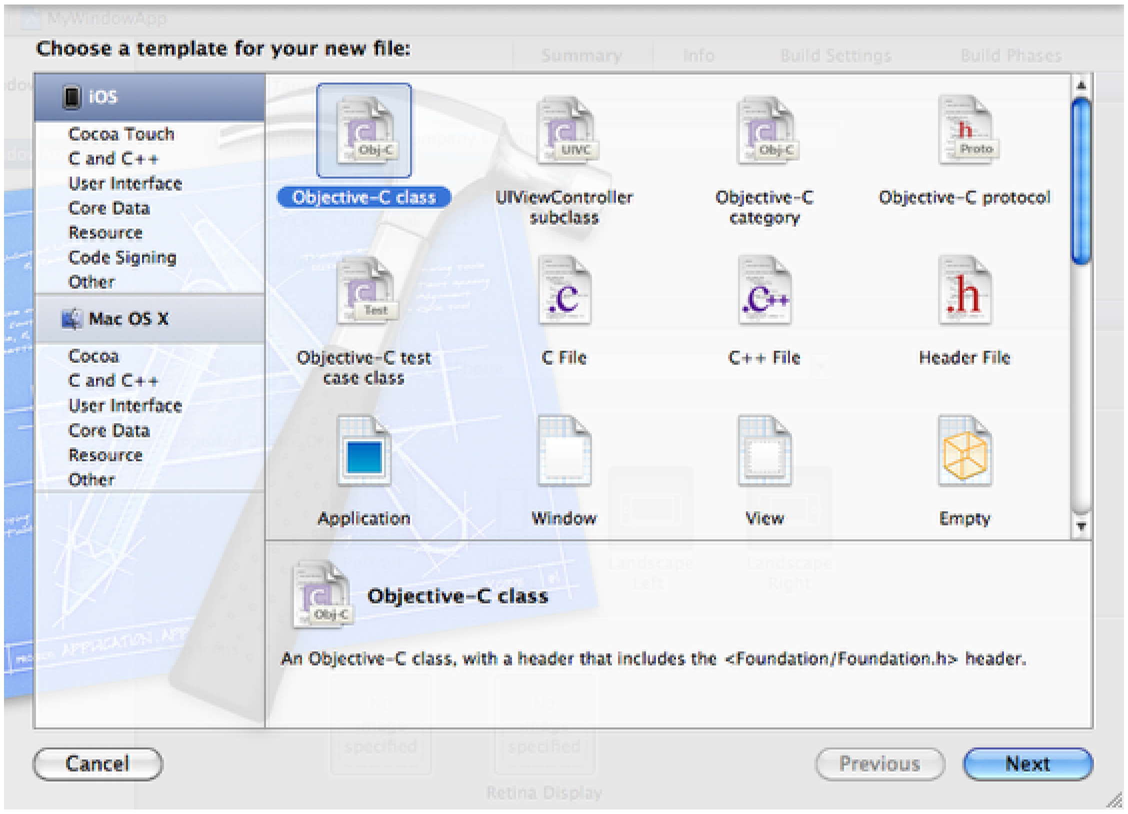
Introduction to Xcode 4 Templates

Let's begin with a quick overview and a few screenshots of existing templates, just to wet your appetite and show you what's possible. At the end I will guide you through prerequisites that are helpful to know when creating your own custom templates.

Introducing File Templates

Whenever you choose File -> New -> New File ... from the Xcode menu, you'll be presented with a dialog showing a selection of file templates. Depending on the type of the file template, one or more files are created and prefilled with default text depending on the file template. The File Template can also contain placeholders which will be replaced during creation, for example the date and time might be filled in or the user's name and company may be added to a copyright header.

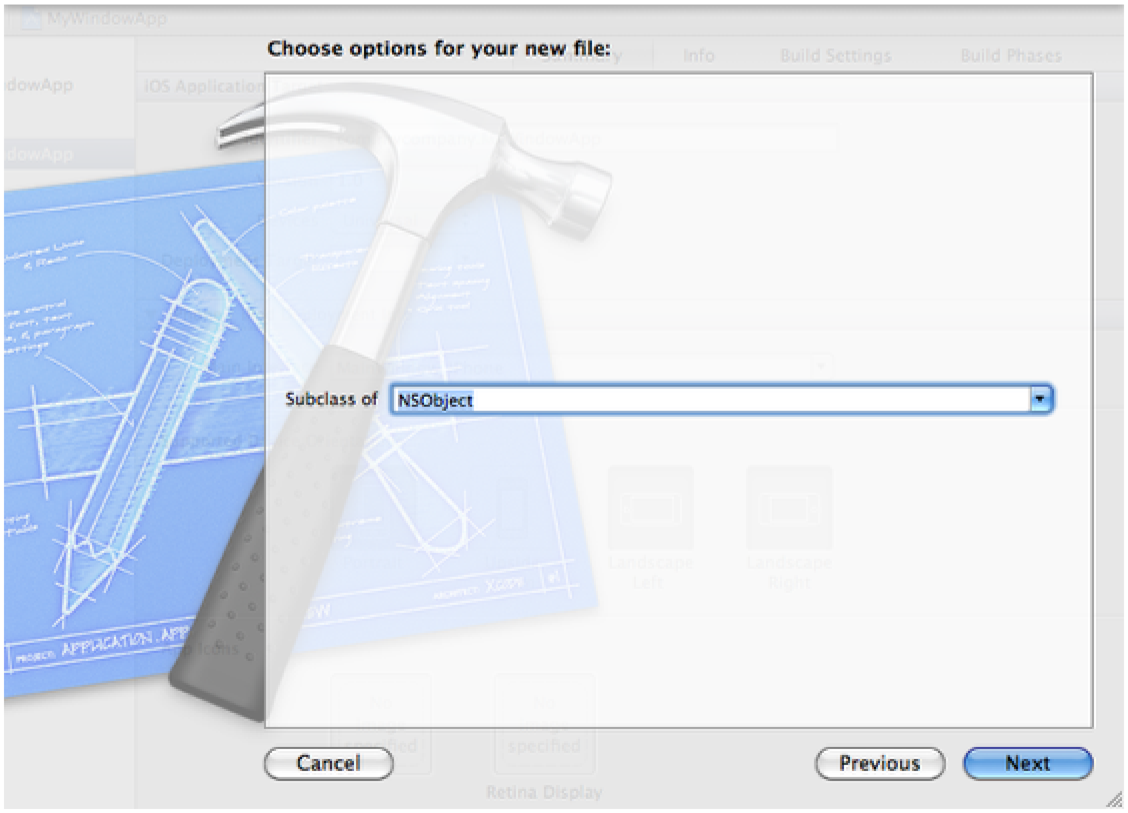
The following images show how to create an Objective-C class template.



Objective-C class

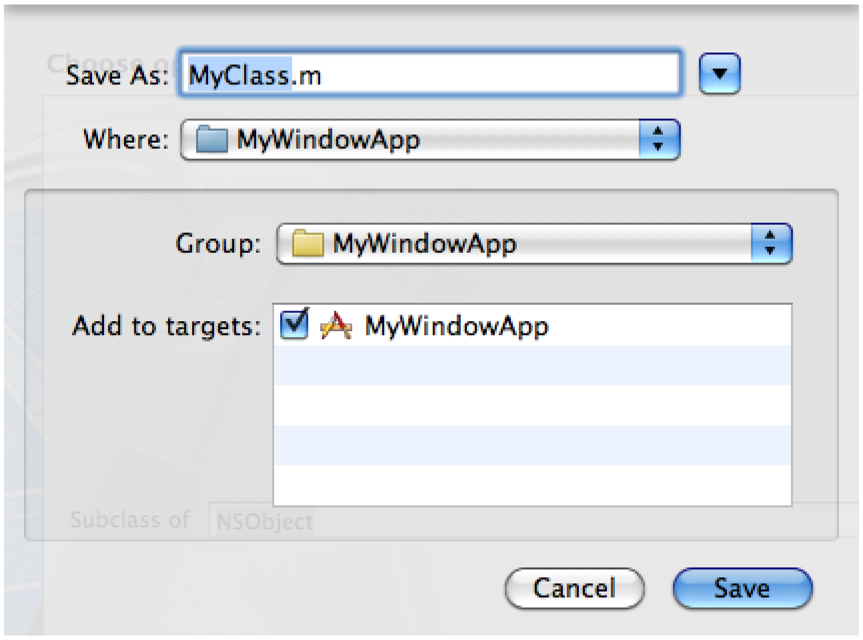
An Objective-C class, with a header that includes the <Foundation/Foundation.h> header.

You will get the option to pick the subclass of the newly created class. The available subclasses are configurable just like the entire options screen is.


Choose options for your new file:

Subclass of NSObject

Creating files from a template also allows you to specify the name of the file and where to put it. In your file templates you can specify, among other things, the default file name and the group where the new file should be placed.

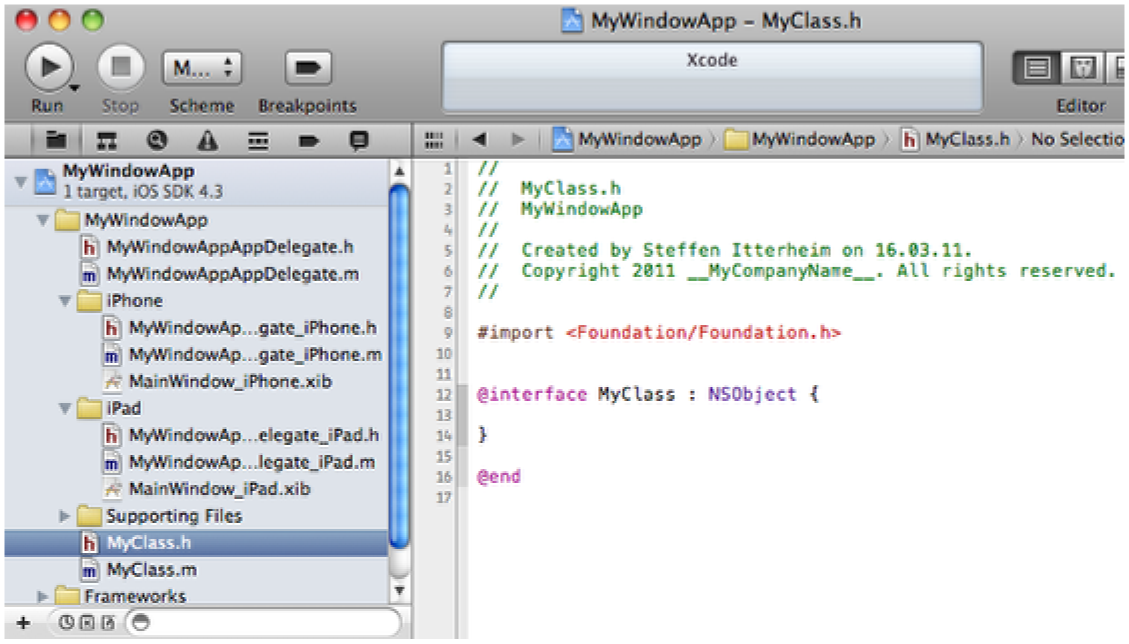

Save As: MyClass.m

Where: MyWindowApp

Group: MyWindowApp

Add to targets: ✓ MyWindowApp

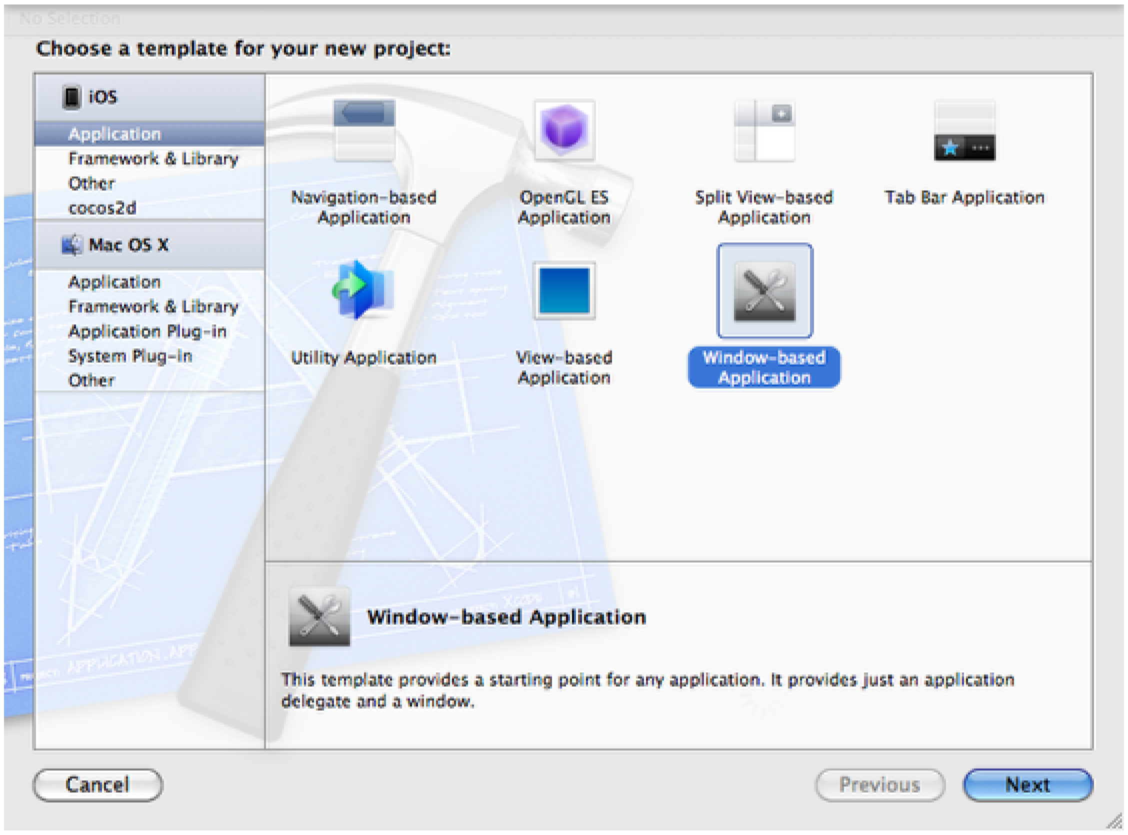
Finally the resulting MyClass class was created. Notice that actually two files were created, a .h and a .m file. In the screenshot below, the MyClass is properly subclassed from NSObject, the comment shows that it was created by me and the date I created it.



```
//
//  MyClass.h
//  MyWindowApp
//
//  Created by Steffen Itterheim on 16.03.11.
//  Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface MyClass : NSObject {

}

@end
```

Introducing Project Templates

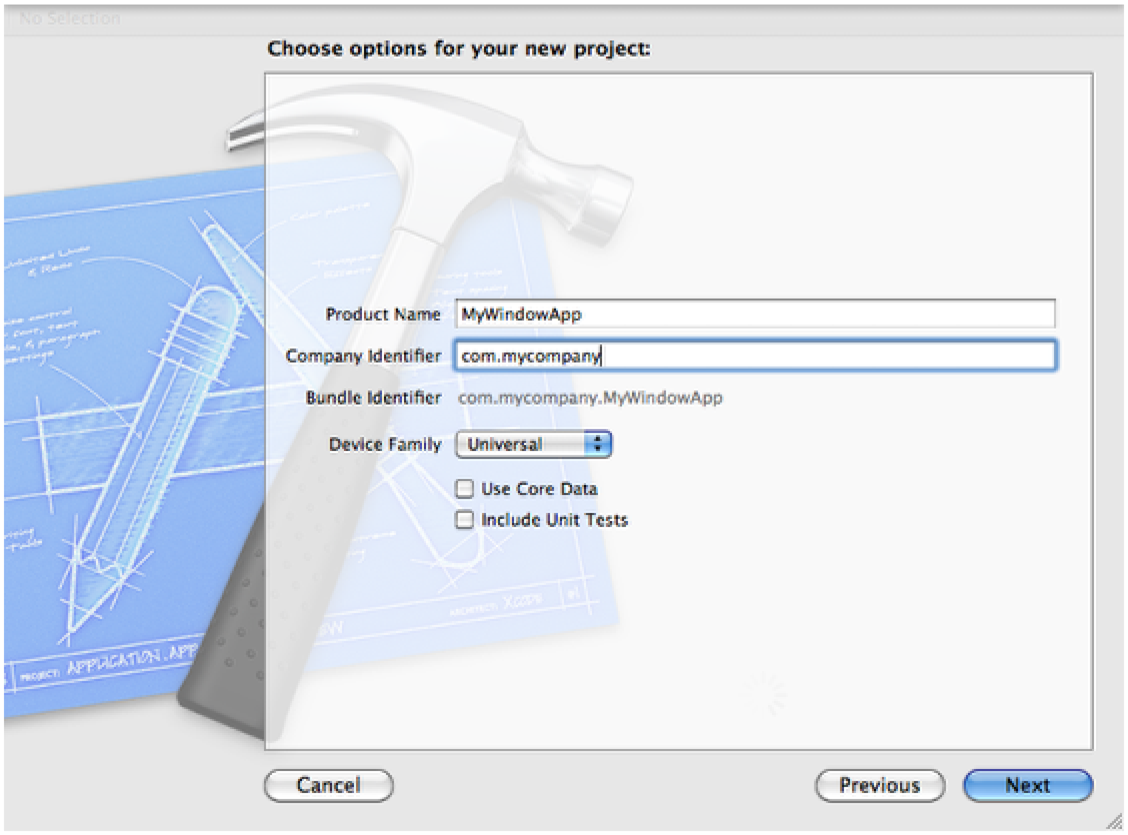
Project Templates are presented to you by Xcode whenever you choose File -> New -> New Project... from Xcode's menu. Below is an overview of the iOS Application project templates. There are countless more, you might want to try them all out and at least click on the Next button once to see which configuration options they have to offer.



Window-based Application

This template provides a starting point for any application. It provides just an application delegate and a window.

In the case of the Window-based Application project template, you'll have several options that you can fill out to customize the new project. This "choose options" dialog can bemodified to ask users for various options before proceeding with the creation of a new Xcode project.



Product Name: MyWindowApp

Company Identifier: com.mycompany

Bundle Identifier: com.mycompany.MyWindowApp

Device Family: Universal

☐ Use Core Data

☐ Include Unit Tests

If you click on Next in this step after filling in all options, you'll be asked where to save the new project. Following that, Xcode will create the project for you and you can start working.

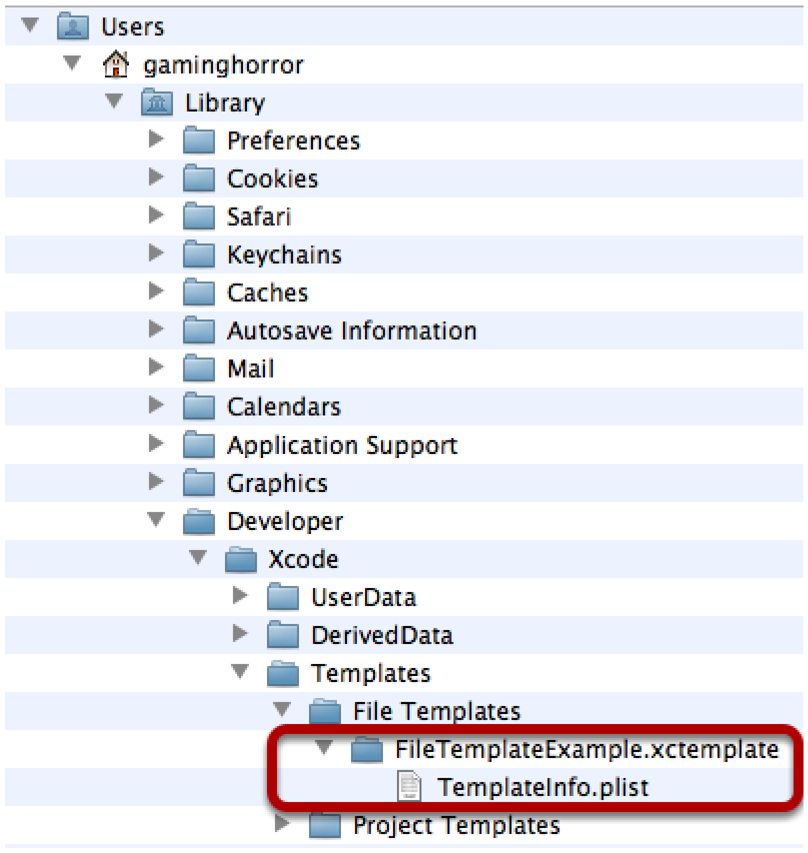
Xcode 4 Template Tutorials

This section contains Tutorials for creating File Templates and Project Templates.

Basic File Template Tutorial

In this tutorial you will learn how to create your own File Template. Compared to Project Templates, File Templates are considerably easier to create and you will benefit from using them throughout the lifetime of your project, as opposed to just being the starting point of the project. You can automate a lot of tedious typing by making use of File Templates, as a result they're generally very popular among Xcode developers.

File Template Folder



The first step for creating both File and Project Templates is to setup the folders where you can create custom Templates. By default, the folder doesn't exist on your system, so you'll have to create it. It is not recommended to modify existing templates from Apple nor should you create your own custom templates in the same location as Apple's templates, since that folder might be erased when you update Xcode / the iOS SDK.

In the above screenshot you'll see the full path to custom File Templates, which is: `~/Library/Developer/Xcode/Templates/File Templates/...`

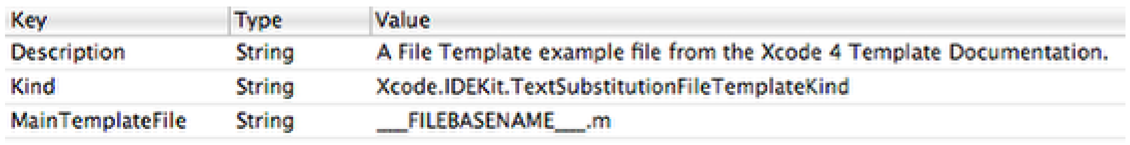
⚠ You will have to create the `Templates` and `File Templates` folder if it does not exist. And if you want your templates to be in a category other than "File Templates" simply create another subfolder with the desired category name, for example "My Templates" and then add the .xctemplate folders into that subfolder.

✓ The ~/ part of the path is shorthand for the /Users/yourusername/ folder.

I've already created another subfolder which is highlighted in the above image. The folder is named `FileTemplateExample.xctemplate` and contains a TemplateInfo.plist file which I'll get to next. For now just keep in mind that to create a file template, you should create a subfolder unter `File Templates` and while you can name the folder any way you want, it **has** to end with `.xctemplate`, otherwise Xcode won't recognize it as a Template.

Refer to Template Folder Locations in the reference section to learn more about template folder locations.

Minimal TemplateInfo.plist



Key	Type	Value
Description	String	A File Template example file from the Xcode 4 Template Documentation.
Kind	String	Xcode.IDEKit.TextSubstitutionFileTemplateKind
MainTemplateFile	String	___FILEBASENAME___.m

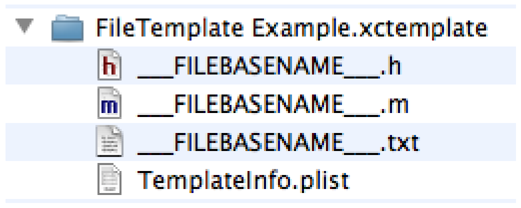
If you open the TemplateInfo.plist by double-clicking it, Xcode will launch and show you the contents of the property list file. This minimal example contains only three keys: Description, Kind and MainTemplateFile.

Description is just that, a description that explains what kind of file will be created.

Kind is absolutely necessary for the Template to work, without this key the Template won't show up in Xcode. All File Templates have their Kind key set to `Xcode.IDEKit.TextSubstitutionFileTemplateKind` with one exception: file templates that create a class for an NSManagedObject data model use the Kind `Xcode.IDECoreDataModeler.ManagedObjectTemplateKind`.

MainTemplateFile specifies which (default) icon should be used to display the File Template, which is based on the file's extension. It also specifies which of the files in the file template folder should be opened for viewing in Xcode after the files have been created.

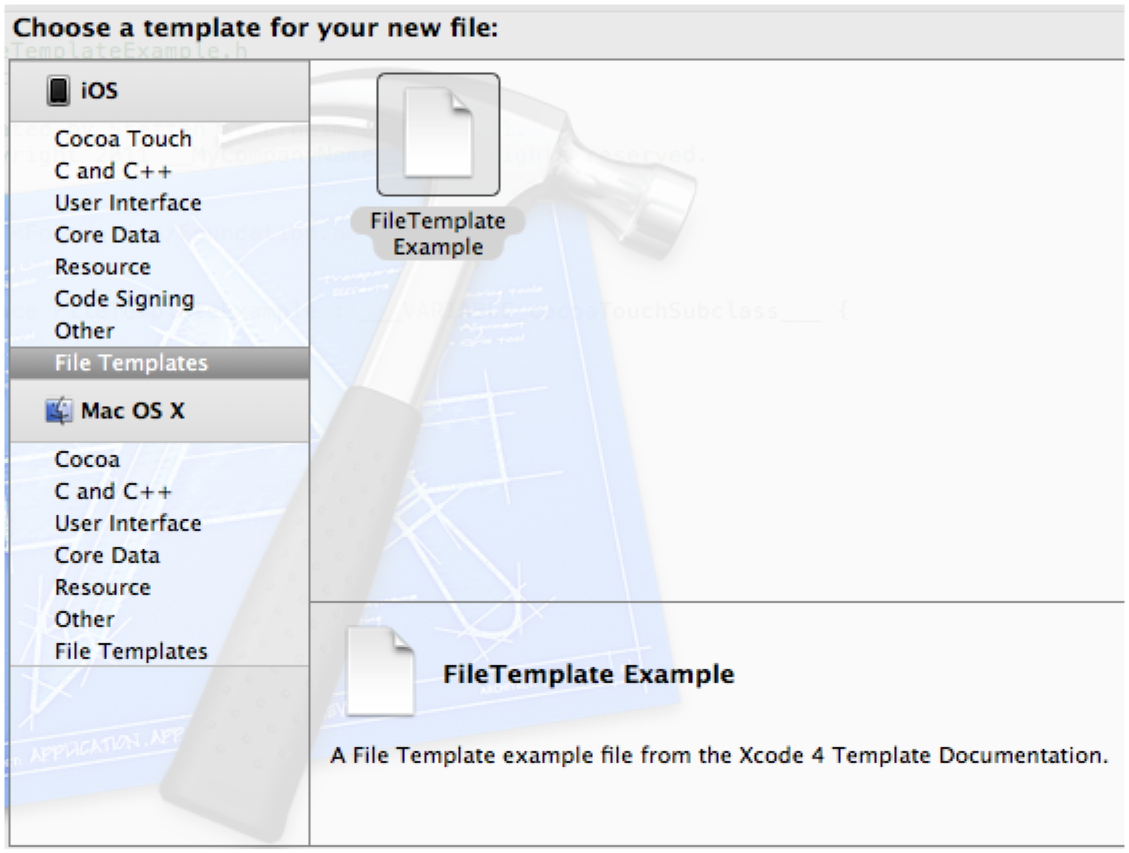
File Names



In most cases the filenames of your file template carry a special name like `___FILEBASENAME___` so that the file name will take on the name specified by the user. There are various placeholders that you can use both in filenames and the file's contents. You can find a complete list of placeholders in the Placeholder Reference and in the Basic File Template Example template files.

Xcode always copies all files in the .xctemplate folder of a File Template and adds them to the project, except for the TemplateInfo.plist file. However, this is not the case with Project Templates. Project Templates (unfortunately) only copy files that are specified in the TemplateInfo.plist Definitions section.

Try it!



A File Template example file from the Xcode 4 Template Documentation.

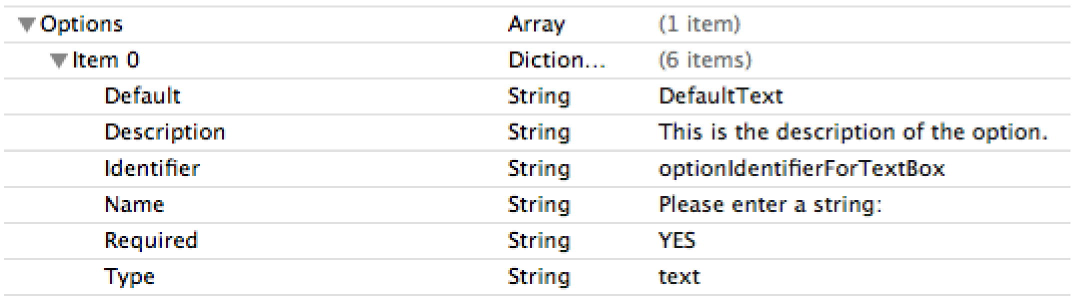
In Xcode, go to File -> New ... -> New File and you should see the .xctemplate folder being listed as a category on the left side. If you select that category the FileTemplate Example template should show up.

If you create a file based on this template, it will actually create three files whose filename is based on your input and with the extensions .h, .m and .txt. Also, all the placeholder strings have been replaced with actual date, filename, project name, identifiers and so on.

Advanced File Template Tutorial

The Advanced File Template Tutorial builds on the Basic File Template Tutorial and explains how to add user input options and other settings.

Adding User Options

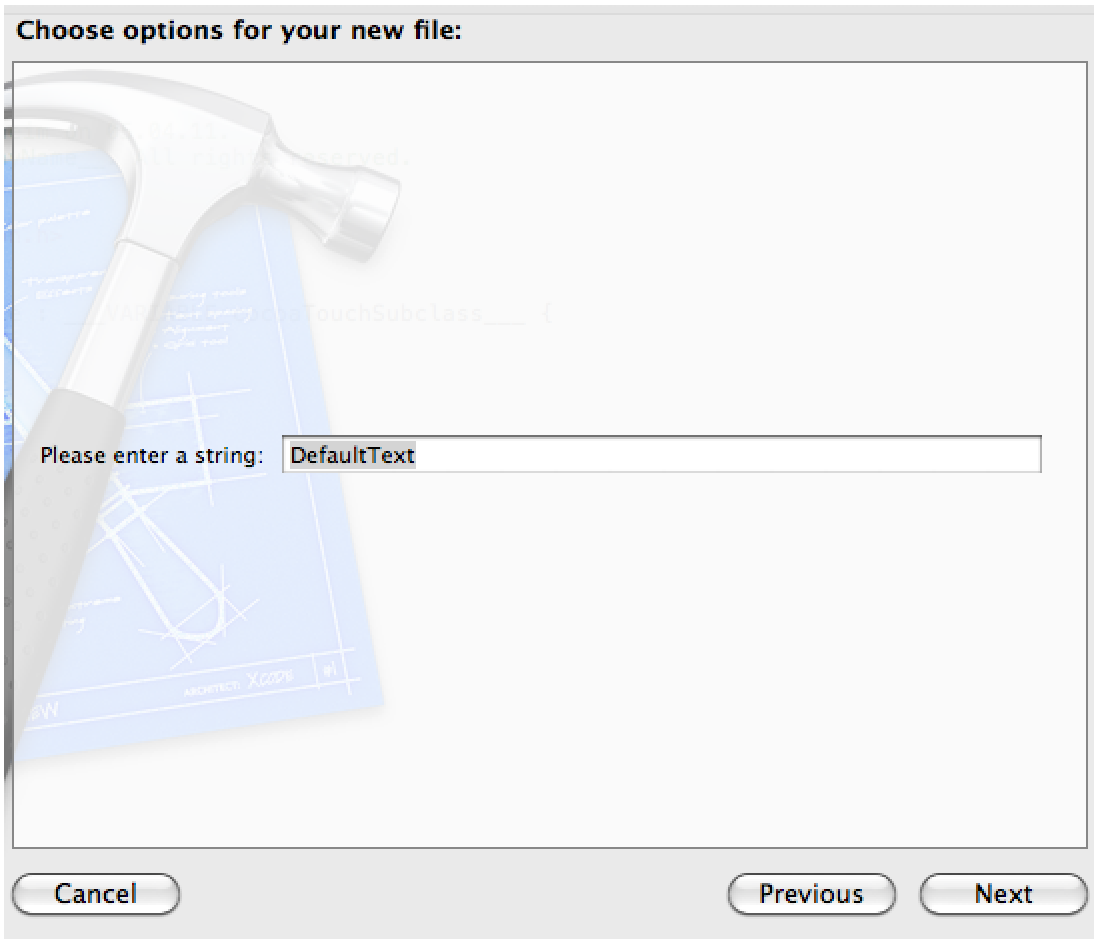


▼Options	Array	(1 item)
▼Item 0	Diction...	(6 items)
Default	String	DefaultText
Description	String	This is the description of the option.
Identifier	String	optionIdentifierForTextBox
Name	String	Please enter a string:
Required	String	YES
Type	String	text

Options allow templates to ask the user for input. The simplest option asks the user for text input, the necessary addition to the TemplateInfo.plist file to create a text input box is shown above.

Not all options described in the Options reference are available in File Templates. In fact, only the text and static Types options work flawlessly with File Templates.

Text Input


Choose options for your new file:

Please enter a string: DefaultText

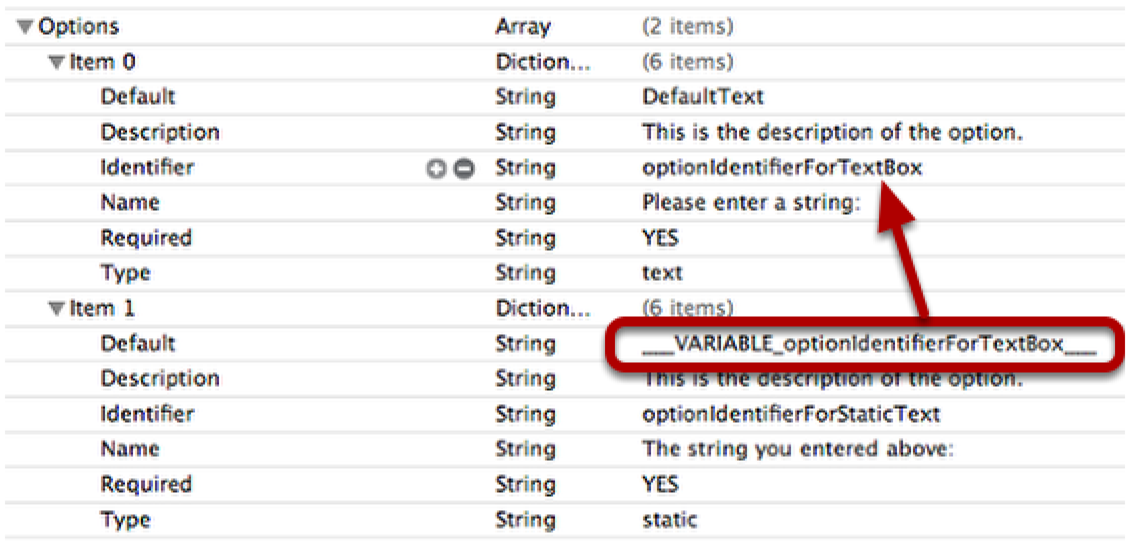
If you create a new file based on the advanced template, you will be presented with the new "Choose options ..." step before proceeding with the dialog where you specify the filename and location.

Using Option Identifiers for variable placeholders

```
Before replacing:

___VARIABLE_optionIdentifierForTextBox___

After replacing:

testing123
```

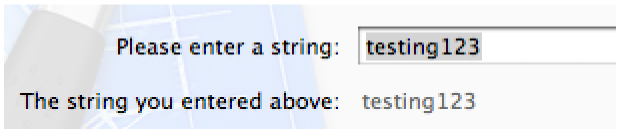
You can use the Identifier key to create a Variable Placeholder which will be replaced by the text the user entered into an option textbox. This is done with the special placeholder `___VARIABLE_identifier___` where identifier is the Identifier of an option. In this case the identifier would have to be named `___VARIABLE_optionIdentifierForTextBox___` and you can then use this placeholder in any file, or even as the file name in case you want to allow the user to specify filenames of specific template files.

More fun with variable placeholders



▼ Options	Array	(2 items)
▼ Item 0	Diction...	(6 items)
Default	String	DefaultText
Description	String	This is the description of the option.
Identifier	String	optionIdentifierForTextBox
Name	String	Please enter a string:
Required	String	YES
Type	String	text
▼ Item 1	Diction...	(6 items)
Default	String	___VARIABLE_optionIdentifierForTextBox___
Description	String	This is the description of the option.
Identifier	String	optionIdentifierForStaticText
Name	String	The string you entered above:
Required	String	YES
Type	String	static

You can even use the variable placeholders within Options themselves. In the above image the TemplateInfo.plist now contains an additional static label with DefaultText set to the variable placeholder for the text box.



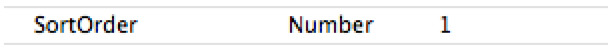
Please enter a string: testing123

The string you entered above: testing123

The result is a static label that is updated as you enter text. While this example is pretty pointless, it does allow you to construct more complex strings and show them to the user to verify their correctness before proceeding. Apple uses this technique to construct and verify reverse domain name notation for the default bundle identifier which consists of the product name and your company name.

The identifier part can also be followed by a colon and a special modifier keyword. For example using `___VARIABLE_optionIdentifierForTextBox:identifier___` will modify the string so that it will be a legal C-style variable name. Refer to the Variable Placeholders section for more information on modifiers.

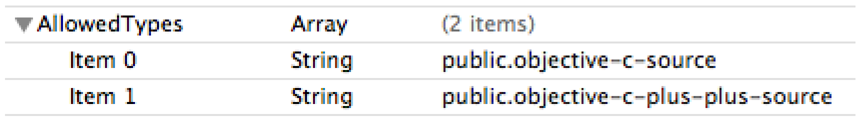
Sorting Templates



SortOrder	Number	1

By default, all templates in a category are sorted alphabetically. You can change that by adding a SortOrder key to each template. Templates are sorted from highest SortOrder number to lowest, which may be confusing at first. Refer to the SortOrder reference section for more information.

Allowing only specific file extensions

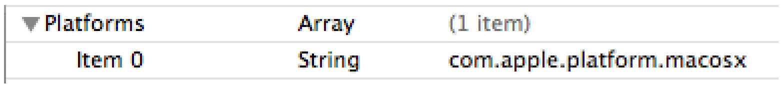


▼ AllowedTypes	Array	(2 items)
Item 0	String	public.objective-c-source
Item 1	String	public.objective-c-plus-plus-source

Nothing would stop the user from specifying an invalid file extension when creating a new file from a File Template. You can limit the allowed file extensions by using the AllowedTypes key. In the above image only the legal extensions for Objective-C source files will be allowed, which are .m and .mm - if the user tries to enter a different file extension, then .m will forcibly be appended to the filename by Xcode.

See the AllowedTypes reference section for more information and a list of Uniform Type Identifiers.

Making a Template Platform-Specific

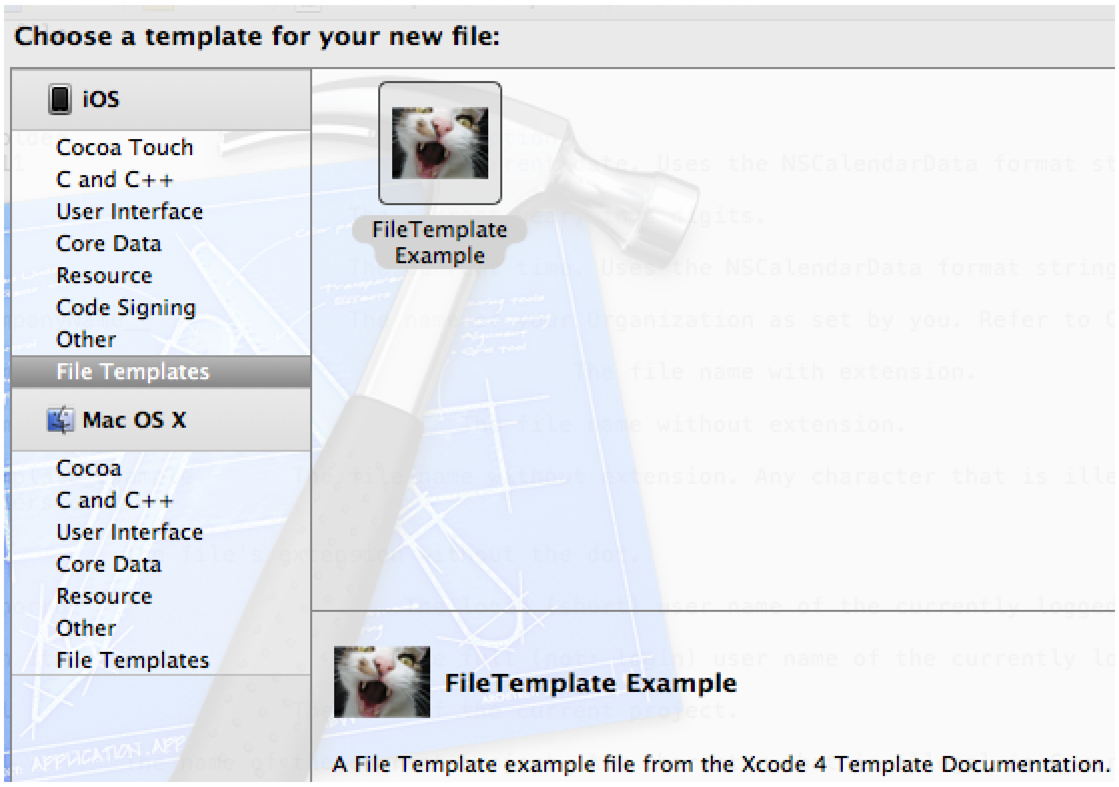


▼ Platforms	Array	(1 item)
Item 0	String	com.apple.platform.macosx

This is merely a cosmetic issue. If your template works only for either iOS or Mac OS, you can have the template show up only in the platform specific categories by adding the Platforms key as seen in the above image. You can find the available values in the Platforms reference.

⚠ The Platforms key does not automatically change any build settings that are needed to build code for a certain platform. To do that, you will have to use the Configurations and SharedSettings dictionaries found under the Project and Targets root keys.

Adding a custom Icon



A File Template example file from the Xcode 4 Template Documentation.

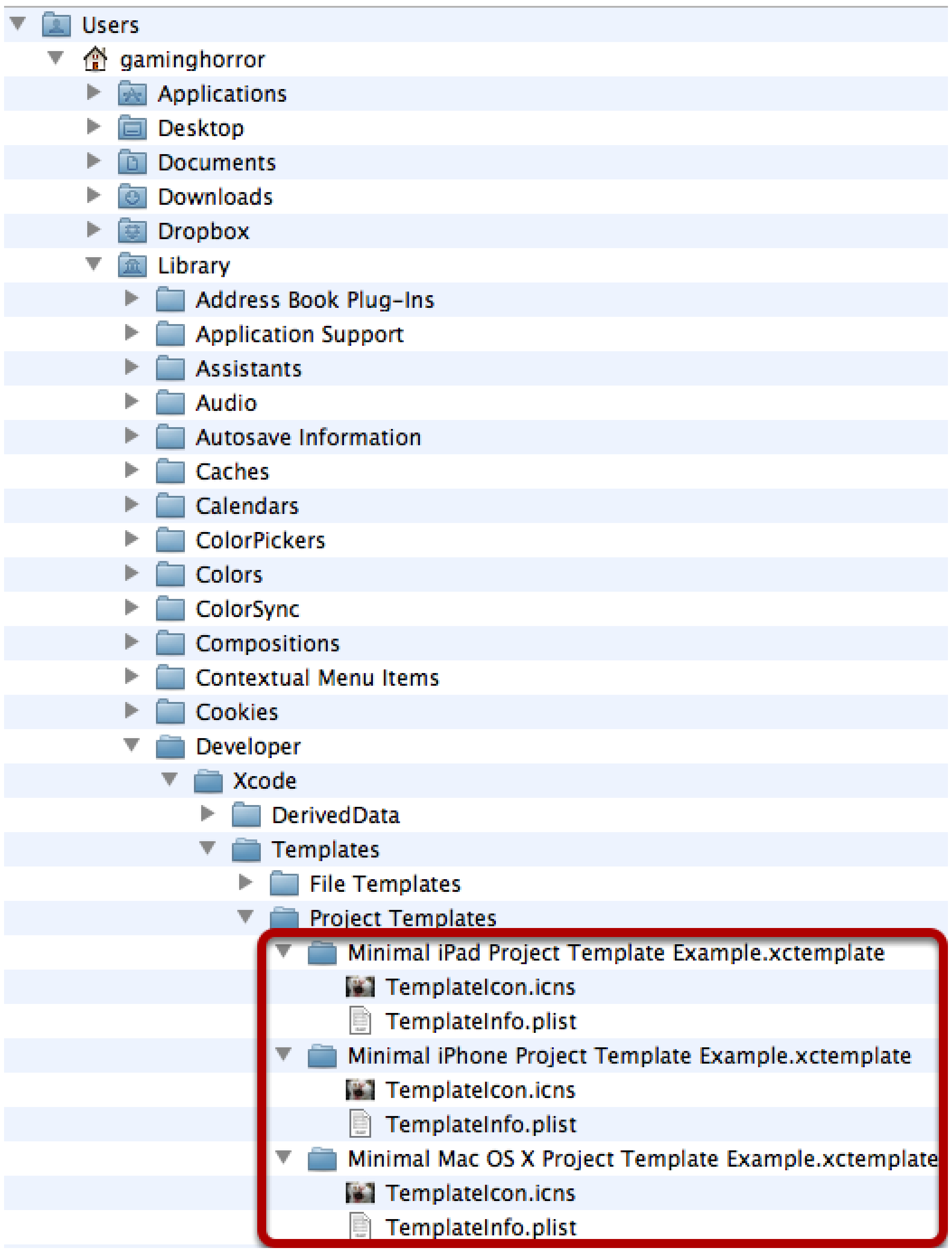
If you don't like the default Icon, you only have to add a file named `TemplateIcon.icns` to the .xctemplate folder. You can learn how to create your own icons with the Icon Composer app (installed with Xcode) in the Icon Composer section.

Minimal Project Template Tutorial

Since Project Templates are much more complex than File Templates it helps to start with a simple tutorial that shows you the absolute minimum that is necessary to create an empty project from a template without any files or modified settings. This tutorial also contains the basic information that applies to all Project Templates.

> ⚠ This tutorial teaches you how to create a working project template. However it will not create a working project since there are simply too many settings to create and configure which is beyond the scope of this tutorial. See also the first question in the Frequently Asked Questions (FAQ) section and refer to the Basic Project Template Tutorial for a working project.

Project Templates Folder Location



```
Users
  gaminghorror
    Applications
    Desktop
    Documents
    Downloads
    Dropbox
    Library
      Address Book Plug-Ins
      Application Support
      Assistants
      Audio
      Autosave Information
      Caches
      Calendars
      ColorPickers
      Colors
      ColorSync
      Compositions
      Contextual Menu Items
      Cookies
      Developer
        Xcode
          DerivedData
          Templates
            File Templates
            Project Templates
              Minimal iPad Project Template Example.xctemplate
                TemplateIcon.icns
                TemplateInfo.plist
              Minimal iPhone Project Template Example.xctemplate
                TemplateIcon.icns
                TemplateInfo.plist
              Minimal Mac OS X Project Template Example.xctemplate
                TemplateIcon.icns
                TemplateInfo.plist
```

As with File Templates, you first have to navigate to the Project Templates folder and create it if necessary.

In the above screenshot you'll see the full path to custom Project Templates, which is:
`~/Library/Developer/Xcode/Templates/Project Templates/...`

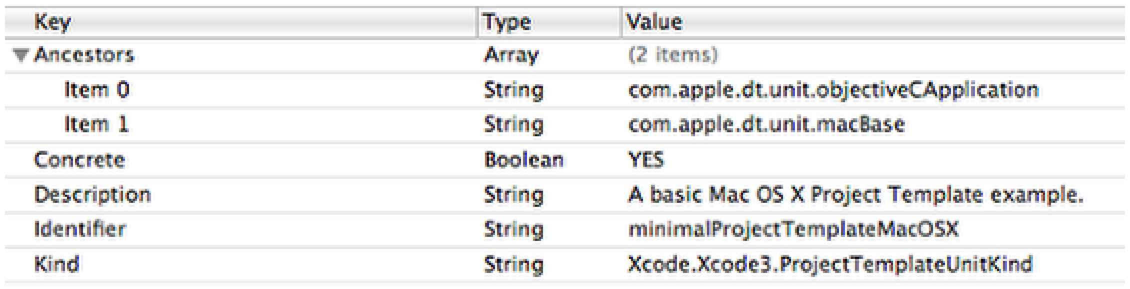
✓ The ~/ part of the path is shorthand for the /Users/yourusername/ folder.

⚠ You will have to create the `Templates` and `Project Templates` folder if it does not exist. And if you want your templates to be in a category other than "Project Templates" simply create another subfolder with the desired category name, for example "My Templates" and then add the .xctemplate folders into that subfolder.

I've already created several project template folders which are highlighted in the above image. Each template is for a different target platform, the differences are merely the Ancestors used in each template. Each folder contains a TemplateInfo.plist and the TemplateIcon.icns for the template's icons. As with File Templates any folder containing project template files **has** to end with `.xctemplate`, otherwise Xcode won't recognize it as a Template.

Refer to Template Folder Locations in the reference section to learn more about template folder locations.

Minimal TemplateInfo.plist



Key	Type	Value
▼ Ancestors	Array	(2 items)
Item 0	String	com.apple.dt.unit.objectiveCApplication
Item 1	String	com.apple.dt.unit.macBase
Concrete	Boolean	YES
Description	String	A basic Mac OS X Project Template example.
Identifier	String	minimalProjectTemplateMacOSX
Kind	String	Xcode.Xcode3.ProjectTemplateUnitKind

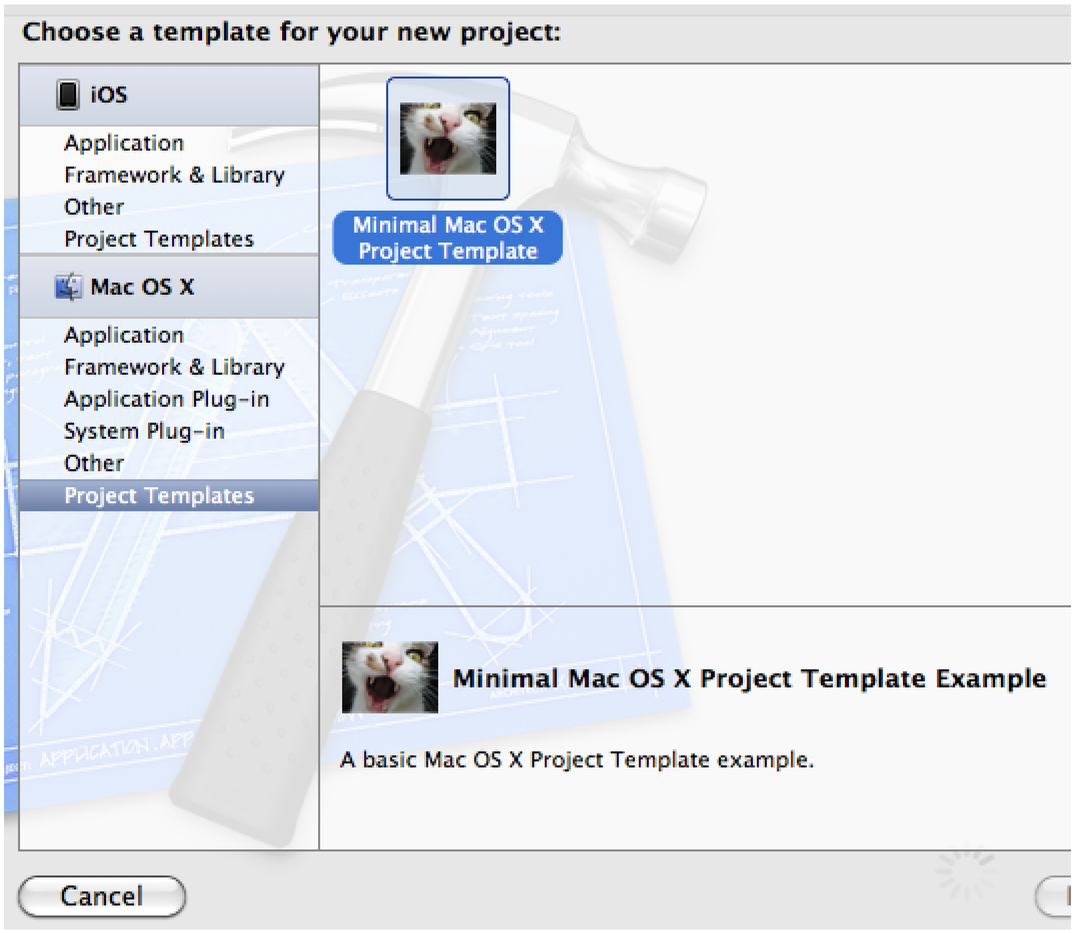
The above image shows what goes into a minimal Project Template. From top to bottom, here's an explanation what these settings do.

Key	Description
Ancestors	With Ancestors you can include - or in OOP terms: inherit from - other TemplateInfo.plist by specifying their Identifier. This is most useful if you have multiple project templates which all have common TemplateInfo.plist information that you want to create and maintain in a single template rather than in each project template individually. In this example, two commonly used base project templates for creating Mac OS X applications written in Objective-C are included. You can get a list of available base templates by looking at the Ancestors reference section.
Concrete	Concrete must be set to YES in order for the Project Template to appear in the New Project dialog. Concrete project templates are never included by other project templates, ie they are the last template in the hierarchy. The project templates that are included via the Ancestors key above do not have a Concrete key or it is set to NO because non-concrete project templates are meant to be included by other project templates.
Description	A string that describes what this Project Template does. It is shown in the New Project dialog when the template is selected.

Identifier	A unique identifier string, usually using reverse domain notation. With the Identifier it is possible to refer to other Project Templates via Ancestors. The Identifier key is mandatory for all project templates and it must be unique. If the Identifier is missing Xcode will crash when trying to open the New Project dialog, and if it's not unique the project template might not work or won't show up in the New Project dialog.
Kind	Kind is a special keyword that tells Xcode the type of template. All TemplateInfo.plist files to be used with Project Templates must use the `Xcode.Xcode3.ProjectTemplateUnitKind` keyword. If Kind is set incorrectly or omitted, the Project Template will not work.

Create the minimal project

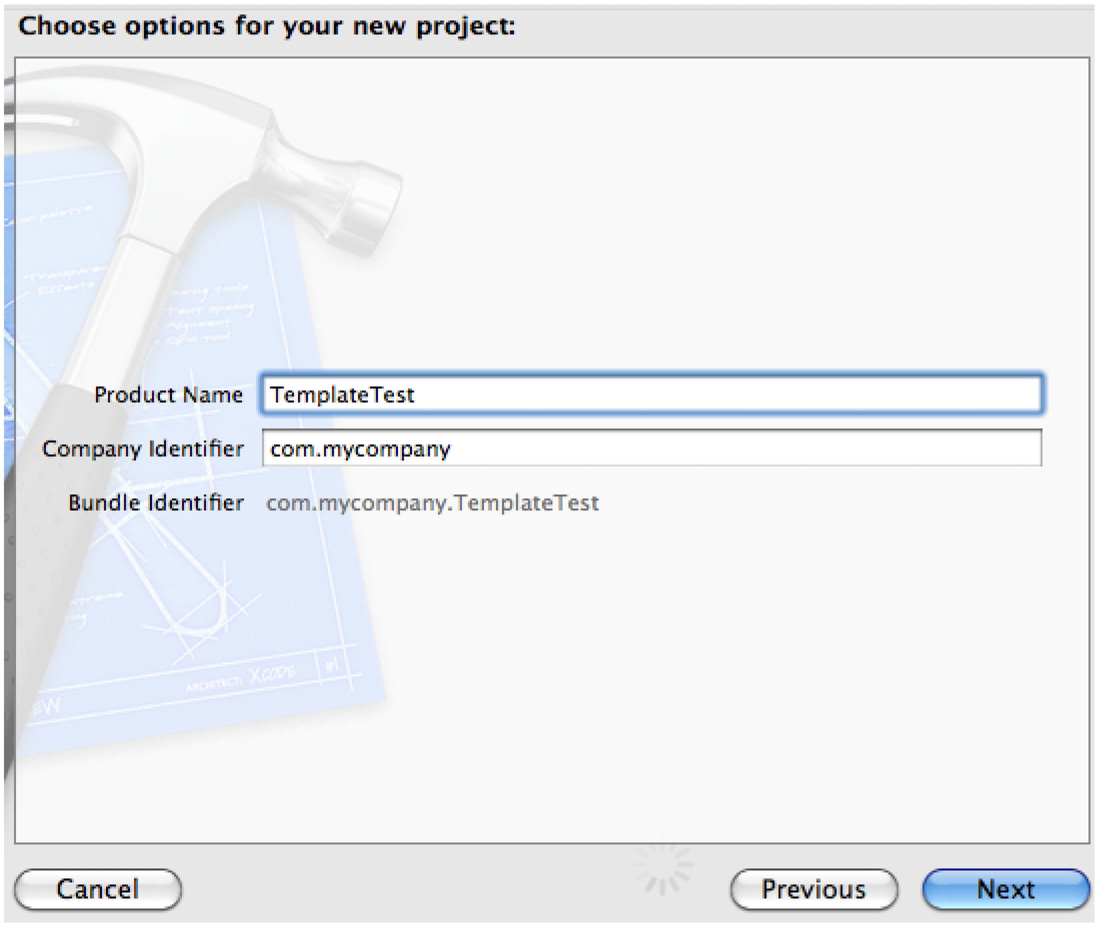
Let's see what we get from creating this minimal project template:



Minimal Mac OS X Project Template Example

A basic Mac OS X Project Template example.

Enter necessary options


Choose options for your new project:

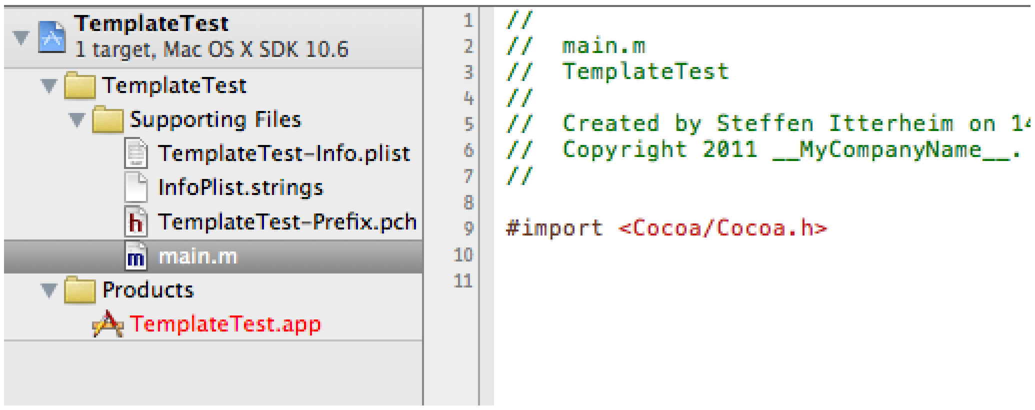
Product Name: TemplateTest

Company Identifier: com.mycompany

Bundle Identifier: com.mycompany.TemplateTest

Then click the Next button and save the project.

New Minimal Project



```
//
//  main.m
//  TemplateTest
//
//  Created by Steffen Itterheim on 14
//  Copyright 2011 __MyCompanyName__.
//

#import <Cocoa/Cocoa.h>


```

The project contains several files which were created by the Ancestors used in this minimal project template example. This may seem like magic because you did not add these files. Moreover you won't find all of these files in the file system, not even in Apple's template folders, because the most common files are actually created and filled with content entirely via TemplateInfo.plist settings.

They are however not complete, for example the main.m file is missing the `void main(int` `argc, char** argv)` function. This is just one reason why this minimal project template will not create a working project. We will fix this situation in the Basic Project Template Tutorial by starting a new project template based on one of the existing project templates.

Basic Project Template Tutorial

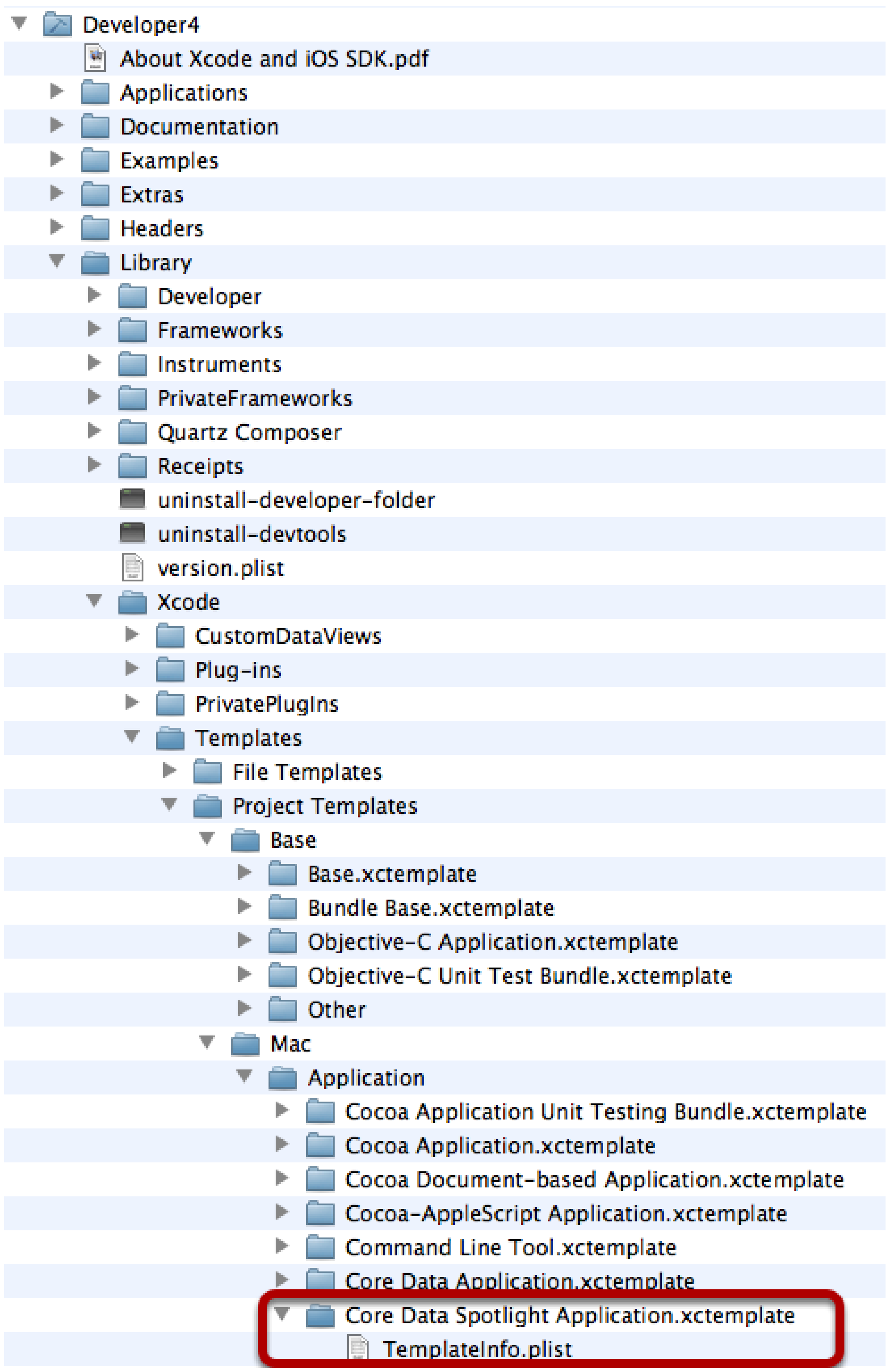
Since you'll rarely, if ever, be starting your own project templates from scratch we'll take a look at existing templates provided by Apple and discuss how they work. These templates are frequently used as starting points for writing your own project templates.

> The project templates in this Basic Project Template Tutorial are not part of the Xcode 4 Template Documentation download. I'm not a lawyer but I assume that it might not be legal to re-distribute Xcode files, even if slightly modified, so I chose not to bundle them. Of course if you follow these tutorials you'll create these project templates anyway. Besides that, learning by doing always works best.

Your Cocoa Application Project Template

As basis for Mac OS X project templates the Cocoa Application template is the ideal starting point for creating your own customized project template. It's also not as complex as the project templates for iOS, so we'll start with the Cocoa Application template.

Copy existing project template



```
Developer4
    About Xcode and iOS SDK.pdf
    Applications
    Documentation
    Examples
    Extras
    Headers
    Library
        Developer
        Frameworks
        Instruments
        PrivateFrameworks
        Quartz Composer
        Receipts
        uninstall-developer-folder
        uninstall-devtools
        version.plist
        Xcode
            CustomDataViews
            Plug-ins
            PrivatePlugIns
            Templates
                File Templates
                Project Templates
                    Base
                        Base.xctemplate
                        Bundle Base.xctemplate
                        Objective-C Application.xctemplate
                        Objective-C Unit Test Bundle.xctemplate
                        Other
                    Mac
                        Application
                            Cocoa Application Unit Testing Bundle.xctemplate
                            Cocoa Application.xctemplate
                            Cocoa Document-based Application.xctemplate
                            Cocoa-AppleScript Application.xctemplate
                            Command Line Tool.xctemplate
                            Core Data Application.xctemplate
                            Core Data Spotlight Application.xctemplate
                                TemplateInfo.plist
```

The Cocoa Application project template can be found at the following location:

`/Developer/Library/Xcode/Templates/Project Templates/Mac/Application/Core Data Spotlight Application.xctemplate`

Copy the entire `Core Data Spotlight Application.xctemplate` folder to the folder: `~/Library/Developer/Xcode/Templates/Project Templates`

Parts of this path may not exist, in which case you'll have to create the `.../Templates/Project Templates` folders.

In the above screenshot the `Developer` folder is named `Developer4`. This is because I installed Xcode 4 into a seperate folder to be able to use both Xcode 3 (residing in `Developer`) and Xcode 4 at the same time. If you changed the installation directory of Xcode from the default `Developer` folder you'll have to keep that in mind when navigating to a `Developer` folder.

Why Core Data Spotlight Application?

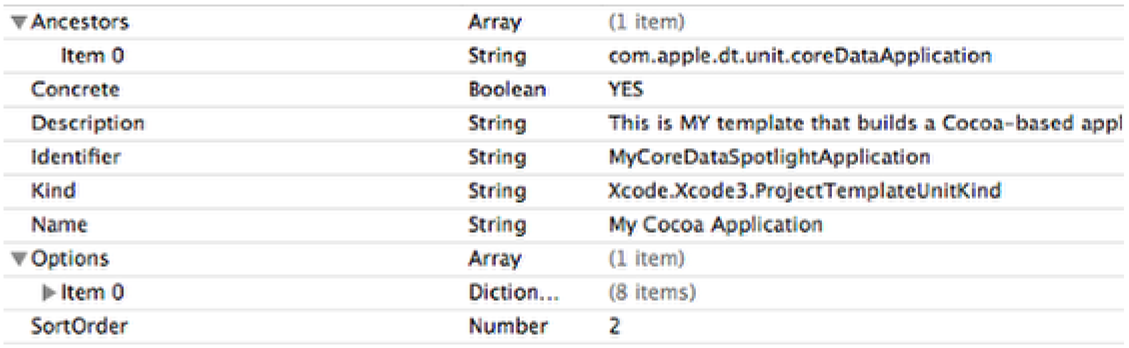
You'll notice that we're not copying the `Cocoa Application.xctemplate` folder. It must seem like the obvious choice but in fact it's only one part of the Cocoa Application template. The Core Data Spotlight Application template is the template that includes the Cocoa Application template via Ancestors.

Here's the inheritance tree that shows you which templates the Core Data Spotlight Application includes:

- Core Data Spotlight Application
  - Core Data Application
    - Cocoa Document-based Application
      - Cocoa Application
        - Mac Base
        - Objective-C Application
          - Bundle Base
            - Base

That means Core Data Spotlight Application offers all the functionality of the above templates. You should use the Core Data Spotlight Application template as basis even if you don't need, for example, core data or a document-based application. The other templates are missing a few crucial settings without which they won't work, so basing a project template only on the Cocoa Data Application template will require additional steps to make it work.

Make it unique!



▼Ancestors	Array	(1 item)
Item 0	String	com.apple.dt.unit.coreDataApplication
Concrete	Boolean	YES
Description	String	This is MY template that builds a Cocoa-based appl
Identifier	String	MyCoreDataSpotlightApplication
Kind	String	Xcode.Xcode3.ProjectTemplateUnitKind
Name	String	My Cocoa Application
▼Options	Array	(1 item)
▶Item 0	Diction...	(8 items)
SortOrder	Number	2

After copying the `Core Data Spotlight Application.xctemplate` folder you **must** do the following in order to create a working copy of the Cocoa Application project template:

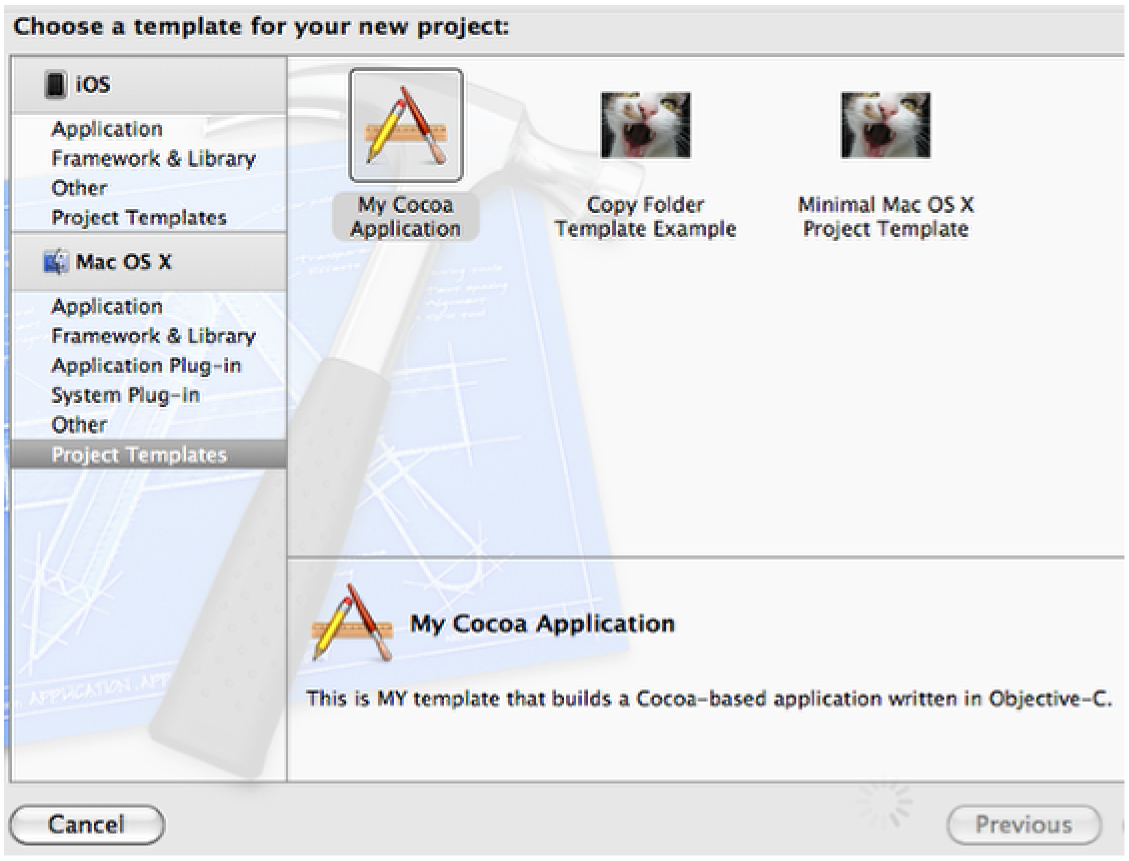
1. rename the folder
2. change the Identifier in TemplateInfo.plist

For #1 you can name the folder `My Core Data Spotlight Application.xctemplate` or something like that but make sure you preserve the folder extension `.xctemplate` without which the template won't show up in the New Project dialog.

For #2 you should open (double-click) the TemplateInfo.plist file and change the Identifier key from `com.apple.dt.unit.coreDataApplication` to anything else - I'm going with `MyCoreDataSpotlightApplication` because the Identifier doesn't really need to be in reverse domain name notation.

These changes are necessary to make sure that the new project is considered a unique, new project template by Xcode 4 and will show up in the New Project dialog. Of course you might also want to modify the Name and Description values to complete the change and have them reflect the purpose of your customized template.

Et voilà!



This is MY template that builds a Cocoa-based application written in Objective-C.

Here's the "My Cocoa Application" template showing up under the "Project Templates" category. If you select it and create a project based on this template you'll end up with exactly the same project as with the original Cocoa Application template. You can now begin with customizing the project template.

Full Control?

You may have noticed that the Core Data Spotlight Application template does not contain any files other than the TemplateInfo.plist file. What if you need to change or remove any of the files the project template will always create?

In this case you want full control over the entire project template. To gain full control over the Cocoa Application template you will have to make copies of all the included project templates listed under each template's Ancestors, or at least those that you might need to modify. Then you'll change the Identifier of each copied template while also changing any occurance of that Identifier in all of the Ancestors arrays.

To be specific, in order to gain full control over the "Cocoa Application" project template you would make a copy of each of these project templates which combined become the "Cocoa Application" project template:

- Core Data Spotlight Application
    - Core Data Application
        - Cocoa Document-based Application
            - Cocoa Application

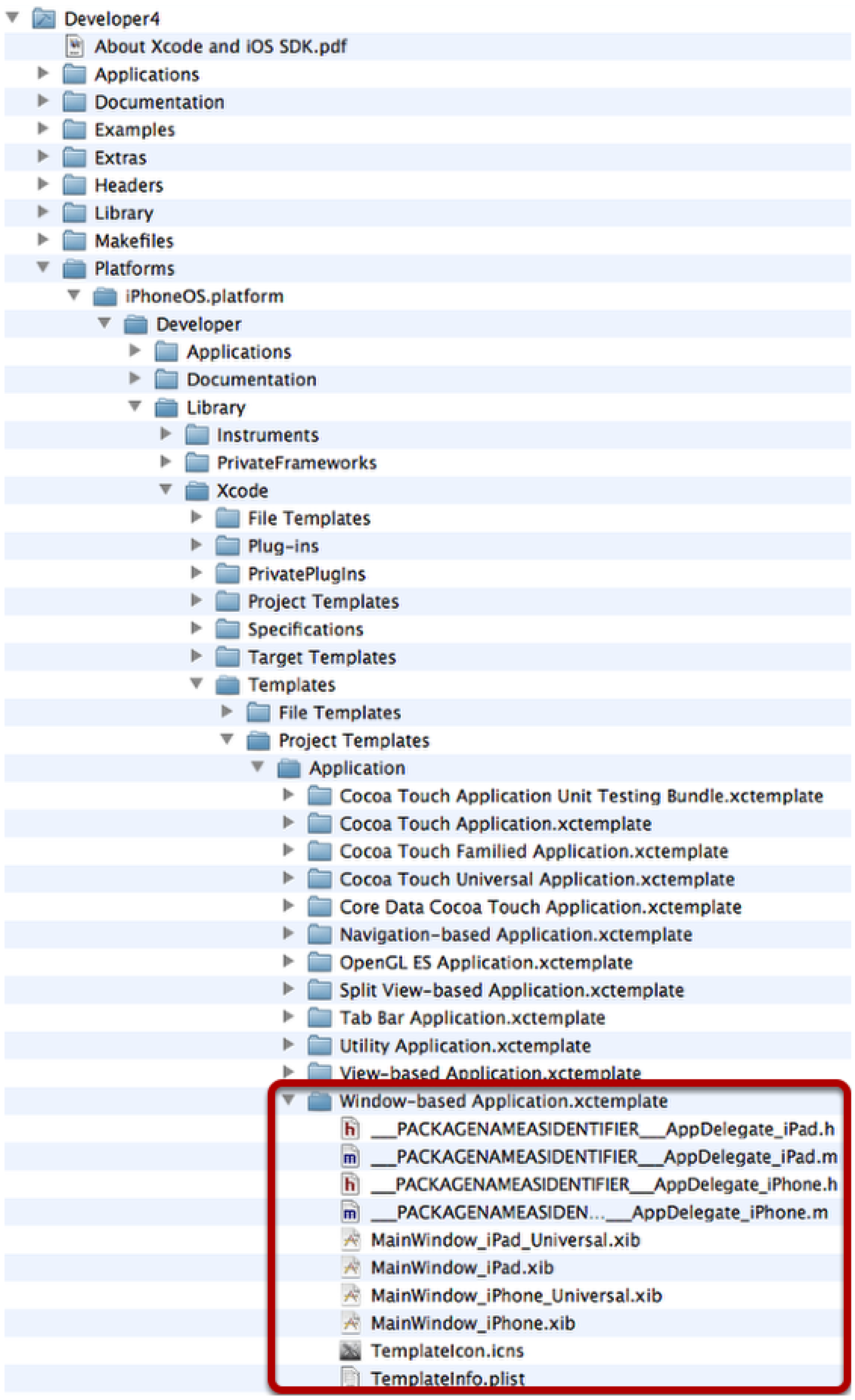
You'll start by changing the Identifier of the Core Data Application template to, say, "MyCoreDataApplication" and you'll also have to change the Ancestors value in the Core Data Spotlight Application template which includes the Core Data Application template so that its value now also reads "MyCoreDataApplication" - this keeps the link between the two project templates intact. You'll then continue on doing the same with the Cocoa Document-based Application and the Cocoa Application template. When you're done you'll have a full copy of the "Cocoa Application" template.

✓ You could do the same with the Mac Base and Objective-C application and possibly even including the Bundle Base and Base templates - but that is likely overkill since these project templates contain both crucial and common settings which you'll rarely need to modify.

Your Cocoa Touch Application Project Template

The most versatile starting point for Cocoa Touch applications is the Window-based Application project template. So let's make a copy of that template.

Copy the project template



```
▼ Developer4
    About Xcode and iOS SDK.pdf
  ▶ Applications
  ▶ Documentation
  ▶ Examples
  ▶ Extras
  ▶ Headers
  ▶ Library
  ▶ Makefiles
  ▼ Platforms
    ▼ iPhoneOS.platform
      ▼ Developer
        ▶ Applications
        ▶ Documentation
        ▼ Library
          ▶ Instruments
          ▶ PrivateFrameworks
          ▼ Xcode
            ▶ File Templates
            ▶ Plug-ins
            ▶ PrivatePlugIns
            ▶ Project Templates
            ▶ Specifications
            ▶ Target Templates
            ▼ Templates
              ▶ File Templates
              ▼ Project Templates
                ▼ Application
                  ▶ Cocoa Touch Application Unit Testing Bundle.xctemplate
                  ▶ Cocoa Touch Application.xctemplate
                  ▶ Cocoa Touch Familied Application.xctemplate
                  ▶ Cocoa Touch Universal Application.xctemplate
                  ▶ Core Data Cocoa Touch Application.xctemplate
                  ▶ Navigation-based Application.xctemplate
                  ▶ OpenGL ES Application.xctemplate
                  ▶ Split View-based Application.xctemplate
                  ▶ Tab Bar Application.xctemplate
                  ▶ Utility Application.xctemplate
                  ▶ View-based Application.xctemplate
                  ▼ Window-based Application.xctemplate
                      ___PACKAGENAMEASIDENTIFIER___AppDelegate_iPad.h
                      ___PACKAGENAMEASIDENTIFIER___AppDelegate_iPad.m
                      ___PACKAGENAMEASIDENTIFIER___AppDelegate_iPhone.h
                      ___PACKAGENAMEASIDEN...___AppDelegate_iPhone.m
                      MainWindow_iPad_Universal.xib
                      MainWindow_iPad.xib
                      MainWindow_iPhone_Universal.xib
                      MainWindow_iPhone.xib
                      TemplateIcon.icns
                      TemplateInfo.plist
```

The Window-based Application project template can be found at the following location:

```
/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Templates
Templates/Application/Window-based Application.xctemplate
```

Notice the difference in location for iOS project templates. Instead of in the /Library/.. folder the templates for iOS are located unter /Platforms/iPhoneOS.platform/..

Copy the entire `Window-based Application.xctemplate` folder to the folder:
`~/Library/Developer/Xcode/Templates/Project Templates`

You do not need to use the /Platforms/iPhoneOS.platform/.. path in the custom templates folder location. Both Mac OS X and iOS templates can reside side-by-side in the same folder.

Parts of this path may not exist, in which case you'll have to create the `.../Templates/Project Templates` folders.

In the above screenshot the `Developer` folder is named `Developer4`. This is because I installed Xcode 4 into a seperate folder to be able to use both Xcode 3 (residing in `Developer`) and Xcode 4 at the same time. If you changed the installation directory of Xcode from the default `Developer` folder you'll have to keep that in mind when navigating to a `Developer` folder.

Why Window-based Application?

The Window-based Application project template provides is the most generic and versatile project templates. It allows you to use Core Data and by default enables the user to also create a universal application. Depending on your project template needs you might want to choose any of the other Cocoa Touch project templates, which are:

- Navigation-based Application
- OpenGL ES Application
- Split View-based Application
- Tab Bar Application
- Utility Application
- View-based Application

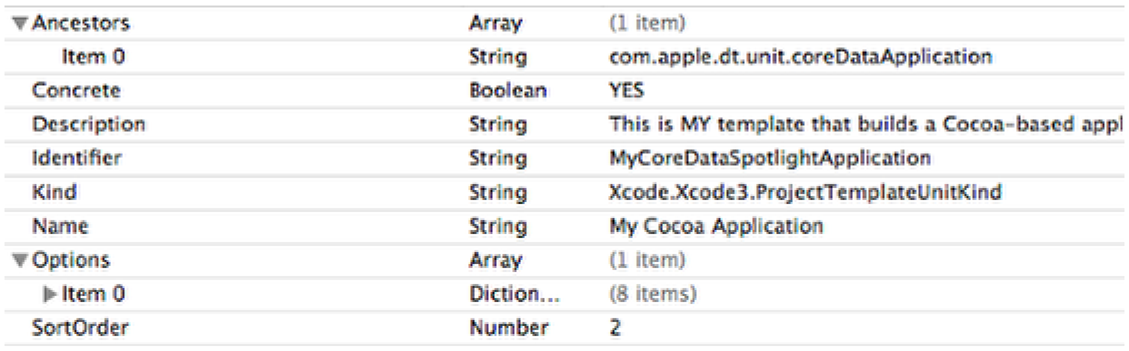
The Window-based Application project template includes via Ancestors both the Core Data Cocoa Touch Application project template, giving it the Core Data options, and the Cocoa Touch Universal Application project template, which provides the option to create either a universal, or an iPad or iPhone specific application.

The inheritance tree for the Window-based Application project template is as follows:

- Window-based Application
    - Core Data Cocoa Touch Application

- Cocoa Touch Universal Application
    - Cocoa Touch Application
        - iPhone Base
        - Objective-C Application
            - Bundle Base
                - Base

Make it unique!



▼Ancestors	Array	(1 item)
Item 0	String	com.apple.dt.unit.coreDataApplication
Concrete	Boolean	YES
Description	String	This is MY template that builds a Cocoa-based appl
Identifier	String	MyCoreDataSpotlightApplication
Kind	String	Xcode.Xcode3.ProjectTemplateUnitKind
Name	String	My Cocoa Application
▼Options	Array	(1 item)
▶Item 0	Diction...	(8 items)
SortOrder	Number	2

After copying the `Window-based Application.xctemplate` folder you **must** do the following in order to create a working copy of the Window-based Application project template:

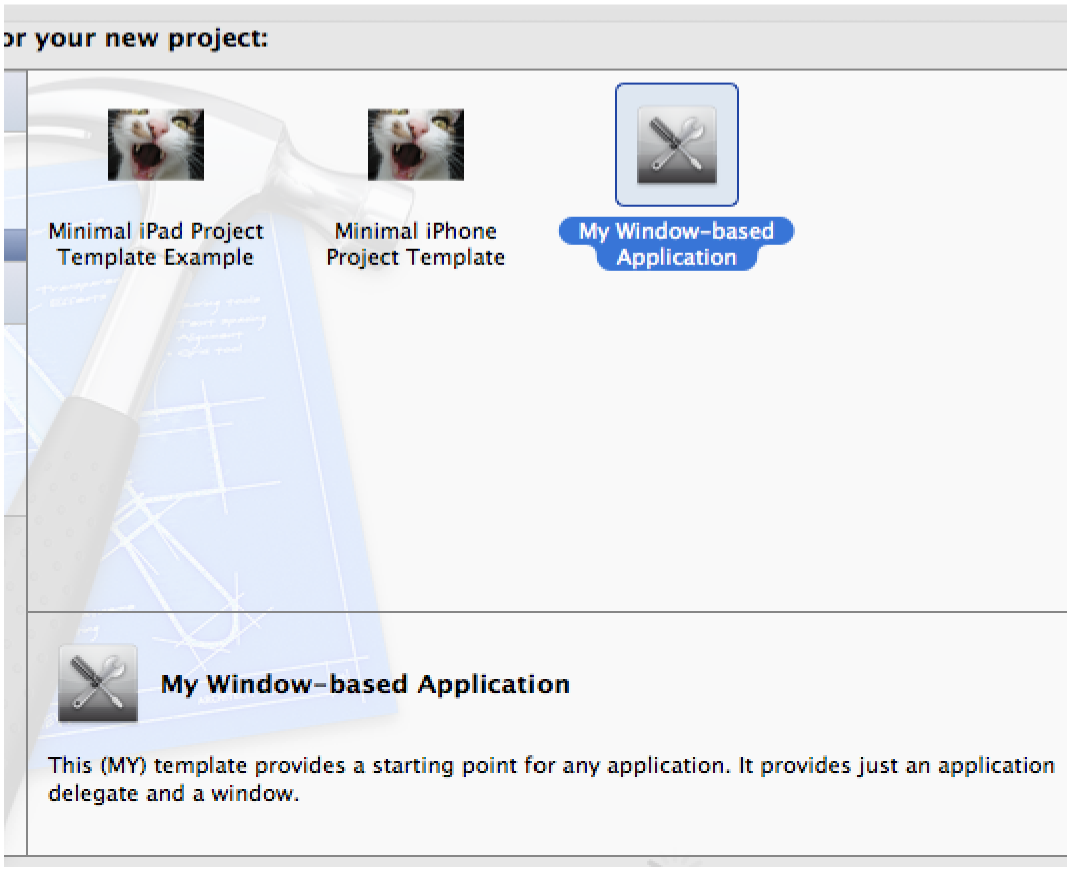
1. rename the folder
2. change the Identifier in TemplateInfo.plist

For #1 you can name the folder `My Window-based Application.xctemplate` or something like that but make sure you preserve the folder extension `.xctemplate` without which the template won't show up in the New Project dialog.

For #2 you should open (double-click) the TemplateInfo.plist file and change the Identifier key from `com.apple.dt.unit.navigationBasedApplication` to anything else - I'm going with `MyNavigationBasedApplication` because the Identifier doesn't really need to be in reverse domain name notation.

You might also want to modify the Name and Description values.

Et voilà!


or your new project:
Minimal iPad Project Template Example
Minimal iPhone Project Template
My Window-based Application

My Window-based Application

This (MY) template provides a starting point for any application. It provides just an application delegate and a window.

Your copy of the Window-based Application project template appears in the New Project dialog. It works as normal and is ready for tweaking.

Full Control?

As with the Your Cocoa Application Project Template you can gain full control over the Window-based Application project template by making copies of all the templates listed under each template's Ancestors. Then change the Identifier of each copied template while also changing any occurance of that Identifier in any of the Ancestors arrays.

To be specific, in order to gain full control you would make a copy of each of these project templates:

- Window-based Application
  - Core Data Cocoa Touch Application
  - Cocoa Touch Universal Application
    - Cocoa Touch Application

If you have made a copy of a different project template, chances are that it includes the Cocoa Touch Familied Application project template instead of Core Data Cocoa Touch Application and Coco Touch Universal Application.

You'll start by changing the Identifier of the Core Data Cocoa Touch Application project template to, say, "MyCoreDataCocoaTouchApplication" and you'll also have to change the Ancestors value in the Window-based Application project template which includes the Core Data Cocoa Touch Application project template so that its value now also reads "MyCoreDataCocoaTouchApplication" - this keeps the link between the two project templates intact. You'll then continue on doing the same with the Cocoa Touch Universal Application and the Cocoa Touch Application project template. When you're done you'll have a full copy of the Window-based Application template.

✓ If you wanted to you could do the same all the way up from iPhone Base respectively iPad Base and Objective-C application to Bundle Base and Base project templates - but that is likely overkill since these project templates contain both crucial and common settings which you'll rarely need to modify.

Xcode 4 Templates Reference

This section contains technical references for the contents of the files and folders involved in creating Xcode Project and File templates.

Contents of TemplateInfo.plist

The following key-value entries are found at the root level of a TemplateInfo.plist file. The tables differentiate between keys which can appear in both Project and File Templates, followed by keys only seen in either Project or File templates. While you can try using Project Template keys in File Templates and vice versa, keep in mind that they will likely not work or not provide all options. For example, while you can use Options in File Templates you can not use all of the UI elements like popup or combo boxes with File Templates.

Table entries are sorted alphabetically by key. Each key is described shortly, and most keys are hyperlinked to sections that contain more information on that particular key. If a key is not hyperlinked, it is either self-explanatory or it has no use or immediate effect and thus remained undocumented.

Items used by both Project and File Templates

Key	Value Description	Value Type
Description	Description text for the template. In Project Templates the Description need only be used in TemplateInfo.plist which have the Concrete flag set.	string
Kind	Describes the type of Xcode template. This is a required key and must not be omitted.	string
Options	List of options for user input. This configures the "Choose options for your new project" page and usually gives you options to enter the Product Name and Company Identifier.	array

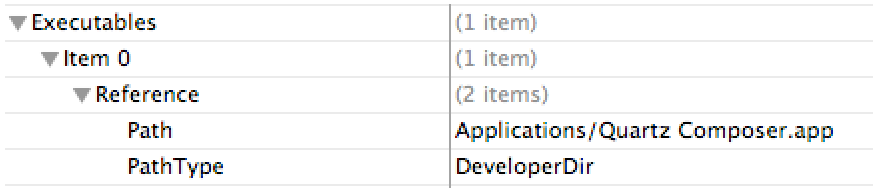
Platforms	Determines in which platform category (iOS or Mac OS X) the template shows up. If not used, the template will show up for both platforms by default.	array
SortOrder	Used to override the default alphabetical sort order of templates in the New File and New Project dialogs.	number

Items Specific to File Templates

Key	Value Description	Value Type
AllowedTypes	Specifies the default and all allowed file extension(s) for the file.	array
DefaultCompletionName	The filename (without extension) that Xcode initially suggests.	string
Icon	Used only by User Interface XIB templates - is always "TemplateIcon.tiff".	string
MainTemplateFile	Which of the template files Xcode should open and display after creating the file(s).	string
Summary	Does not seem to be used by Xcode, could be (reserved for) a tooltip. Usually identical to the Description except that the sentence doesn't end with a punctuation character. In a few cases it's an abbreviated Description.	string
Title	Used only by User Interface XIB templates - does not seem to be utilized.	string

Items Specific to Project Templates

Key	Value Description
Ancestors	A list of items with the value part being the Identifier of another TemplateInfo.plist file. Via ancestors you can include other project templates and take over their settings, or override them. Works similar to inheritance in OOP.
Concrete	Marks the Project Template as the one to be listed in the New Project dialog. If not set, the TemplateInfo.plist will be assumed to be included as ancestor by another TemplateInfo.plist and it will not be shown in the New Project dialog.
Definitions	Defines the content of the Nodes entry with the same name (eg if a Definitions key equals a Nodes value). Normally it is used to indicate where a file is (Path) but can also specify the actual text content of the file.

Executables	Only used by the Quartz Composer Plug-In template. Specifies the path to the Quartz Composer.app which indicates that you can somehow specify the dependency to an external app. The effect of this setting is unclear. For reference:  ▼Executables (1 item) ▼Item 0 (1 item) ▼Reference (2 items) Path Applications/Quartz Composer.app PathType DeveloperDir
Identifier	Unique Identifier for this template, using reverse domain notation (com.yourcompany.thistemplatefilename).
Name	The name of the project template as shown in the New Project dialog. If omitted the project template will assume the name of the .xctemplate folder.
Nodes	Lists all the files in the project. Can also contain names of placeholders or functions to add to a file.
NSSupportsSuddenTermination	Only used by the Quartz Composer Plug-In template, where it is unchecked. Effect unclear.
Project	Configure Project-Wide Build Configuration settings.
ProjectOnly	Indicates that this template should only be listed in the New Project dialog, but not when adding a new target. Used only by the "Empty" project template.
TargetOnly	Indicates that this template should only be listed in the Add Target dialog, but not when creating a New Project. Used for example by the "Aggregate" target template.
Targets	Allows you to add targets to the project and configure Target-specific settings.

AllowedTypes

Value Type:	array
Can be used in:	✓ File Templates ✗ Project Templates

A list of allowed UniformTypeIdentifiers (UTI) for a file template's MainTemplateFile. The value determines the default file extension that is appended by Xcode.

For example, the Objective-C class template specifies these allowed types: `public.objective-c-source` and `public.objective-c-plus-plus-source`. This means the source file can either be Objective-C (.m extension) or Objective-C++ (.mm extension), with .m being the default (first item in the list). If the user enters a file name with a file extension that is not an Objective-C file extension, for example if the user enters "MyFile.cpp" as filename, the resulting file will be named "MyFile.cpp.m". If multiple UTIs are used but none of the allowed file extensions are used, the extension of the first UTI in the AllowedTypes array will be appended.

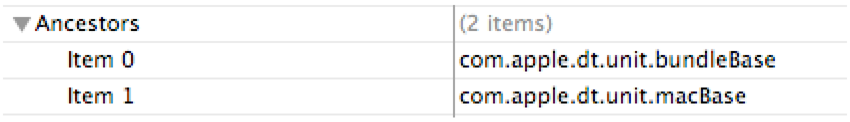
The Following table lists some commonly used UTIs by File Templates. If you want the complete list, refer to Apple's Uniform Type Identifiers Reference.

UniformTypeIdentifier	Description
public.c-header	C header file (.h)
public.c-source	C source file (.c)
public.c-plus-plus-source	C++ source file (.cpp)
public.objective-c-source	Objective-C source file (.m)
public.objective-c-plus-plus-source	Objective-C++ source file (.mm)
...	...

Ancestors

Value Type:	array
Can be used in:	✗ File Templates ✓ Project Templates

The Ancestors array allows you to "subclass" other Project Templates by specifying the Identifier of one or more TemplateInfo.plist in the Value part of each Item. See the following screenshot:



▼Ancestors	(2 items)
Item 0	com.apple.dt.unit.bundleBase
Item 1	com.apple.dt.unit.macBase

The information provided by the Ancestor templates will be included in the current template, unless a specific setting is already specified here. In this case the ancestor's setting will be ignored (overridden).

⚠ Some keys, like Kind can not be inherited and have to be specified in every TemplateInfo.plist file.

In general Ancestors are used to define commonly used settings in template files that are then included by all templates which use the same settings. The higher up in the Ancestor hierarchy a template is (up to the template which has the Concrete flag set) the more specific the settings usually get. For example, you'll usually only find the Description of a template in the template which also has the Concrete flag set.

Useful Ancestors

Below is an incomplete list of existing Ancestors provided by Xcode which should be used in your own Project Templates because they provide crucial default settings and options.

Type of Application

Include either one of these Ancestors if your project is either an Objective-C application, or an Objective-C unit test bundle. These options are mutually exclusive and should not be used together.

Ancestor Template Identifier	Description
com.apple.dt.unit.objectiveCApplication	Adds default settings and options for Objective-C applications. Inherits from the `com.apple.dt.unit.bundleBase` and `com.apple.dt.unit.base` templates. Should be used for all project templates that interface with Objective-C even if all code is written in C or C++ due to the fact that working with Apple's libraries will always make it an Objective-C application.
com.apple.dt.unit.cocoaTouchApplication	Adds default settings and options for Cocoa Touch applications. Inherits from the `com.apple.dt.unit.objectiveCApplication` and `com.apple.dt.unit.iPhoneBase` templates.
com.apple.dt.unit.cocoaTouchFamiliedApplication	Adds the device family popup option with which the user can select from creating an iPhone or iPad application when creating a new project. Inherits from the `com.apple.dt.unit.cocoaTouchApplication` template.
com.apple.dt.unit.cocoaTouchUniversalApplication	Overrides the device family popup option with which the user can select from iPhone, iPad or Universal when creating a new project. Inherits from the `com.apple.dt.unit.cocoaTouchFamiliedApplication` template.

⚠ Including the Objective-C application template `com.apple.dt.unit.objectiveCApplication` either directly or indirectly is formally required because it adds important settings without which project templates won't work.

Target Platform

Depending on the desired Target Platform you should included one of these project templates in your custom project template which for the most part modify build settings for the desired target platform. These options are mutually exclusive and should not be used together.

Ancestor Template Identifier	Description

com.apple.dt.unit.macBase	Include this template if the target platform for the resulting project is a Mac OS X application.
com.apple.dt.unit.iPhoneBase	Include this template if the target platform for the resulting project is a iPhone/iPod Touch application (non universal). You can always add iPad support via Xcode at a later time.
com.apple.dt.unit.iPadBase	Include this template if the target platform for the resulting project should be a universal application with iPad support. This template inherits from the `com.apple.dt.unit.iPhoneBase` above and changes the TARGETED_DEVICE_FAMILY build setting to 2 and sets the ARCHS build setting to $(ARCHS_UNIVERSAL_IPHONE_OS).

Concrete

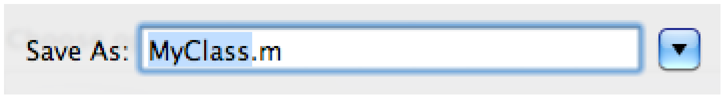
Value Type:	boolean
Can be used in:	✗ File Templates ✓ Project Templates

This flag is set in TemplateInfo.plist which should be listed in the New Project dialog. It marks the last TemplateInfo.plist in the Ancestors hierarchy. Project Templates without the Concrete flag or with Concrete set to NO will not be listed in the New Project dialog.

DefaultCompletionName

Value Type:	string
Can be used in:	✓ File Templates ✗ Project Templates

Specifies the default filename (without extension) that Xcode suggests when the file is created. It's the highlighted part in the following screenshot showing the part of the "save as" dialog that pops up as the final step of creating a file template:



Definitions

Value Type:	array
Can be used in:	✗ File Templates ✓ Project Templates

Definitions can fulfill several purposes depending on their use. With Definitions you can:

- define beginning/end placeholder strings and indentation of replacement strings
- replacement strings for specific variables or wildcards
- define the location of existing files and the group they should reside in in the Groups & Files

pane
- copy contents of entire folders

Each Definition key is the name of a Nodes string (value). If there is no corresponding Nodes entry for a particular key, that key will be ignored.

Definitions Parameters

The default for a Definitions key/value pair is that the value data type is a string. In this case, the string will be added to the file specified by the key. The Definition in this case is merely holding the string to be added to the file. In the common case that the key is a dictionary, there will be the same key specifying the string to add to the file, but it'll be inside the dictionary.

The following dictionary keys can be used:

Key	Description	Value Type
Beginning	The string added at the beginning of a placeholder.	string
End	The string added at the end of a placeholder.	string
Indent	The indentation level (in "tabs") of strings added in between the Beginning and End section. Eg. this determines the indentation level of placeholder strings.	string
Group	The name (and path) of the Xcode Group the file should be added to.	string
Group	If Group is an array, you can specify subgroups by creating an array item for each group. For example, to put a file into the group path `/Classes/MyCode/Tests/` you would add in the same order three array items: `Classes`, `MyCode` and `Tests` without the slashes.	array
TargetIndices	You can specify which target the file should be added to. Targets are numbered sequentially beginning with 0. By default each file will be added to each target in the project. Specifying an empty TargetIndices array prevents the file from being added to all targets. This is done for all files which are not supposed to be listed in the "Compile Sources" Build Phase, for example all header files should have an empty TargetIndices array.	array
SubstituteMacros	Effect unknown.	boolean
Path	The path to an existing file, relative to the .xctemplate folder of the Concrete TemplateInfo.plist file.	string

PathType	Affects the files' Location property in Xcode, which by default is "Relative to Group". Must only be used together with the Path key, otherwise Xcode will crash when creating the template. Valid values are: • `Absolute` (Location: Absolute Path) • `DeveloperDir` (Location: Relative to Developer Directory) • `Group` (default) (Location: Relative to Group) • `Project` (Location: Relative to SDK) ⚠ Whether other PathType values are supported is unknown. The usual suspects like BuiltProducts, BuiltProductsDir, BuildProducts, etc. have been tested and did not work.	string

With Definitions, you can also create new variables. Refer to the Variable Placeholders section for more information.

Example Entries

Empty File

Key	Value	Data Type
MyFile.m	// This is an empty file	string

This will create a new file named MyFile.m and the the file will have a single line reading `//This is an empty file`.
Despite the fact that the Property List editors only give you a single line to edit strings in you can actually create and use multi-line strings.

Adding strings to a File

Key	Value	Data Type
MyFile.m:anything	// Anything goes	string
MyFile.m:something	// This is quite something	string

In this case, the strings `// This is quite something` and `// Anything goes` will be added to MyFile.m in the order in which they are defined in the Nodes section. If the Nodes section specifies MyFile.m:something before (item number is smaller) the entry MyFile:anything then the final MyFile.m contents will be:

```
// This is quite something
// Anything goes
```

Add Beginning/End sections with indented content

Key	Value	Data Type
MyFile.m		dictionary
Beginning	// This is the beginning	string
End	// This is the end	string
Indent	2	number

If this definition is specified along with the above, then the resulting MyFile.m contents will be:

```
// This is the beginning
// This is quite something
// Anything goes
// This is the end
```

The Beginning and End keys specify the start and end of the MyFile.m which could also be a particular section of MyFile.m. For example you can also create additional Beginning/End/Indent sections for MyFile.m:something and MyFile.m:anything.

Everything in between Beginning and End is indented by the number of tabs specified by the Indent key. In this case the text in between is indented two times.

Placing files into groups

Key	Value	Data Type
MyFolder/SomeFile.m		dictionary
Path	MyFolder/SomeFile.m	string
Group	MyGroup/SomeFile.m	string

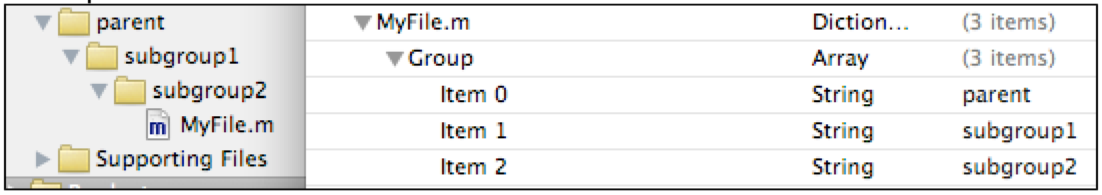
The above allows you to specify the location of an existing file as indicated by the Path key. The Group key determines the Group location where the file should be placed in Xcode's Groups & Files pane.

Placing files into subgroups

Key	Value	Data Type
MyFolder/SomeFile.m		dictionary
Path	MyFolder/SomeFile.m	string
Group	..	array

Item 0	parentGroup	string
Item 1	subGroup1	string
Item 2	subGroup2	string

Example:



- parent
  - subgroup1
    - subgroup2
      - MyFile.m
- Supporting Files

MyFile.m	Diction...	(3 items)
Group	Array	(3 items)
Item 0	String	parent
Item 1	String	subgroup1
Item 2	String	subgroup2

To put a file into subgroups, you'll have to change the Data Type of Group to array and add each part of the path as seperate items. This will place the file into the Group path `/parentGroup/subGroup1/subGroup2/`.

Copy Entire Folder

Copying folders works, but …
… there are two significant flaws:

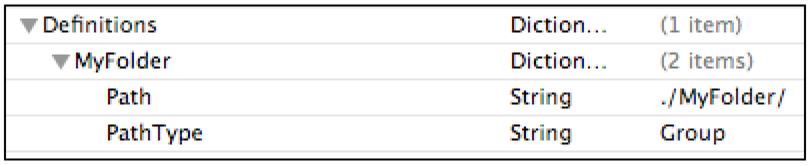
- files in the folder and subfolders will be added to the Copy Bundle Resources phase, not to the Compile Sources phase
- the name of the last file in the folder will become a folder reference in Xcode, listing again all the files & folders

Thus far all attempts at fixing or working around this behavior has remained futile. Please let me know if you have been able to figure out if this is possible, and how. Some developers' projects contain hundreds of files which makes creating project templates a tedious and error-prone task.

My suspicion is that if there's a solution it might be connected to a particular but currently unknown PathType. But it could as well be a dead-end.

Key	**Value**	**Data Type**
MyFolder/		dictionary
Path	./MyFolder	string
PathType	Group	string

Example:



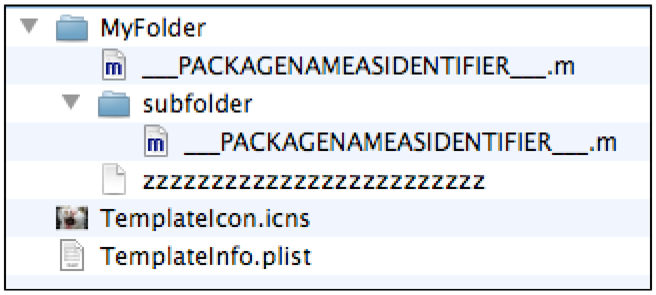
▼ Definitions	Diction…	(1 item)
▼ MyFolder	Diction…	(2 items)
Path	String	./MyFolder/
PathType	String	Group

In this example I used `./MyFolder/` as the Path. But you can specify the path in a number of ways all of which are legal and give the same results:

- `./MyFolder/` (preferred style, it makes the clearest indication that Path refers to a folder and not a file)
- `MyFolder/`
- `./MyFolder`
- `MyFolder`

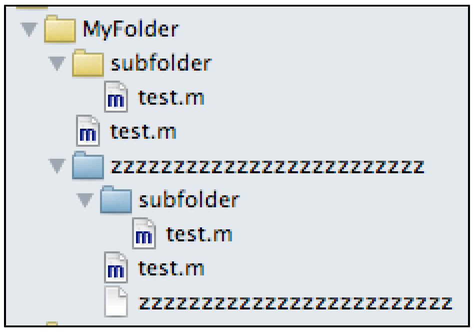
The latter two examples assume that there is no file named `MyFolder` as these Paths could refer to either a file or a folder.

Project Template files & folders:

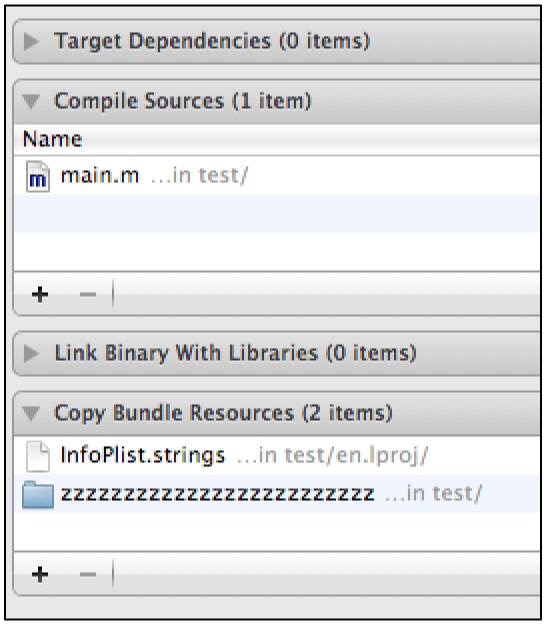


```
▼ MyFolder
      ___PACKAGENAMEASIDENTIFIER___.m
  ▼ subfolder
        ___PACKAGENAMEASIDENTIFIER___.m
      zzzzzzzzzzzzzzzzzzzzzzzzz
  TemplateIcon.icns
  TemplateInfo.plist
```

Result:



```
▼ MyFolder
  ▼ subfolder
      test.m
    test.m
  ▼ zzzzzzzzzzzzzzzzzzzzzzzzz
    ▼ subfolder
        test.m
      test.m
      zzzzzzzzzzzzzzzzzzzzzzzzz
```


Target Dependencies (0 items)
Compile Sources (1 item)
Name
main.m ...in test/
+ −
Link Binary With Libraries (0 items)
Copy Bundle Resources (2 items)
InfoPlist.strings ...in test/en.lproj/
zzzzzzzzzzzzzzzzzzzzzzzz ...in test/
+ −

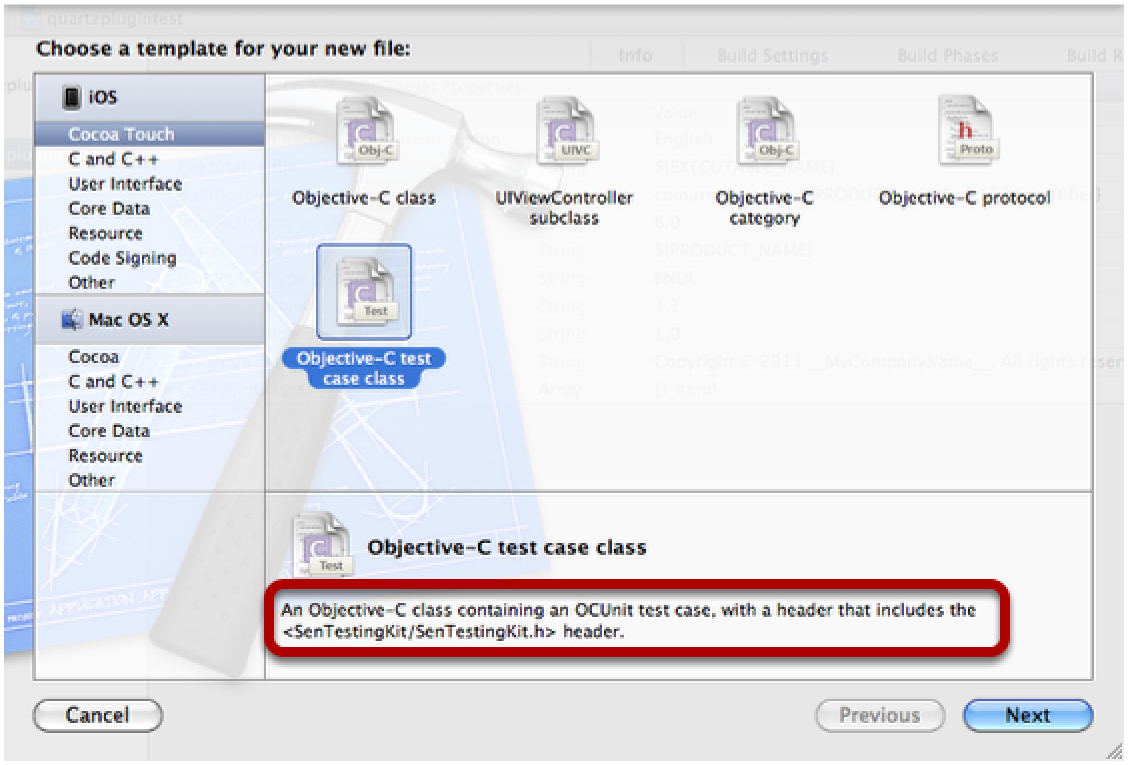
This will copy all the files in the project template subfolder `MyFolder/` to the new project and add the files to the group `MyGroup/MyFolder`.

✓ I've provided an example template named "Copy Folder Template Example" that you might want to check out if you're interested in investigating the copy-folder behavior.

Description

Value Type:	string
Can be used in:	✓ File Templates ✓ Project Templates

✓ In Project Templates, the Description only needs to be specified in concrete templates. Meaning those TemplateInfo.plist which have the Concrete flag set.



An Objective-C class containing an OCUnit test case, with a header that includes the <SenTestingKit/SenTestingKit.h> header.

The description of a template is highlighted in the above screenshot.

Identifier

Value Type:	string
Can be used in:	✗ File Templates ✓ Project Templates

The Identifier is used to refer to a specific TemplateInfo.plist file by reference, usually to subclass from it using the Ancestors array. The Identifier must be unique and is written in reverse domain notation. For example: `com.yourcompany.yourtemplate.templatename`

⚠ Identifier is also used to identify Options where reverse domain notation is not normally used. As long as the Identifier string is unique it does not matter what that string is.

Identifier is mandatory!
Every TemplateInfo.plist for Project Templates must have a unique Identifier key, otherwise Xcode might crash when trying to open the New Project dialog.

Kind

Value Type:	string

Can be used in:	✓ File Templates ✓ Project Templates

Kind is a required key, it must be in every TemplateInfo.plist file. It is not inherited by including other TemplateInfo.plist files via the Ancestors key. In fact, omitting the Kind key or setting it incorrectly can lead to Project Templates not showing up in the New Project dialog or crashes when creating a Project Template with a missing or incorrect Kind key in any of its TemplateInfo.plist files, including Ancestors.

Possible Values	Description
Xcode.Xcode3.ProjectTemplateUnitKind	Template is a Project Template
Xcode.IDEKit.TextSubstitutionFileTemplateKind	Template is a File Template
Xcode.IDECoreDataModeler.ManagedObjectTemplateKind	Template is a File Template for an NSManagedObject data model. The main difference is that you will be presented with a "Select the data models with entities you would like to manage" screen. Used by the CoreData NSManagedObject subclass File Template.

⚠ If Kind is missing in just one of the TemplateInfo.plist files of a template (it is not inherited!) the template will not be listed in the New File / New Project dialog!

⚠ All Xcode 4 Project Templates use the `Xcode.Xcode3.ProjectTemplateUnitKind` key despite its reference to Xcode 3. This is not a typo.

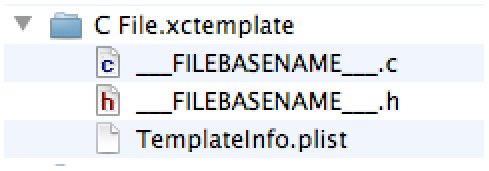
MainTemplateFile

Value Type:	string
Can be used in:	✓ File Templates ✗ Project Templates

With MainTemplateFile you specify the file which should be opened in Xcode after a File has been created from a template, since a template can have more than one file. The MainTemplateFile key can also change the icon that is used to display the File Template in the New File dialog. For example, if the MainTemplateFile points to a header file with the .h extension, an icon with a red "h" will be used as the File Template's icon.

The MainTemplateFile key is optional and without it Xcode will still copy, rename and add all the files in the .xctemplate folder to the project.

The MainTemplateFile specifies the filename of a file in the template's .xctemplate folder. For example, for the C File template the MainTemplateFile is `___FILEBASENAME___.c`


C File.xctemplate
___FILEBASENAME___.c
___FILEBASENAME___.h
TemplateInfo.plist

Nodes

Value Type:	array
Can be used in:	✗ File Templates ✓ Project Templates

In the Nodes array you define filenames and placeholders for content section of files. The Definitions section then determines where to place the files in the new project and what content should be added to or replaced in the file. Both go hand in hand. Without a Definitions counterpart, the Nodes entries will only create empty files.

✓ **Copying entire folders**
You can also copy the contents of entire folders instead of just single files. This seems to work reasonably well with bundle resource files, but not with source code. Refer to Definitions#CopyFolder for more information.

Nodes values

Usually a Nodes value will be a filename. The filename can include the relative path to the file if the file is in a subfolder or should be created in a subfolder.

Placeholders are specified by appending a colon (:) followed by the name of the placeholder. Multiple placeholders can be used in a single Nodes entry. This is usually done to create variables for placeholders. Refer to the Variable Placeholders section for more information.

Example Entries

Key	Value
Item 0	MyFile.m

The value `MyFile.m` will be used in Definitions as a key in order to specify various parameters for that particular file, including location, group and file contents.

Key	Value
Item 0	MyFile.m:something

A node that has a colon (:) following the filename indicates the use of placeholders. The

Definitions section will then define the content for the placeholder `something`. This is usually used to add content to newly created files.

Multiple colons can be used to define multiple placeholders.

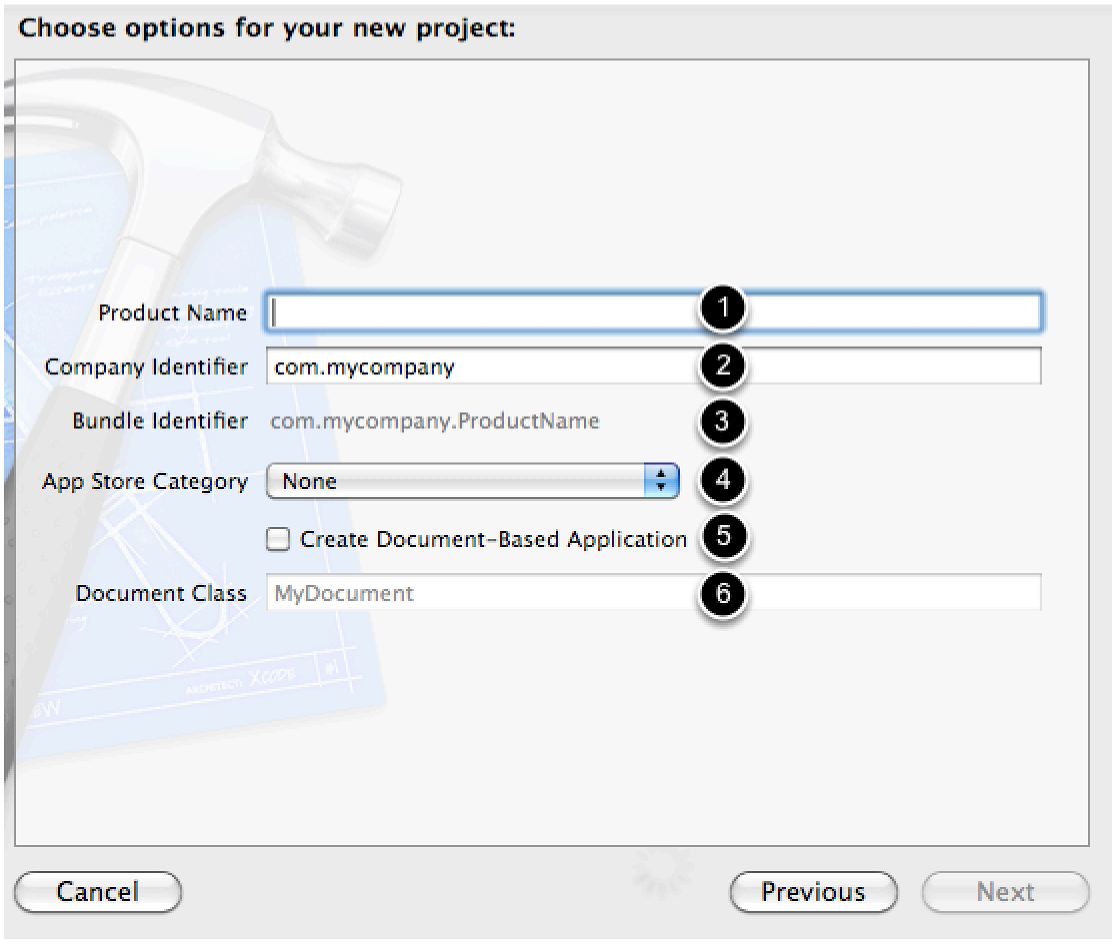
Options

Value Type:	array
Can be used in:	✓ File Templates ✓ Project Templates

Options determine the various options the user can configure in the first step of creating a new File or Project from a template. Common options include an input field for the Project or File name.

✓ If no Options are specified, the "Choose Options for ..." step will not be presented to the user.

Below is an example screenshot of various options (six in this case) that can be presented to the user:



Choose options for your new project:

Product Name

Company Identifier: com.mycompany

Bundle Identifier: com.mycompany.ProductName

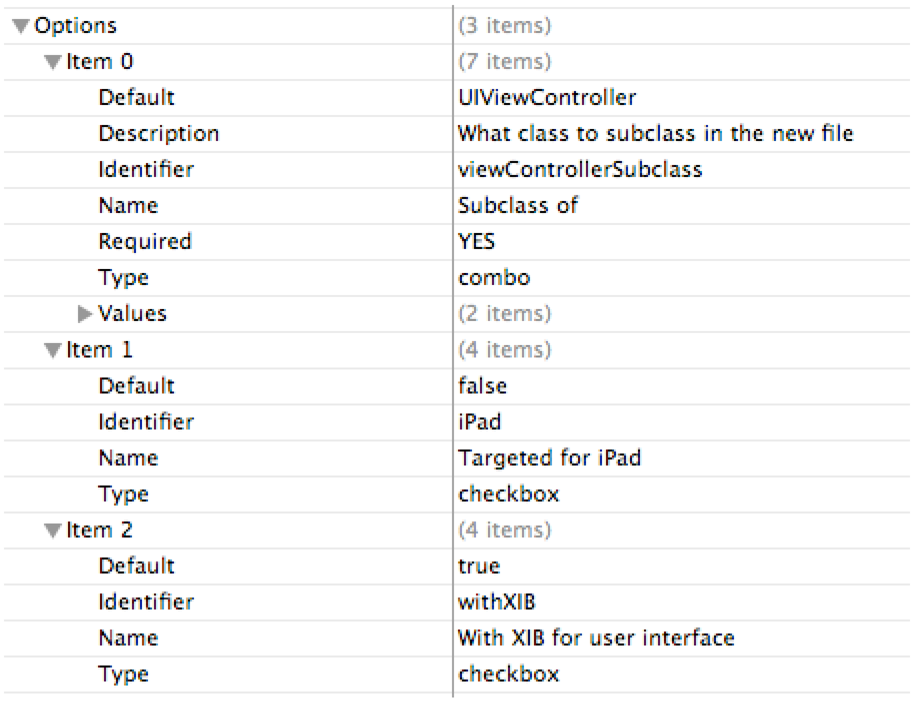
App Store Category: None

Create Document-Based Application

Document Class: MyDocument

1. An editable text box.
2. An editable text box with default text.
3. A static label for information. The label's text is dynamically composed of the Product Name and Company Identifier using the Variable Placeholders syntax.
4. A popup selection. Combo boxes which also allow arbitrary text input are also possible.
5. A checkbox.
6. A text box with default text which is currently not editable (grayed out). It is connected to the checkbox above and only editable if the checkbox is checked.

Each item in the Options array is a dictionary. Below is an example Options array defining three options:



▼ Options	(3 items)
▼ Item 0	(7 items)
Default	UIViewController
Description	What class to subclass in the new file
Identifier	viewControllerSubclass
Name	Subclass of
Required	YES
Type	combo
▶ Values	(2 items)
▼ Item 1	(4 items)
Default	false
Identifier	iPad
Name	Targeted for iPad
Type	checkbox
▼ Item 2	(4 items)
Default	true
Identifier	withXIB
Name	With XIB for user interface
Type	checkbox

The dictionary keys for each item are described next.

Options used by both File and Project Templates

Key	Value Description	Value Type
Default	The default selection or text. In case of a checkbox type use either "true" or "false". In case of a static label type it's the uneditable text that is displayed. The default value can also specify other options using the Variable Placeholders syntax.	string
Description	Describes what this option does. The text is displayed as tooltip when the mouse hovers over the option.	string
Identifier	Identifiers are used to uniquely identify options, for example to use them for other options or referencing the option's content in variables.	string
Name	The text displayed left to the control.	string

Required	If checked, this option is required and must be filled in by the user. As long as a Required option does not have a valid value the Next button on the "Choose Options …" dialog will be grayed out.	boolean
SortOrder	The SortOrder for this option. Can be used to rearrange the order in which options are presented to the user. By default they appear in the same order as their position in the Options array.	number
Type	The type of edit control used, eg text box or combo box.	string

Options Specific to File Templates

Key	Value Description	Value Type
Values	The selectable options for a combo box or popup control. See also Type.	array

Options Specific to Project Templates

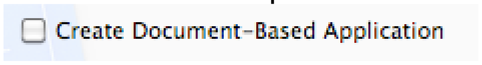
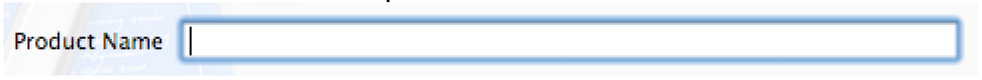
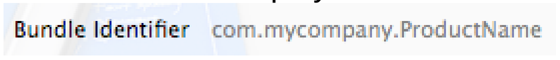
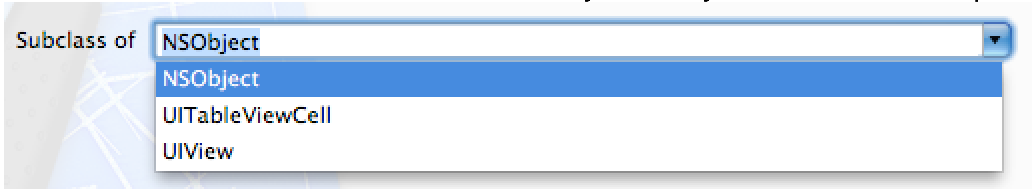
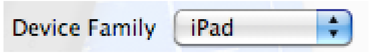
Key	Value Description	Value Type
EmptyReplacement	The string with which an editable text box is prefilled (ie the default text). Does not always do its job for some reason. In case of combo or popup boxes it should match one of the Values strings.	string
NotPersisted	Unknown, no effect? Used with ProductName option.	boolean
Placeholder	Used only in one of Apple's Project Templates, effect unknown.	string
RequiredOptions	Can be used to specify that a seperate, boolean option must be true or false before the regular Options in this TemplateInfo.plist can be changed, otherwise they will be greyed out. The key specifies an Identifier of an option while the key's value is either true or false, indicating that the option is only editable if the required option is set to the corresponding truth value.	dictionary
SpecialType	Effect unknown, only use was to provide the string `LSApplicationCategoryType`	string
Units	Each Units key has the same name as the Values of a popup or combo box and can contain any settings that would normally be in the root of the TemplateInfo.plist. Depending on the active popup selection, the specific Units branch is assumed to be active and the others are ignored. This allows you to use only specific TemplateInfo.plist settings based on a user choice.	dictionary

Project Templates require an option with the Identifier 'productName'
All Project Templates require one particular option: a text field with Identifier set to `productName`. Otherwise the Project Template might fail to save, or irreversibly delete all files in the target folder or from the user's desktop!

The `productName` option is provided by including in Ancestors the `com.apple.dt.unit.base` Template or one of the templates including it, for example the often used `com.apple.dt.unit.objectiveCApplication` base template for Objective-C applications. Rather than adding the `productName` option yourself you should rely on using Apple's base templates.

Options keys

Type

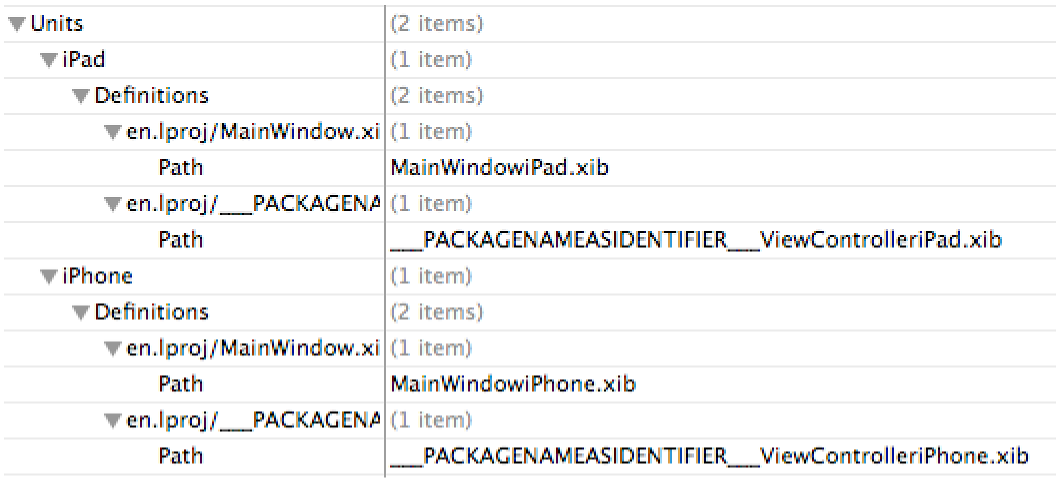
Valid Types	Description
checkbox	A checkbox. Example: 
text	An editable text box. Example: 
static	A static label to display information. Example:  Bundle Identifier com.mycompany.ProductName
combo	Creates a drop-down combo box with an editable text field. The user can pick one of the preconfigured items from the Values list, or enter a custom string. The screenshot shows the combo box used by the Objective-C class template:  Subclass of NSObject NSObject UITableViewCell UIView
popup	Creates a popup selection control. The user can only pick from the preconfigured items from the Values list. 

Not all options described in the Options reference are available in File Templates. In fact, only the text and static Types options work flawlessly with File Templates.

Units

Allows you to specify different TemplateInfo.plist items depending on a user selection.

In the example screenshot below there are two popup Values named iPad and iPhone. Assuming the user has selected the iPad setting, the contents of the iPad dictionary can be assumed to be added to the root of the TemplateInfo.plist thus adding two new Definitions entries which essentially modify the filename of two files to reflect the iPad setting.



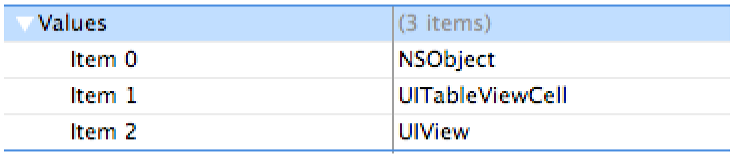
▼Units	(2 items)
▼iPad	(1 item)
▼Definitions	(2 items)
▼en.lproj/MainWindow.xi	(1 item)
Path	MainWindowiPad.xib
▼en.lproj/___PACKAGENA	(1 item)
Path	___PACKAGENAMEASIDENTIFIER___ViewControlleriPad.xib
▼iPhone	(1 item)
▼Definitions	(2 items)
▼en.lproj/MainWindow.xi	(1 item)
Path	MainWindowiPhone.xib
▼en.lproj/___PACKAGENA	(1 item)
Path	___PACKAGENAMEASIDENTIFIER___ViewControlleriPhone.xib

You are not limited to using Definitions in Units, you can actually add any of the available TemplateInfo.plist settings to each individual Units' selection dictionary (iPad and iPhone in this case).

Values

A list of strings with which a combo box or popup will be filled (see Type). The Values strings will be selectable by the user. The values can be used as keys for the Units dictionary to specify settings that are used when that particular setting is selected by the user.

Here is an example Values array used by the Objective-C class template. It gives the user the option to subclass his new Objective-C class from the following preconfigured classes:



▼Values	(3 items)
Item 0	NSObject
Item 1	UITableViewCell
Item 2	UIView

Platforms

Value Type:	array
Can be used in:	✓ File Templates ✓ Project Templates

If Platforms is not specified in the TemplateInfo.plist, the template will appear for all platforms by default. Since there are currently only two platforms, the Platforms usually specifies only one platform in order to exclude the template from showing up for the other platform.

Possible Values	Description
com.apple.platform.iphoneos	Template is listed in the iOS platform categories
com.apple.platform.macosx	Template is listed in the Mac OS X platform categories

⚠ The Platforms key does not automatically change any build settings that are needed to build code for a certain platform. To do that, you will have to use the Configurations and SharedSettings dictionaries found under the Project and Targets root keys.

✓ For Project Templates you can also use the existing, platform-specific templates that Apple provides. To do so, add an item to the Ancestors and specify one of these Identifiers for Mac, iPhone and iPad specific platforms:

- com.apple.dt.unit.macBase
- com.apple.dt.unit.iPhoneBase
- com.apple.dt.unit.iPadBase

Including the above templates will also change build settings and other options so that your project template conforms to the standard template for that particular platform.

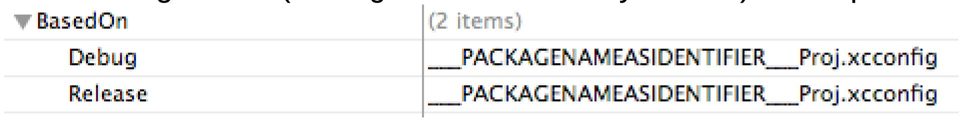
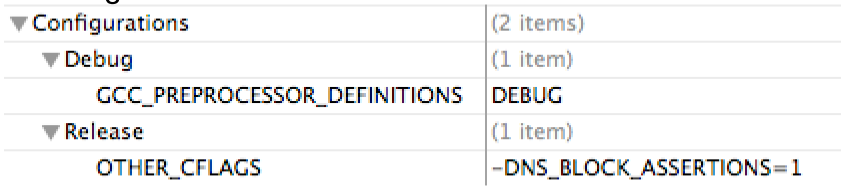
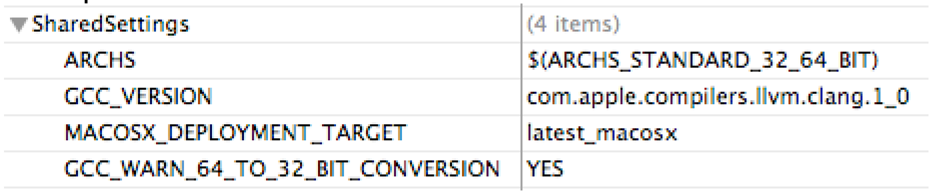
Project

Value Type:	dictionary
Can be used in:	✗ File Templates ✓ Project Templates

Project allows you to specify build settings for the project. There are a few predefined keys, which allow you to specify XCConfig files (BasedOn), create build configurations and specify any build settings for each particular build configuration (Configurations), and finally you can specify build settings which should be applied to all build configurations (SharedSettings). The special SDK key allows you to specify which Base SDK should be used.

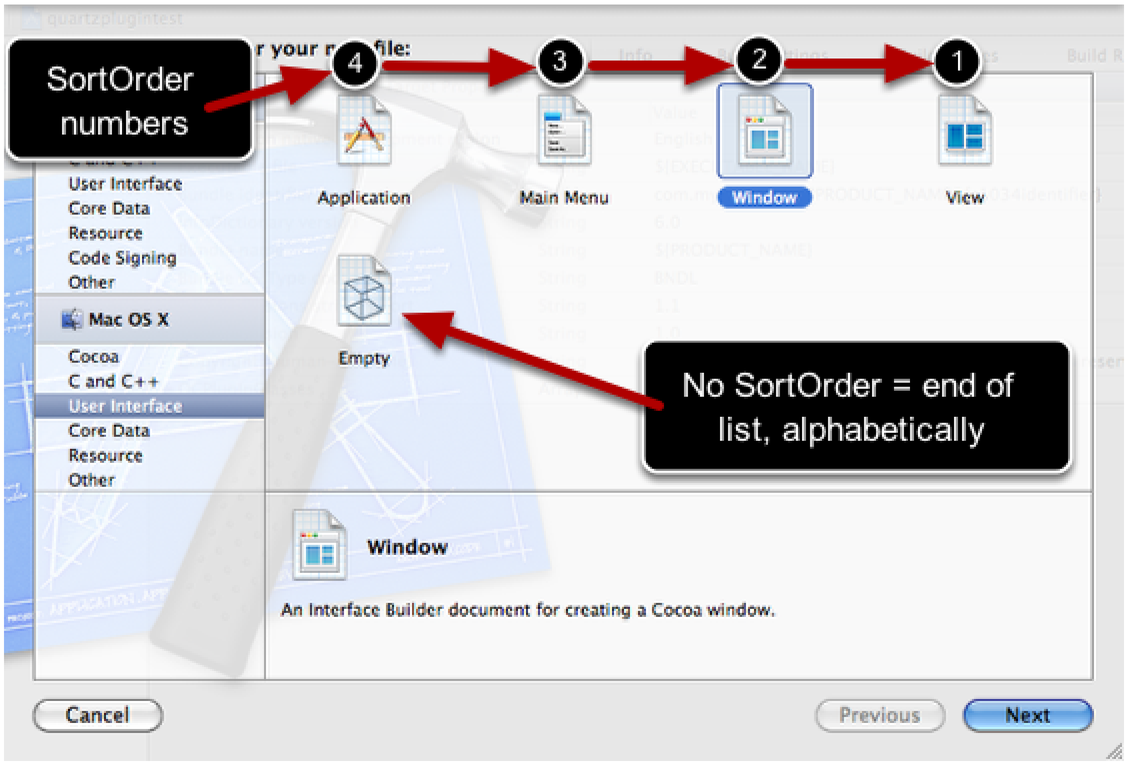
✓ You might want to check out Apple's Xcode/GCC Build Settings Reference for a complete list of build settings.

Key	Description	Value Type

SDK	The name of the Base SDK used by this project. Can be either `iphoneos` or `macosx` for iOS respectively Mac OS X projects.	string
BasedOn	Allows you to specify any .xcconfig files in your template. XCConfig files contain Build Settings and you can specify one file for each build Configuration (`Debug` and `Release` by default). Example:  BasedOn (2 items) Debug: ___PACKAGENAMEASIDENTIFIER___Proj.xcconfig Release: ___PACKAGENAMEASIDENTIFIER___Proj.xcconfig	dictionary
Configurations	Allows you to specify the Build Configurations in your project. Most projects have a `Debug` and `Release` build configuration, but you can add more. Each build configuration is also a dictionary, in which you can specify build settings. The key is the GCC setting and the value the value for that particular setting. For example, in this screenshot the Debug build configuration will have the DEBUG preprocessor configuration set while the Release build configuration specifies in "Other Linker Flags" the proper setting for blocking assertions:  Configurations (2 items) Debug (1 item) GCC_PREPROCESSOR_DEFINITIONS: DEBUG Release (1 item) OTHER_CFLAGS: -DNS_BLOCK_ASSERTIONS=1	dictionary
SharedSettings	Much like Configurations, except that you don't specify a build configuration (eg `Debug` or `Release`) since the shared build settings naturally will be used by all build configurations. Here is an example:  SharedSettings (4 items) ARCHS: $(ARCHS_STANDARD_32_64_BIT) GCC_VERSION: com.apple.compilers.llvm.clang.1_0 MACOSX_DEPLOYMENT_TARGET: latest_macosx GCC_WARN_64_TO_32_BIT_CONVERSION: YES	dictionary

SortOrder

Value Type:	number
Can be used in:	✓ File Templates ✓ Project Templates



The above screenshot illustrates how the SortOrder item works. Items are sorted in decending order of the SortOrder. Items without a SortOrder are sorted alphabetically and listed after any items with a SortOrder.

For example, to add a new template so it is listed as the first item, even before the "Application" template, you would have to give it a SortOrder of 5.

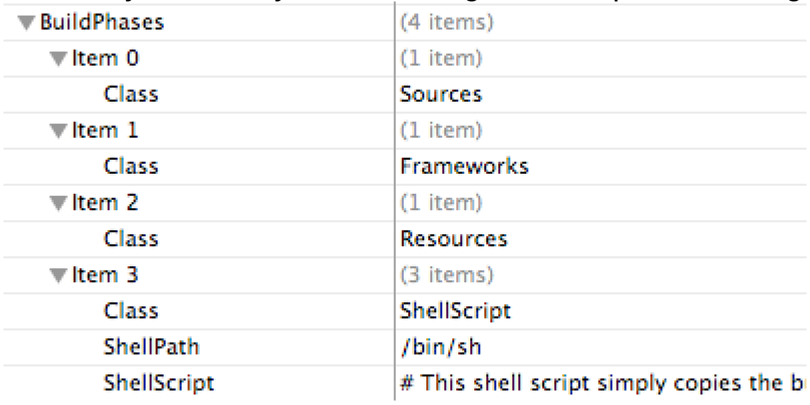
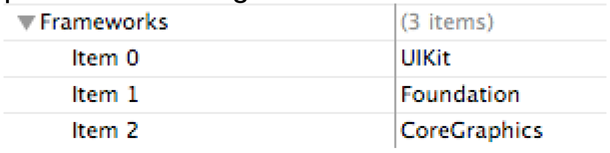
If you would add another template titled "Application enhanced" without a SortOrder, it would be listed before the "Empty" template because templates without a SortOrder are sorted alphabetically.

⚠ Keep in mind that the SortOrder is reversed, eg. the sort order goes from highest SortOrder number to lowest which may be confusing initially. If you notice that templates are sorted incorrectly, make sure the SortOrder is from highest to lowest, not vice versa.

Targets

Value Type:	array
Can be used in:	✗ File Templates ✓ Project Templates

Key	**Description**	**Value Type**

BasedOn	For each build configuration you can specify which XCConfig file the target's build configuration is based on. See the same setting in Project.	dictionary
BuildPhases	List of build phases, the order of the items in the array determines the order of the Build Phases. Each item is a dictionary in which you can configure build phase settings.  BuildPhases (4 items) Item 0 (1 item) Class Sources Item 1 (1 item) Class Frameworks Item 2 (1 item) Class Resources Item 3 (3 items) Class ShellScript ShellPath /bin/sh ShellScript # This shell script simply copies the b	array
BuildToolArgsString	Used only by External Build System template to specify build tool arguments.	string
BuildToolPath	Used only by External Build System template to specify the build tool path from the user options.	string
Configurations	Target-specific build settings for particular build configurations. See the same setting in Project.	dictionary
Dependencies	An array of numbers. Most likely specifies a Targets entry by its index.	array
Frameworks	Names of frameworks (without the .framework extension) which will be added to the "Link Binary With Libraries" build phase of this target.  Frameworks (3 items) Item 0 UIKit Item 1 Foundation Item 2 CoreGraphics	array
Name	The name of the target, the default setting is the placeholder ___PACKAGENAME___	string
OtherFrameworks	Same as Frameworks, but the frameworks are not added to the "Link Binary With Libraries" build phase of the target. In the Groups & Files pane these frameworks will be placed in the `/Frameworks/Other Frameworks` group.	array
ProductDependencies	An array of numbers. Most likely specifies a Targets entry by its index.	array

ProductType	Specifies the type of resulting product, these are the available options which should be self-explanatory: com.apple.product-type.application com.apple.product-type.bundle com.apple.product-type.framework com.apple.product-type.kernel-extension com.apple.product-type.library.dynamic com.apple.product-type.library.static com.apple.product-type.tool	string
SharedSettings	Target-specific build settings for all build configurations. See the same setting in Project.	dictionary
TargetType	Used only by Aggregate target and External Build System templates. Possible values: • Aggregate • Legacy	string

Dependencies
What Dependencies and ProductDependencies specify is unclear. Whenever they were used, there was only one item in the array with the value being 0. They're used in unit test templates, so this might indicate a dependency on the project to be tested.

BuildPhases

Possible Build Phase Keys	Description	Value Type
Class	Possible values for Class: • CopyFiles • Frameworks • Headers • Resources • Rez • ShellScript • Sources Rez is the "Build Carbon Resources" build phase. All other build phases should be self-explanatory.	string

DstPath	If Class is CopyFiles, specifies the destination path of the copy files build phase.	string
DstSubfolderSpec	Used if Class is CopyFiles. Unclear what it does, only used in Command Line tool project template, value is 0. Might be used to change the value in the Destination popup box of the Copy Files build phase, but using values other than 0 did not change anything.	number
RunOnlyForDeploymentPostprocessing	If Class is CopyFiles, specifies that this build phase is only run for deployment postprocessing. The value is either YES or NO.	string
ShellPath	If Class is ShellScript, specifies the path to the shell script interpreter, for example `/bin/sh`	string
ShellScript	If Class is ShellScript, this setting is the shell script that will be run.	string

Template Folder Locations

Templates are found in two distinct locations on your hard drive.

Location of Apple Templates

The official Xcode templates that are installed with Xcode reside in two locations:

Mac OS X / General Templates folder:

```
/Developer/Library/Xcode/Templates/
```

iOS Specific Templates folder:

```
/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Templates
```

`/Developer` (at the beginning of the path) is the default installation location of Xcode. If you have installed Xcode to a different folder, replace `/Developer` with whatever path you installed Xcode.

In the iOS tree, in the `../Xcode/` folder you will also find several "File Templates" and other "xxx Templates" folders. They seem to be remnants of Xcode 3 as they contain templates from Xcode 3. Please ignore these folders and make sure you are inside the `../Xcode/Templates/` folder and none other - it's easy to get confused when you're navigating the iOS templates. You'll notice when you're dealing with Xcode 3 templates if they contain TemplateChooser.plist files and the folder names don't end with `.xctemplate`.

Location for User Templates

To prevent a new Xcode installation from overwriting your own templates, there's a custom template folder for each user. It is generally recommended to store all modified or personally customized templates in the User Templates folder.

Custom User Templates folder:

```
~/Library/Developer/Xcode/Templates/
```

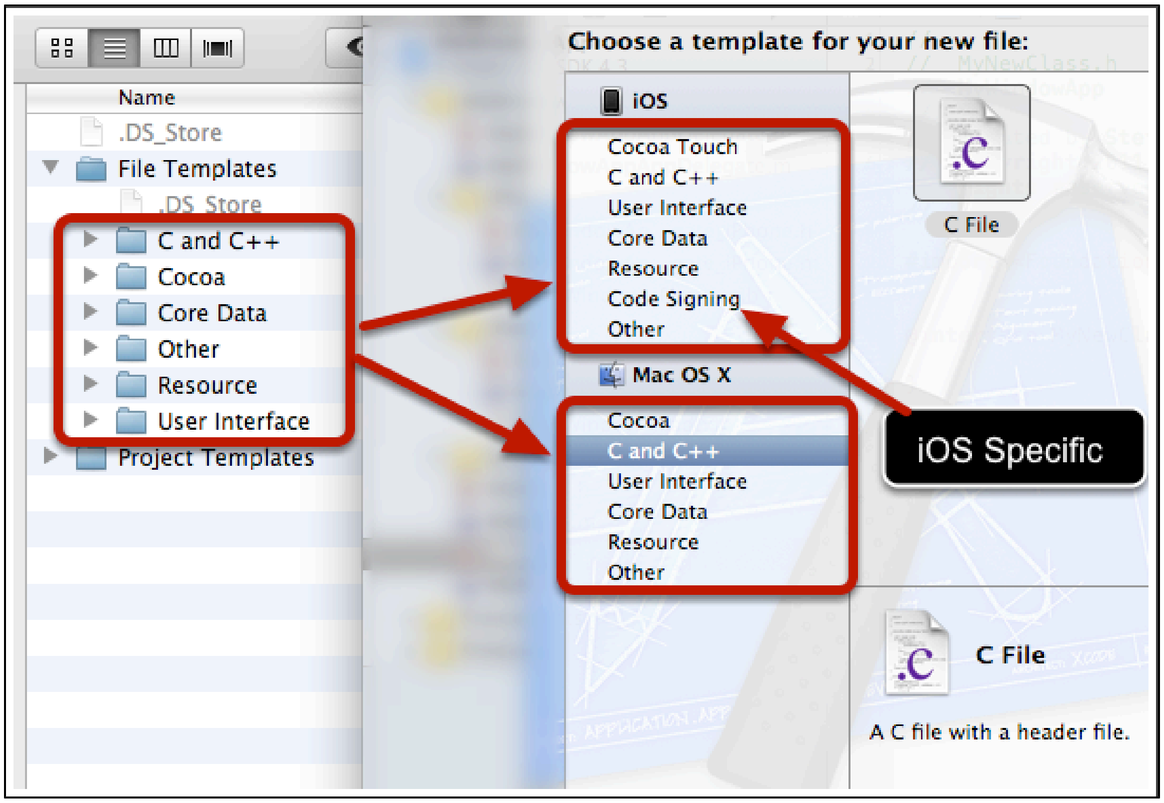
You may have to create the `Templates` folder as well as the `File Templates` and `Project Templates` subfolders if they do not exist. This is usually the case, unless some other development software has created these folders for you.

Despite the official Mac OS X and iOS templates being located at very different folders you can have both Mac OS X and iOS templates in the same custom user template folder.

Subfolders & Categories

In each Templates folder are two main subfolders `File Templates` and `Project Templates` differentiating between file and project templates. It is not strictly necessary to make this distinction, these two folders are indeed optional. But it is highly recommended to stick to the convention in case this is somehow important to how templates behave and generally to avoid confusion.

In both `File Templates` and `Project Templates` folders are further subfolders without the .xctemplate extension but containing .xctemplate folders. Such folders define the categories on the left hand side of the New File and New Project dialogs, as seen in the screenshot below. Each subfolder containing .xctemplate folders is listed as a category in the New File / New Project dialog on the left hand side of the dialog.



The platform categories iOS and Mac OS X are provided by Xcode. They are likely hardcoded and can't be changed.

You'll notice that the same categories can exist for both platforms. You'll also see that a platform can add its specific category which only appears for one platform, in this case "Code Signing" only exists for iOS templates because this folder only contains project templates for iOS. To specify which platform a project template supports you would use the Platforms key.

Placeholder Reference

This is a reference of all predefined placeholder strings which Xcode will replace with actual content when a template is created. These placeholder strings can be used in any file that a template creates and adds to the project, mostly in source code files but XML and text-based resource files will also have the placeholder strings replaced.

Note that placeholders can also be used in filenames of files in file and project templates.

Placeholder	Description
___DATE___	The current date. Uses the NSCalendarData format string "%x".
___YEAR___	The current year, in 4 digits.

___TIME___	The current time. Uses the NSCalendarData format string "%X".
___ORGANIZATIONNAME___	The name of your Organization as set by you. Refer to Change your Organization Name for more info.
___FILENAME___	The file name with extension.
___FILEBASENAME___	The file name without extension.
___FILEBASENAMEASIDENTIFIER___	The file name without extension. Any character that is illegal in C variable names (eg # + etc.) is replaced with an underscore.
___FILEEXTENSION___	The file's extension without the dot.
___USERNAME___	The login (short) user name of the currently logged in user.
___FULLUSERNAME___	The full (not: login) user name of the currently logged in user.
___PROJECTNAME___	The name of the current project.
___PROJECTNAMEASIDENTIFIER___	The name of the current project. Any character that is illegal in C variable names (eg # + etc.) is replaced with an underscore.
___PROJECTNAMEASXML___	The name of the current project. Special characters are properly escaped to create a valid XML string, eg < and > are replaced with < and &rt; respectively.
___PACKAGENAME___	The name of the current package. Usually identical to project name.
___PACKAGENAMEASIDENTIFIER___	The name of the current package. Usually identical to project name. Any character that is illegal in C variable names (eg # + etc.) is replaced with an underscore.
___PACKAGENAMEASXML___	The name of the current package. Usually identical to project name. Special characters are properly escaped to create a valid XML string, eg < and > are replaced with < and &rt; respectively.
___UUID___	A universally unique identifier (UUID).
___UUIDASIDENTIFIER___	A universally unique identifier (UUID). Any character that is illegal in C variable names (eg # + etc.) is replaced with an underscore.

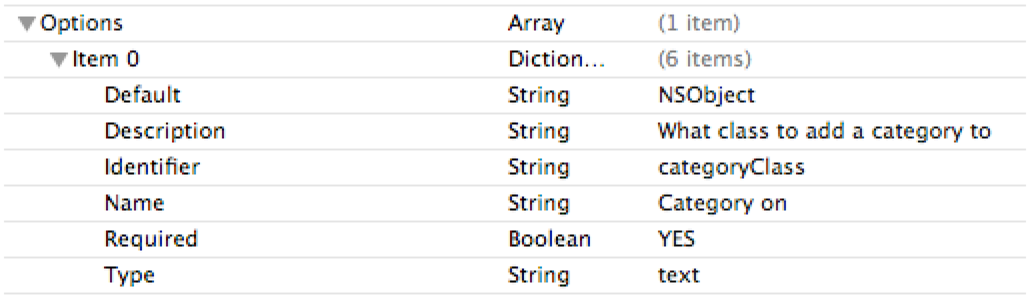
___VARIABLE_identifier___	Replaced with a custom string defined by the corresponding identifier of a Definitions entry or an Option. The identifier can be followed by a colon and a special keyword to modify the string. See Variable Placeholders for more information.

Variable Placeholders

This section explains how you can create custom placeholders to be used with the ___VARIABLE_identifier___ placeholder as well as the Definitions specific ___*___ placeholder.

Options Variables

Options variables are placeholders following the syntax ___VARIABLE_identifier___ whereas `identifier` is the identifier used by a particular Options entry. A typical example is an option that allows the user to specify the name of the subclass of an Objective-C class file template:



▼Options	Array	(1 item)
▼Item 0	Diction...	(6 items)
Default	String	NSObject
Description	String	What class to add a category to
Identifier	String	categoryClass
Name	String	Category on
Required	Boolean	YES
Type	String	text

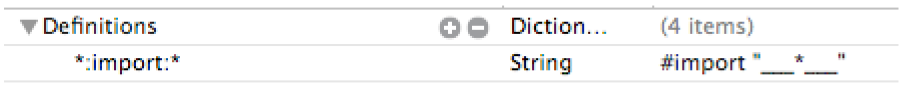
The identifier of this option is `categoryClass` so you can use the placeholder ___VARIABLE_categoryClass___ in your files in order to replace it with the user's input or selection of the categoryClass option.

✓ The identifier part of an Options placeholder variable can be appended by special keywords to further modify it. Currently there are only three known modifiers `identifier`, `bundleIdentifier` and `RFC1034Identifier`:

- ___VARIABLE_identifier:identifier___
  - ensures that a legal C-style variable name is created, eg replaces illegal characters with underscore
- ___VARIABLE_identifier:bundleIdentifier___
  - ensures that a legal bundle identifier is created
- ___VARIABLE_identifier:RFC1034Identifier___
  - ensures a legal RFC1034 aka domain name identifier is created

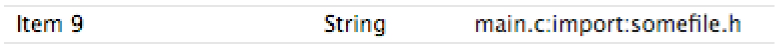
Definitions Variables

Similarly, you can specify variable placeholders in a Definitions entry by using the ___*___ syntax:



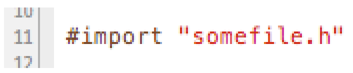
▼ Definitions	Diction…	(4 items)
:import:	String	#import "___*___"

The key is `*:import:*` which means it applies to any file that is followed by the variable `import` which in turn is followed by another variable (in this example: `somefile.h` after that. The second star (asterisk) becomes the actual variable that can be used in the definition's value as a placeholder with the ___*___ syntax. That means ___*___ will be replaced with the string following `import`. This can be used in the Nodes section as follows:



Item 9	String	main.c:import:somefile.h

You'll see that the file is `main.c` and that it is followed by the `import` variable already defined in the Definitions section. Following that is the string `somefile.h` which will result in the following line being added to main.c when the template is created:



```
#import "somefile.h"
```

The ___*___ syntax only works within the Definition section of a TemplateInfo.plist. It can not be used as a regular placeholder in files.

Variables can be nested multiple times. For example, you can redefine the variable (key) as `*:import:special:*` and in the Nodes section you then have to write it as `main.c:import:special:somefile.h` to achieve the same effect. This kind of namespace is helpful to group related variables together.

Commonly Used Definitions Variables

The default Xcode templates define some commonly used variables which you might want to use for your own project. They also serve as great examples for how the variable placeholders work and what you can do with them.

Definition key	Definition Value
:import:	`#import "___*___"`
Example Node	**Result**

main.c:import:somefile.h	`#import "somefile.h"`

Definition key	Definition Value

*:comments	`//`
	`//  ___FILENAME___`
	`//  ___PACKAGENAME___`
	`//`
	`//  Created by ___FULLUSERNAME___ on ___DATE___.`
	`//  Copyright ___YEAR___ ___ORGANIZATIONNAME___. All rights reserved.`
	`//`

Example Node	**Result**

main.c:comments	`//`
	`//  main.c`
	`//  cmdtool`
	`//`
	`//  Created by Steffen Itterheim on 01.04.11.`
	`//  Copyright 2011 __MyCompanyName__. All rights reserved.`
	`//`

Definition key	Definition Value
::importFoundation	`#import <Foundation/Foundation.h>`
Example Node	**Result**
MyClass.h:anything:importFoundation	`#import <Foundation/Foundation.h>`

Definition key	Definition Value

:class:	`@class ___*___;`
Example Node	**Result**
MyClass.h:class:MyClass	`@class MyClass;`

Definition key	**Definition Value**
:dealloc:	`[___*___ release];`
Example Node	**Result**
MyClass.m:dealloc and MyClass.m:dealloc:myViewController	`- (void)dealloc { [myViewController release]; [super dealloc]; }`

Definition key	**Definition Value**
*:init	Beginning: `- (id)init { self = [super init]; if (self) {` End: `} return self; }`

Example Node	Result

MyClass.m:init	`- (id)init`
	`{`
	`    self =`
	`[super`
	`init];`
	`    if`
	`(self) {`
	`    }`
	`    return`
	`self;`
	`}`

Definition key	Definition Value
:synthesize:	`@synthesize ___*___;`
Example Node	**Result**

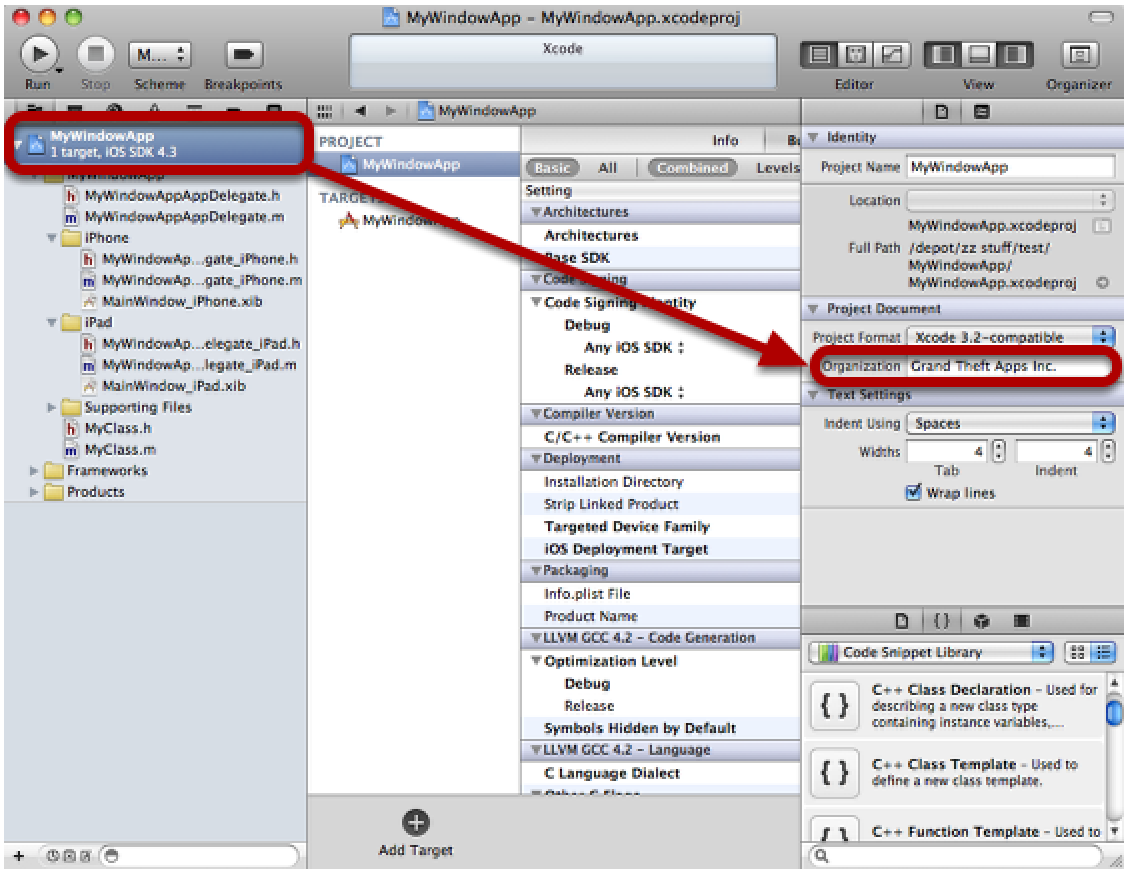
MyClass.h:synthesize:myVariable=myVariable_	`@synthesize`
	`myVariable=myVariable_`

Related Information

Change your Organization Name

Many File Templates contain a placeholder that reads ___MyCompanyName___ even after the file was created. It is supposed to be replaced by the name of your organization, but only if the organization name is set. By default the organization name is empty.

In Xcode 4 you can change your organization name for each project. To do so, in the Project Navigator list click on the Project itself (first item, blue icon).



Project Name: MyWindowApp
Full Path: /depot/zz stuff/test/MyWindowApp/MyWindowApp.xcodeproj
Project Format: Xcode 3.2-compatible
Organization: Grand Theft Apps Inc.
Indent Using: Spaces
Widths: 4 4

With the File Inspector open, you can enter your organization name in the appropriately named field. Now whenever you create a new file in this project, the ___MyCompanyName___ string will be replaced with your organization name.

✓ If the File Navigator panel on the right does not show up, in the Xcode menu go to View -> Utilities -> File Inspector to show the File Inspector panel.

✓ An alternative way to globally set your Organization Name is to open the Address Book app, select your own Card and edit the card. You can then enter your company name which will be used by Xcode from then on.

⚠ Changing the organization name will **not** automatically change your organization name in already existing files. You'll have to do this manually using search & replace. It is recommended to change your organization name at the start of a project.

Property List Editors

If you're not familiar with Property Lists (.plist files) it helps to understand the different kinds of data types that can be stored using the key/value pairs used by Property Lists. You can find a Property List data types overview on Wikipedia.

For Xcode 4 Templates the basic data types are boolean, number and string. It should be fairly obvious how they behave. The more interesting data types are arrays and dictionary. In a Property List these data types allow you to create deeply nested collections of data. Each array and dictionary can contain any other data type, including arrays and dictionaries.

In dictionaries, you can and usually have to edit the name of the key whereas in arrays the keys can not be edited and will always be displayed as "Item x" with x being a sequential number, starting with "Item 0".

Property List Editors

Until Xcode 4 the default Property List editor was the aptly named "Property List Editor.app" found in the `/Developer/Applications/Utilities/` folder. You can still use that, but the preferred editor is actually Xcode 4 itself. If you have Xcode 4 installed, double-clicking any .plist file should open Xcode and a window showing the contents of the Property List file. The Xcode 4 editor is slightly more comfortable to work with.

Wikipedia lists two noteworthy Property List editor alternatives. One is Pledit which enables Windows users to edit property list files. PlistEdit Pro on the other hand is for Mac users. I haven't tried either of the two alternatives.

⚠ The built-in Property List Editor of Xcode 4 as well as the separate Property List Editor app only display the first line of multi-line text. To find out whether a certain string actually consists of multiple lines, you can double-click the value to edit it and use the cursor keys to scroll up and down to change lines.

If you want to edit a multiline text you should temporarily select the entire text (Cmd+A) and copy the selected text (Cmd+C). This allows you to edit the entire text in a seperate text editor. To paste the text back into the value, double-click the value and select the entire text again (Cmd+A) to overwrite the entire existing text, and then paste your edited text (Cmd+V).

You could use a regular text editor to edit the plist but this is not advisable. You'll often write text containing characters that need to be escaped properly in order to retain the format of an XML file.

For example in XML you can't just write < or > in a regular string without breaking the entire XML format. Instead you would need to write

```
&lt;
```

and

```
&rt;
```

respectively. It's better to leave this error-prone job to the Property List editors or at the very least a generic XML editor.

Property List Programming Guide

If you are interested in a programmatic interface to Property Lists, for example if you want to create your own Property List editor or maybe a tool that makes creating Xcode Templates easier, you should have a look at Apple's Property List Programming Guide.

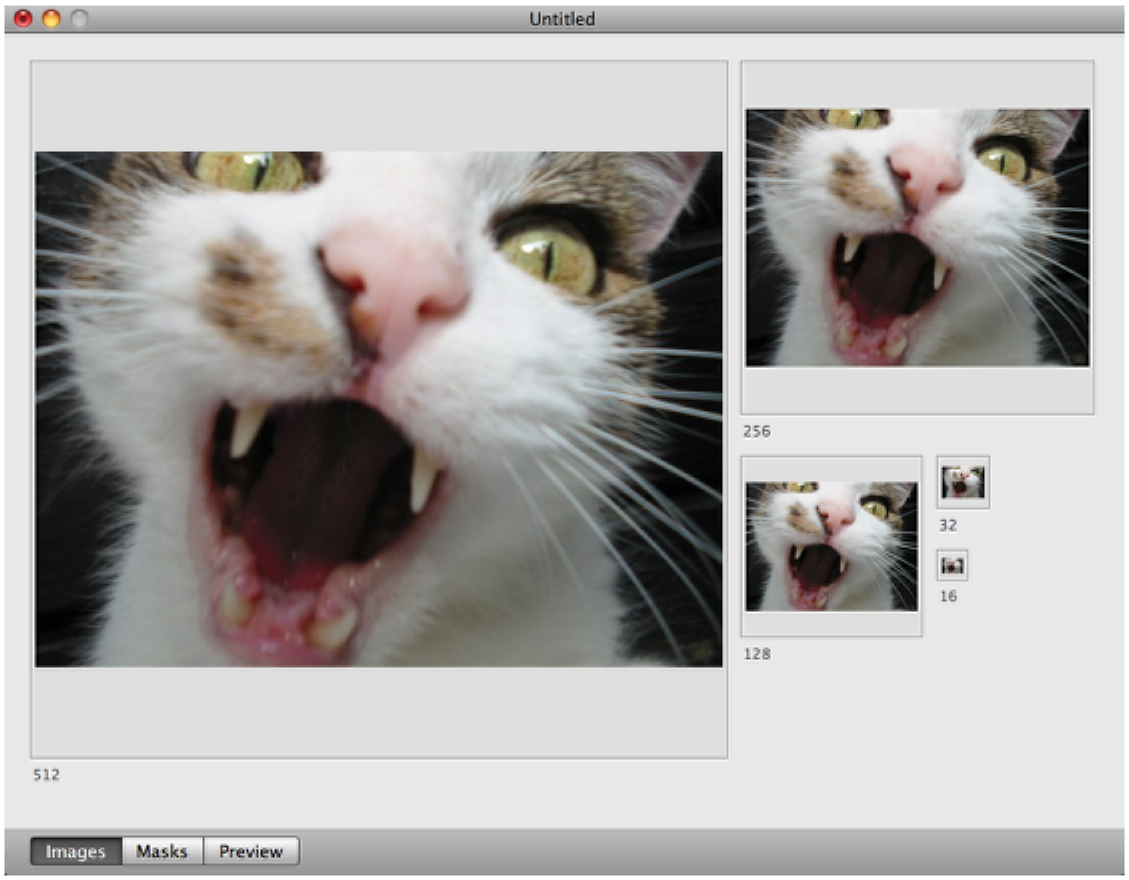
Icon Composer

Each File and Project Template is usually accompanied with a `TemplateIcon.icns` file. This file contains the icons for the template.

Icon Composer

You can create .icns files using Apple's Icon Composer.app which you can find in the `/Developer/Applications/Utilities/` folder of your Xcode installation. You can drag an image file onto Icon Composer to one of the predefined image size boxes (512, 256, 128, 32, 16). Once you have an image in Icon Composer, you can drag the image from within Icon Composer to another predefined image box. It is advisable to start with the largest image (512) and drag the image to subsequently smaller image boxes.

You can not drag .JPG and .GIF files onto Icon Composer. It does accept .PNG files, and it may accept other file formats but I haven't tried others.



TemplateIcon.icns

When you are satisfied with your new icon file, you'll have to save it into your template folder and name it `TemplateIcon.icns`. Xcode will then use it as the icon of your Template.

Frequently Asked Questions (FAQ)

- Can I create new project templates from scratch?
- Xcode reports an error or crashes when creating a new Template. What's wrong?
- Xcode crashes when trying to open the New Project dialog. How do I fix this?
- Do I have to restart Xcode for template changes to show up?
- Do template folders have to have the .xctemplate extension?
- What are the TemplateChooser.plist files used for?
- I have lots of files in my project template, can I copy them all at once?
- Can I place files in a project template into a subgroup hierarchy in Xcode?
- Can I use subfolders in the File Templates or Project Templates folder?
- In the New File/Project dialog (left side) is it possible to have subfolders?
- My folder does not show up as category, what's wrong?
- My template does not show up, what's wrong?
- Can I modify the New File / New Project dialog?
- Can I add a .xcodeproj as referenced project to my project template?

Can I create new project templates from scratch?

Yes.

However there are so many variables and required template settings to configure that it is hardly the best way to create your own custom templates entirely from scratch. Instead, it is highly recommended to use the Apple project template that best fits your project's needs, then copy it to your custom folder location and start tweaking it beginning with renaming the project templates Identifier. Please refer to the Basic Project Template Tutorials which explain how to make a copy of an existing Project Template.

At the very least you should rely on the commonly used Ancestors which contain settings for Objective-C projects and the desired target platform (Mac OS X respectively iPhone, iPad, Universal).

Xcode reports an error or crashes when creating a new Template. What's wrong?

There are various combinations of settings that can not be used together, resulting in errors, crashes and in same cases even data loss (please see Options#ProductNameRequired). I encountered several issues. There's too many to document all of these combinations, and it's unlikely that I can help you with a particular crash, but I can give you some advice on how to find and fix these issues:

- Retrace the changes you've made since the last time the template worked, and undo these changes.
- Check the documentation - are you using parameters together which are or seem unrelated? If you are using parameters that are only supported by File Templates in a Project Template or vice versa there should be alarm bells ringing.
- If the "logic" approach doesn't help, (temporarily) slim down your template step by step to narrow down the root issue. Test after each change. If the crash disappears, it must be caused by whatever you have removed last. Try re-adding that part step-by-step and by doing so you'll be able to pin down the issue.
- As a general rule of thumb: test early and often! You should test your templates frequently, and definitely before and after each major change or addition. This includes building the code because some issues (eg missing files) won't crop up until you actually build the project.

Xcode crashes when trying to open the New Project dialog. How do I fix this?

Malformed TemplateInfo.plist can cause Xcode to crash when it is supposed to bring up the New Project dialog. One common cause is to not have a unique Identifier key in the TemplateInfo.plist and all other templates it includes via Ancestors. You might also have created a circular reference via Ancestors with two Templates including each other.

One way to fix this is to remove all custom templates from the Template Folder Locations and adding them back in one by one to find out which one is causing the crash. Always check all the Ancestors beginning with the Concrete Template to verify that the inherited templates are also well-formed and valid.

Do I have to restart Xcode for template changes to show up?

No. But you do have to close the New File or New Project dialog, then re-open it, for Xcode to recognize your changes.

Xcode 4 will scan the user template folder whenever the New File, New Project or New Target dialog is opened. You might want to memorize the keyboard shortcuts CMD+N and SHIFT+CMD+N for opening the New File and New Project dialog respectively.

Do template folders have to have the .xctemplate extension?

Yes.

Xcode 4 will only recognize templates in folders using the .xctemplate extension. Xcode 3 templates did not have this requirement, which makes them easy to tell apart from Xcode 4 templates.

> ⛔ Merely appending the .xctemplate extension to Xcode 3 template folders will not make them work in Xcode 4.

What are the TemplateChooser.plist files used for?

They're only used by Xcode 3 templates. Xcode 4 doesn't use TemplateChooser.plist files anymore. If you see that file in a template's folder it's an indicator that this is actually a Xcode 3 template which won't work with Xcode 4.

I have lots of files in my project template, can I copy them all at once?

Yes and no.

You can point to a folder instead of each individual file with Nodes and Definitions. Xcode will then copy the contents of the entire folder (including subfolders) and add all files in that folder to the newly created project. However, these files will only be added to the Copy Bundle Resources build phase, in other words they won't be compiled so this is generally only useful for resource files.

Please refer to Definitions#CopyFolder for more information.

Can I place files in a project template into a subgroup hierarchy in Xcode?

Yes. You can create a hierarchy of groups using project templates.

Please refer to Definitions#Subgroups for an example.

Can I use subfolders in the File Templates or Project Templates folder?

Yes.

But only the folder containing a folder with the .xctemplate extension will show up in the category list. Meaning, you can't have a treeview of categories in the New File / New Project dialog.

In the New File/Project dialog (left side) is it possible to have subfolders?

No. See above.

My folder does not show up as category, what's wrong?

One common reason is that it does not contain a subfolder with the .xctemplate extension, or none of the .xctemplate extension folders contain a (valid) TemplateInfo.plist file. For example, the TemplateInfo.plist and all ancestors must have a correctly configured Kind key.

> ⚠ If your file system is formatted to be case sensitive, you may have to consider the possibility that Xcode may only look for TemplateInfo.plist files but not templateinfo.plist.

My template does not show up, what's wrong?

Verify that you have stored the template files and folders in the correct location for User Templates.

Check that the template folder ends in `.xctemplate` because without this folder extension the templates won't be recognized by Xcode. Make sure all TemplateInfo.plist files that your project template includes (via Ancestors) are also in folders which end in `.xctemplate`.

Check that all the TemplateInfo.plist files that your template includes (via Ancestors) contains the Kind item. Verify that Kind exists and that it is of the correct type (check for spelling errors, too).

Check that the Identifier of the template and all templates it includes via Ancestors is unique. If any of them isn't unique the template might not show up.

Can I modify the New File / New Project dialog?

Only with what's possible using template Options. You can determine the various options available to the user and depending on these settings create different versions of your template. But you can't add your own code or user interface to the process.

Can I add a .xcodeproj as referenced project to my project template?

Not really.

Adding the referenced project itself is no problem. The problem is that the files used by that external project aren't copied automatically when the template is created. If you don't mind copying those yourself or having your user copy them manually, you'll be ok.

If you specify the other project's files in the Nodes / Definitions sections of the template, they will be added to the new project, but that is not what you want. The files are used by the other project, they should not be added to the newly created project. At best you can get these files copied this way but you'd have to remove them from the newly created project, because they're also used by the referenced project.

In general the Xcode 4 project template system was never meant to support referenced Xcode projects. I wasn't able to find a satisfying solution after many hours of trial and error.