HelloWorld : Création d'un Projet avec Project Builder

Version française

http://trad.applescript.free.fr

Préambule

Ce tutoriel n'est absolument pas une traduction officielle de la Société Apple.

Ce tutoriel est un essai de traduction française de "HelloWord: Creating a Project with Project Builder", documentation technique Apple.

Dans l'espoir que cette version française comblera l'attente de tous les utilisateurs francophones, je vous souhaite une bonne lecture.

Un utilisateur Mac francophone

Merci à Daniel et Laurent pour leur aide et leurs conseils indispensables.

HelloWorld : Création d'un projet avec Project Builder

Ce tutoriel montre comment créer un projet avec Project Builder. Dans ce tutoriel, vous allez créer un nouveau projet à partir d'un modèle, le construire et le lancer, puis vous lui ajouterez une fonction, enfin vous construirez la nouvelle application et la lancerez. Tout au long de ce tutoriel, vous apprendrez comment éditer les fichiers, fixer les erreurs et utiliser le nouveau Carbon Event Model.

Ce tutoriel n'a pas pour but de vous apprendre la programmation dans Mac OS X. Vous allez créer une application Carbon, mais vous n'avez pas besoin de connaître Carbon pour tirer bénéfice de ce que vous allez apprendre.

Ce tutoriel comporte les chapitres suivants :

Créer le projet

Pour lancer Project Builder, double-cliquez, dans le Finder, sur son icone. Vous trouverez l'application Project Builder dans le répertoire `/Developer/Applications/`.

Dans Project Builder, choisissez File > New Project. Project Builder affiche alors une boîte de dialogue comportant plusieurs modèles parmi lesquels il vous faut choisir. Sélectionnez "Carbon Application (Nib Based)" et cliquez sur Next. Saisissez alors le nom du projet "Hello World", puis choisissez un emplacement pour l'enregistrement et cliquez sur Finish.

Project Builder crée un répertoire pour votre projet, il y place un fichier projet et d'autres fichiers source, puis il ouvre la fenêtre du projet. Le projet contient déjà des exemples de fichiers source que vous pouvez compiler et exécuter en l'état.

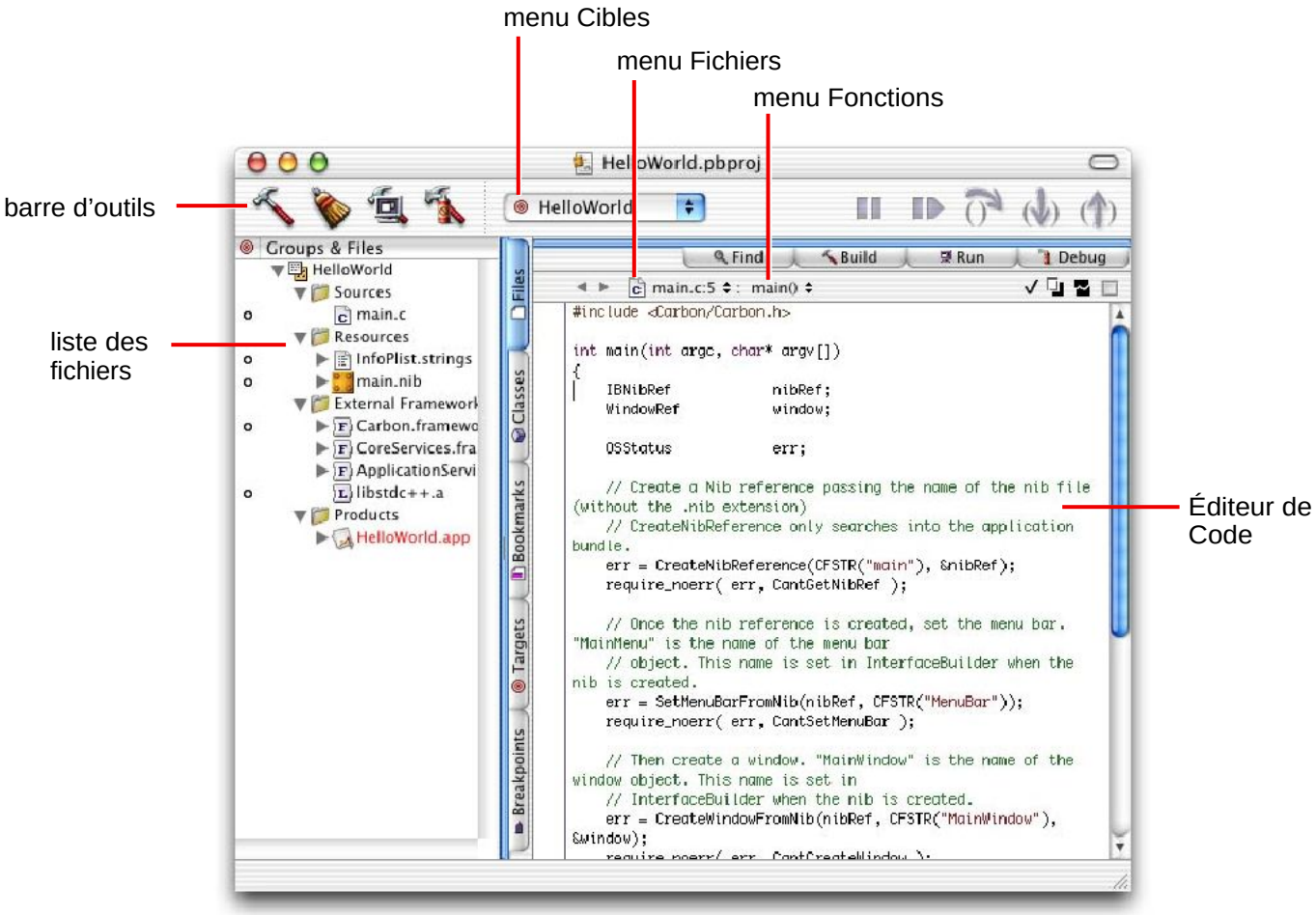
Prenez le temps pour regarder dans Project Builder le contenu de ce nouveau projet. Si vous connaissez déjà Projects (projets), Targets (cibles) et Frameworks (structures), vous pouvez passer directement à "Construire et lancer l'application modèle" (page 8).

Un projet Project Builder contient deux types d'éléments primaires : file references (fichiers référence) et targets (cibles). Les fichiers sont situés dans la liste à gauche de la fenêtre du projet, les cibles sont listées dans le menu déroulant de la barre d'outils.

- Les fichiers peuvent être des références de fichiers sources, de fichiers ressources, de librairies et de frameworks. Les fichiers, en tant que tels, ne seront pas dans votre projet. Vous pouvez regrouper ensemble des fichiers apparentés. Si un cercle est à côté d'un fichier, il est utilisé par la cible sélectionnée dans le menu cible.

- Les cibles spécifient comment construire des choses depuis les fichiers de votre projet. Les build styles permettent de modifier dynamiquement certains réglages dans la construction d'une cible. Un simple projet, comme celui-ci, a juste une cible qui construit une application. Un projet complexe peut en contenir plusieurs. Par exemple, un projet pour un logiciel client-serveur pourrait contenir des cibles pour une application-client, une application-serveur, un framework privé que les applications utilisent ensemble, et des outils de commandes en ligne que vous pouvez utiliser au lieu des applications. Le tutoriel "AboutBox.pdf" montre comment gérer les cibles.

Maintenant, regardons la fenêtre de votre projet.



En voici les éléments les plus importants :

- La fenêtre “Groups & Files” affiche tous les fichiers de vos projets. Pour en éditer un, cliquez dessus. Notez les onglets situés sur le côté droit de cette liste, ils vous permettent de visualiser, sous forme de listes, les classes, les signets (lesquels fonctionnent comme les signets d'un navigateur internet), les cibles et les points d'arrêt de vos projets.

- La barre d'outils permet d'accéder rapidement aux commandes les plus communes. Ces commandes agissent sur la cible et le style de construction spécifiés dans les menus proches. Dans ce tutoriel, vous n'utiliserez que les boutons “Build” (le marteau) et “Build et Run” (l'écran et le marteau).

- Le menu Cibles permet de contrôler ce que votre projet construit. Vous l'utiliserez dans le tutoriel “AboutBox.pdf”.

- Les onglets Actions, situés juste sous la barre d'outils, permettent d'exécuter différentes opérations et de visualiser leurs résultats, y compris la recherche (Find), la construction

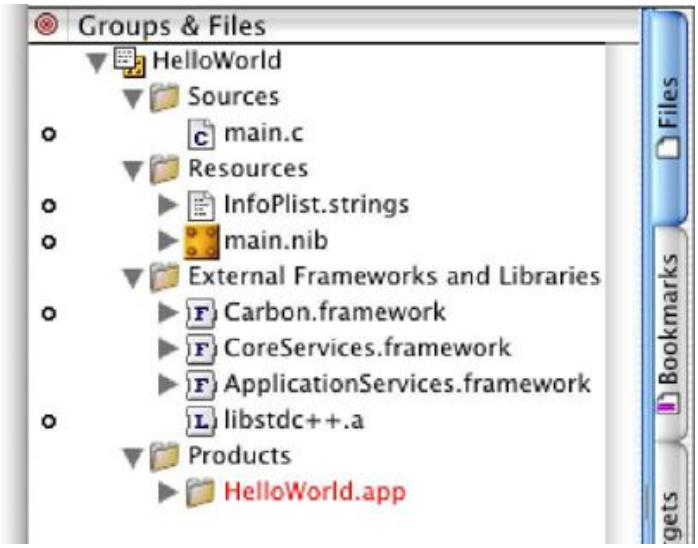
(Build), le lancement (Run) et le débogage (Debug). Dans ce tutoriel, vous n'utiliserez que les onglets Build et Run.

- Le menu Fichiers affiche la liste de tous les éléments que vous avez ouverts dans le projet depuis le démarrage de Project Builder.

- Le menu Fonctions montre la liste de toutes les fonctions, méthodes et classes définies ou déclarées dans le fichier.

- L'Éditeur de Code est un éditeur de texte complet et spécialisé permettant d'éditer le code source.

Dans la liste des fichiers de votre projet, les fichiers sont rangés en quatre groupes : Sources, Ressources, Frameworks et Librairies Externes, et Productions. Ce sont aussi les quatre principaux types de fichiers que Project Builder peut contenir. Pour voir ce qu'ils contiennent, cliquez sur les triangles à côté de chacun.



Voici ce que contiennent ces groupes :

- Sources : ce sont des fichiers qui sont compilés pour produire du code objet. Dans Project Builder, les fichiers en-têtes (header) sont listés dans la fenêtre du projet et sont généralement dans ce groupe. Ce projet contient seulement un fichier source, `main.c`.

- Ressources : ce sont des fichiers qui contiennent des ressources ou qui peuvent être compilés pour produire des ressources. Ce projet en contient deux : `main.nib`, qui contient les ressources de l'application, et `InfoPlist.strings`, qui contient certaines chaînes de caractères qui peuvent être localisées (en français par exemple). Pour en savoir plus sur les

fichiers Nib, voir le tutoriel “Converter_Fr.pdf”. Pour en savoir plus sur `InfoPlist.strings`, voir le tutoriel “AboutBox.pdf”.

- Frameworks et Librairies Externes : ce sont différents types de librairies. Dans Mac OS X, un framework est une librairie partagée qui regroupe fichiers en-têtes et ressources. Si vous cliquez sur le triangle à côté d’un framework, vous verrez la liste de ses fichiers en-têtes. Ce projet contient la librairie `libstdc++.a` et les frameworks `Carbon.framework`, `CoreServices.framework` et `ApplicationServices.framework`.

- Productions : ce sont les éléments que les cibles de votre projet peuvent produire. Il contient un fichier : `HelloWorld.app`. Son nom est en rouge car il n’existe pas de `HelloWorld.app` sur le disque. Mais lorsque vous l’aurez construit, cette référence pointera dessus.

Vous pouvez déplacer les fichiers dans les groupes comme vous voulez. Les groupes existent uniquement pour votre convenance et n’affectent pas les capacités de Project Builder pour trouver ou compiler les fichiers.

Maintenant, retournons au Finder et regardons de nouveau les fichiers présents dans le dossier HelloWorld de votre projet.

Le dossier `build` contient les fichiers créés lors de la construction de la cible de votre projet. En ce moment, il contient juste le dossier `intermediates`, lequel contient des informations sur votre projet. Après que vous aurez construit votre cible, le dossier `build` contiendra l’application construite et `intermediates` contiendra les fichiers créés lors de la construction de cette application, comme les fichiers objets.

`English.lproj` contient des ressources qui sont disponibles uniquement en langue anglaise. Ici, il contient `InfoPlist.strings`. Vous pouvez aussi avoir des dossiers `lproj` pour d’autres

langues, comme `French.lproj` ou `Japanese.lproj`.

`HelloWorld.pbproj` est le document qui garde la trace des fichiers et des cibles de votre projet. Vous pouvez double-cliquer dessus pour ouvrir votre projet.

Vous noterez que bien que `main.nib` soit dans un groupe différent de celui de `main.c`, ils sont tous les deux dans le même dossier. Notez aussi que `Carbon.framework` n'est pas dans celui-ci. Il est dans le répertoire `/System/Library/Frameworks`. Le projet contient uniquement une référence à `Carbon.framework`.

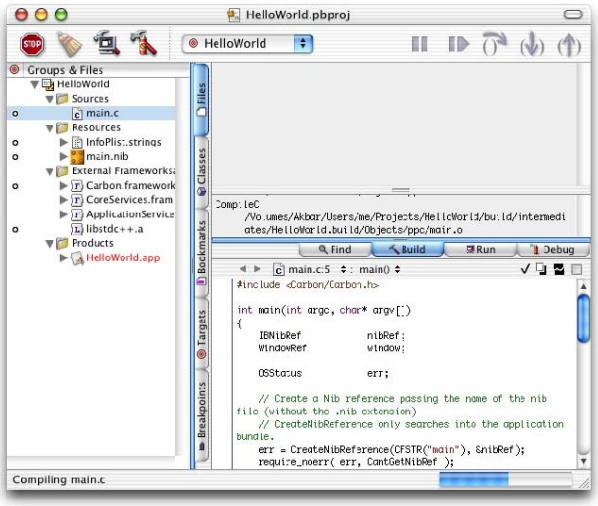
Construire et lancer l'application modèle

En premier, vous construirez et lancerez l'application modèle sans la modifier, juste pour obtenir un gabarit pour Project Builder.

1. Construction de l'application.

   Cliquez sur le bouton Build. Il est représenté par un marteau dans la barre d'outil.

   Le panneau Build s'ouvre de bas en haut à partir du haut de la fenêtre de votre projet. Dans ce panneau, la partie haute affiche les messages d'erreur, s'il y a, et la partie basse affiche le journal de la construction. Comme vous n'avez pas modifié les fichiers sources, il ne devrait pas y avoir de messages d'erreur.

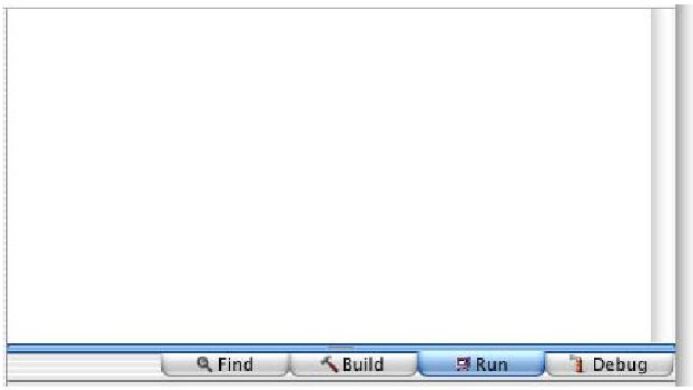


Lorsque la construction est finie, “Build succeeded” apparaît au bas de la fenêtre du projet. Pour fermer ce panneau, il suffit de cliquer sur l’onglet Build.

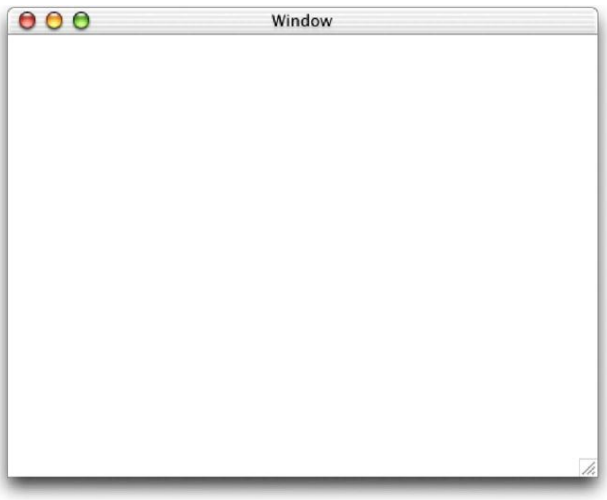
2. Lancement de l’application.

Cliquez sur le bouton “Build et Run”. Il est représenté par un marteau et un écran d’ordinateur dans la barre d’outils.

Project Builder lance l’application et le panneau Run s’ouvre, il affiche les messages écrits à `stdout` et `stderr`. Parfois, il peut être pratique d’effectuer un débogage en écrivant à ces flux. Project Builder permet aussi d’exécuter un débogage plus complexe, comme le montre le tutoriel “Debugger_Fr.pdf”.



L’application affiche une fenêtre vide. Pour quitter, appuyer sur Cmd + Q.



Examen de l’application modèle

Maintenant, nous allons examiner le code de l’application modèle et apprendre comment obtenir la documentation sur les fonctions appelées.

1. Regardons le code existant.

   Pour regarder le code, cliquez sur `main.c` dans la liste des fichiers de la fenêtre du projet.

   Vous n’avez besoin que de six fonctions pour écrire une application qui utilise le Carbon Event Manager. Voici la liste, établie à partir du fichier `main.c`, qui se trouve dans votre projet, avec la vérification d’erreur et les commentaires :

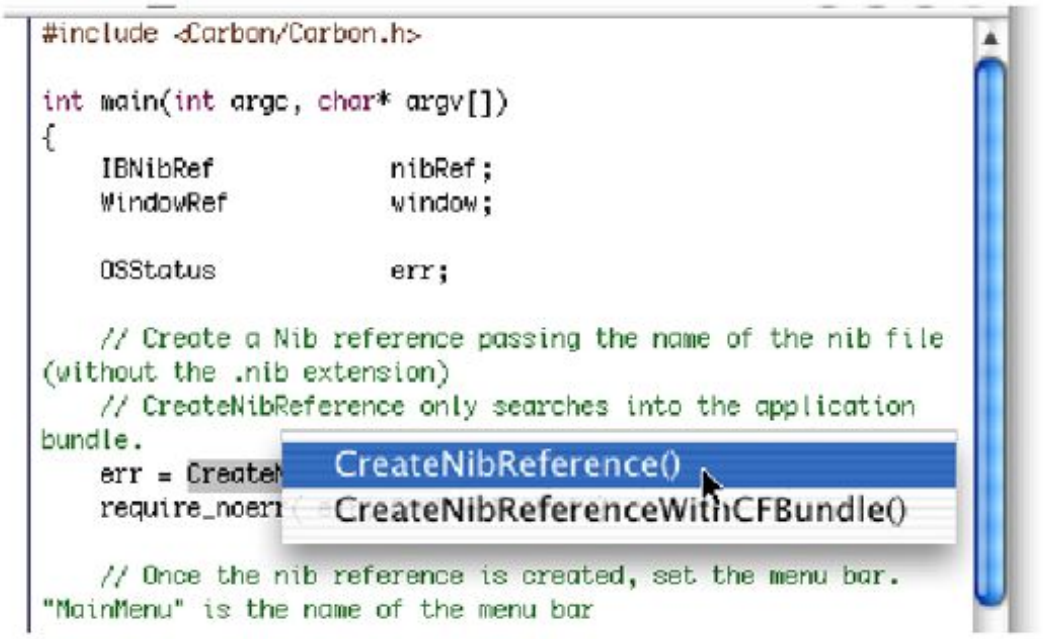
```
CreateNibReference( CFSTR("main"), &nibRef );
err = SetMenuBarFromNib(nibRef, CFSTR("MenuBar"));
CreateWindowFromNib( nibRef, CFSTR("MainWindow"), &window );
DisposeNibReference( nibRef );
ShowWindow( window );
RunApplicationEventLoop();
```

   Notez que vous n’avez pas besoin d’initialiser les toolboxes, ni d’écrire votre propre boucle d’Events. Tout cela est géré automatiquement.

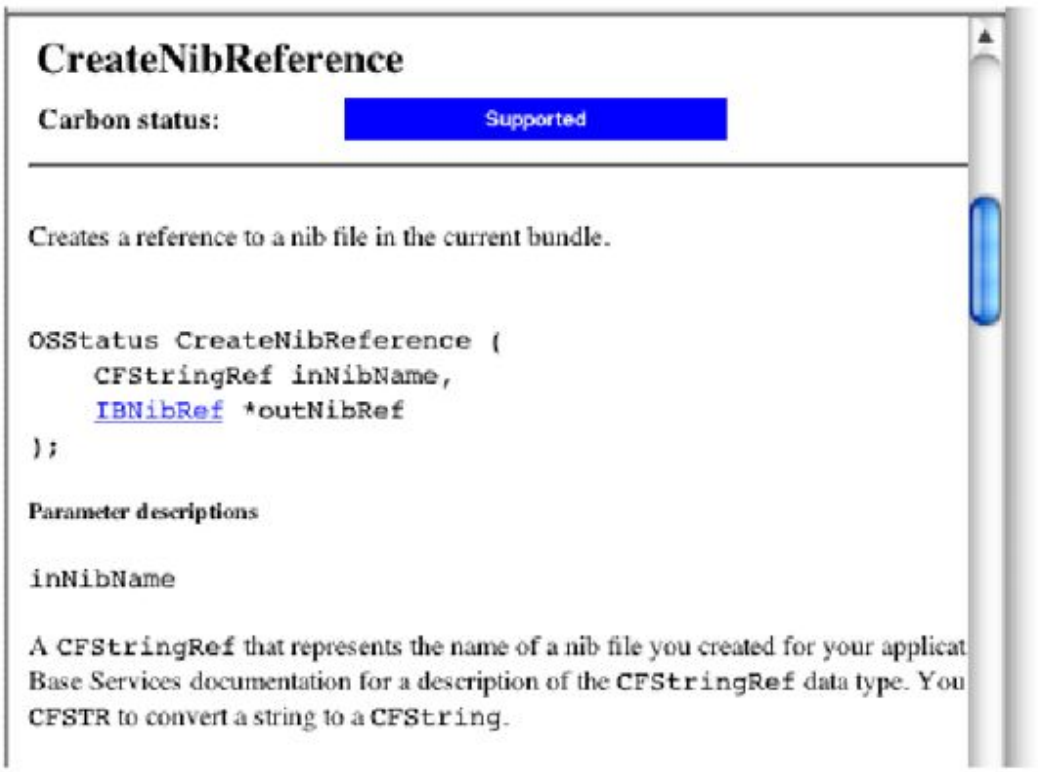
   Pour plus d’informations sur l’écriture de programmes qui utilisent le Carbon Event Manager et les fichiers Nib, voir le tutoriel “Converter_fr.pdf”.

2. Demandons une documentation.

   Si vous voulez visualiser la documentation d’une fonction dans la librairie Carbon, faites juste Option + double-clic dessus. Par exemple, faites Option + double-clic sur `CreateNibReference`. Un menu apparaît avec deux fonctions dont les noms commencent par `CreateNibReference`.



Sélectionnez celle du haut. La documentation sur `CreateNibReference` apparaît dans la fenêtre. Pour visualiser cette documentation, vous pouvez avoir besoin de faire défiler la fenêtre.



Vous pouvez copier et coller n'importe quel texte dans la documentation, comme une déclaration de fonction. Vous pouvez aussi cliquer sur un lien pour obtenir plus d'informations. Si le lien renvoie sur une page internet, elle s'ouvrira dans le navigateur défini par défaut.

Notez que vous devez construire votre projet avant de pouvoir utiliser Option + double-clic ou n'importe quelle autre caractéristique décrite dans cette section. Car ces caractéristiques se servent de l'index créé lors de la construction du projet.

Vous pouvez utiliser Option + double-clic pour obtenir la documentation pour la plupart des symboles de Carbon et Cocoa.

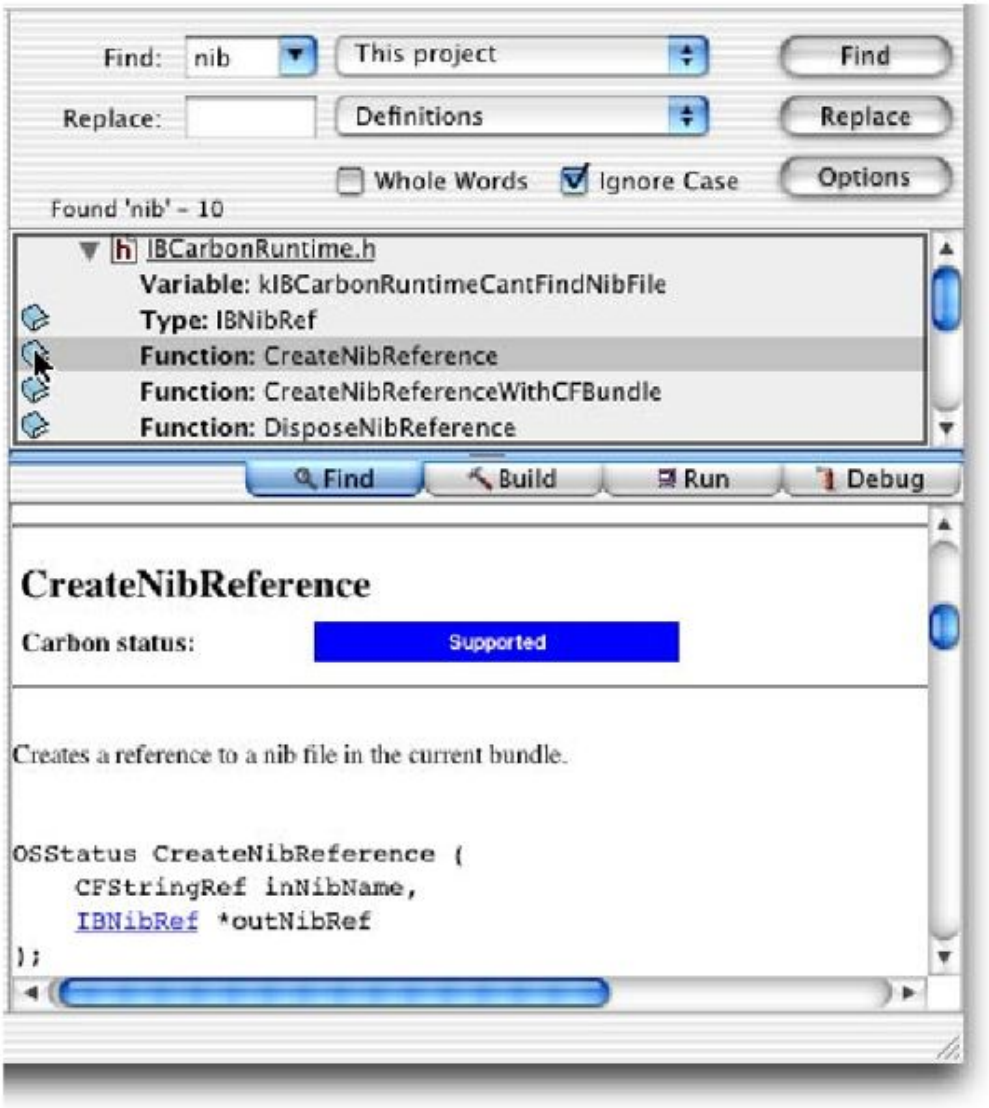
3. Demandons une déclaration

S'il n'y a pas de documentation pour un symbole, faites alors Cmd + double-clic dessus pour voir sa déclaration. Mais d'abord, retournez à `main.c` en cliquant sur `main.c`, dans la liste des fichiers. Puis, faites Cmd + double-clic sur `require_noerr` et choisissez `#define require_noerr` dans le menu qui apparaît. Sa déclaration apparaît dans la fenêtre.

Vous pouvez utiliser Cmd + double-clic pour obtenir la déclaration de n'importe quel symbole de votre projet.

4. Recherchons un symbole incomplet

Si vous ne vous rappelez que d'une partie du nom d'un symbole et non son nom exact, utilisez le panneau Find (recherche). Par exemple, cliquez sur l'onglet Find, saisissez "nib" dans le champ pour la recherche, choisissez Definitions dans le deuxième menu déroulant, et cliquez sur le bouton Find. Project Builder trouvera tous les symboles de votre projet qui contiennent le mot "nib". Si le mot trouvé comporte à côté de lui un icone représentant un livre, vous pouvez cliquer sur ce livre pour visualiser sa documentation.



Vous pouvez aussi utiliser le panneau Find pour effectuer une recherche, dans tous les fichiers de votre projet, pour du texte ou une expression.

Écriture de la nouvelle fonction

Maintenant vous allez écrire la fonction qui affiche “Hello World !” dans cette fenêtre vide.

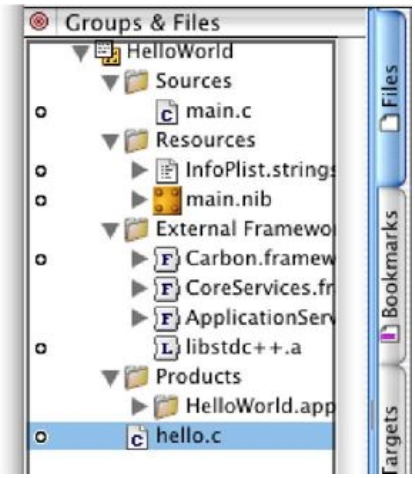
1. “Écrire le fichier source” (page 14)
2. “Écrire le fichier En-tête” (page 17)

Écrire le fichier source

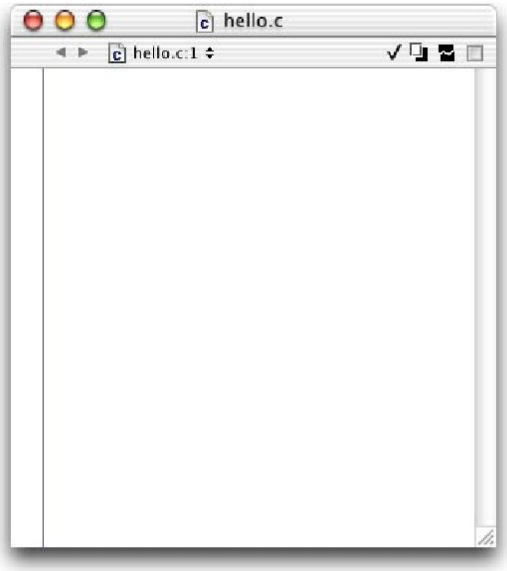
1. Créez le fichier source.

   Choisissez File > New File. Sélectionnez Empty File et cliquez sur Next. Saisissez `hello.c` comme nom pour le fichier et cliquez sur Finish.

   Project Builder va alors créer un nouveau fichier et mettre une référence de ce fichier dans votre projet.



Le fichier est automatiquement ouvert dans la fenêtre projet de l'éditeur de code. Si vous le souhaitez, vous pouvez l'ouvrir dans une fenêtre séparée en double-cliquant dessus dans la liste des fichiers.



2. Saisissez le code source.

Vous pouvez, soit saisir le code manuellement, soit le copier-coller depuis ce document.

Le fichier hello.c

```
#include <Carbon/Carbon.h>

pascal OSStatus PrintHello( EventHandlerCallRef handlerRef,
     EventRef event, void *userData )
{
     short int fNum    //<<-- Erreur! oubli du point-virgule

     SetPort( GetWindowPort( (WindowRef) userData) );
     GetFNum( "\pPalatino", & fNum );
     TextFont( fNum);
     TextFace( bold + italic );
     TextSize( 48 );
     MoveTo( 5, 50 );
     DrawString( "\pHello World !" );
     return noErr;
}
```

Plus tard, vous installerez cette fonction comme un gestionnaire d'Event que le Carbon Event Manager appellera chaque fois que la fenêtre de cette application sera mise à jour. Quelques précisions sur cette fonction :

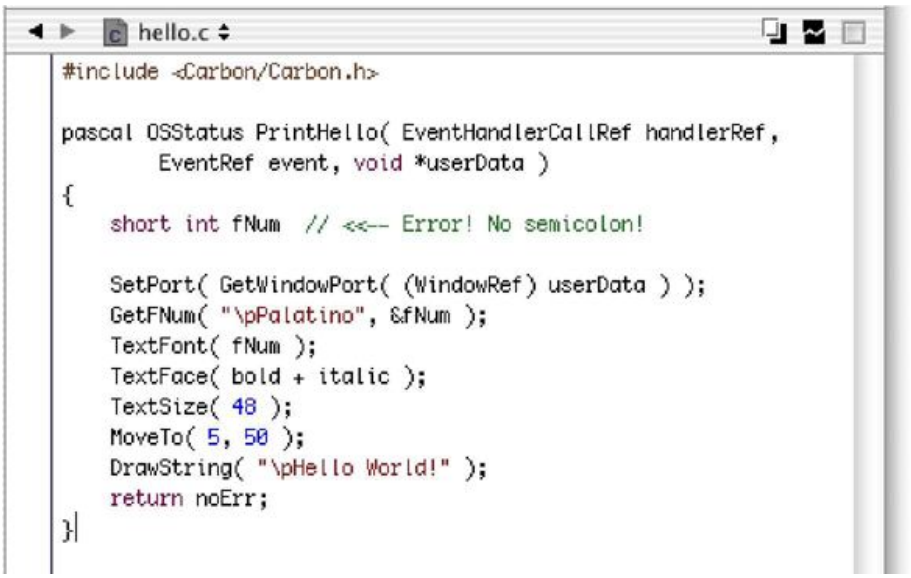
- Elle n'utilise pas les deux premiers paramètres, `handlerRef` et `event`. C'est à dire, comme le Carbon Event Manager délègue les types et les paramètres de retour aux gestionnaires d'Event, cette fonction doit déclarer ces paramètres bien qu'elle ne les utilise pas.

- Le dernier paramètre, `userData`, permet de transmettre n'importe quelles données voulues à cette fonction. Lorsque vous installerez cette fonction, vous spécifierez que `userData` devra pointer sur la fenêtre de l'application, ainsi cette fonction pourra modifier la fenêtre.

- Elle contient également une erreur de syntaxe que vous régulariserez par la suite.

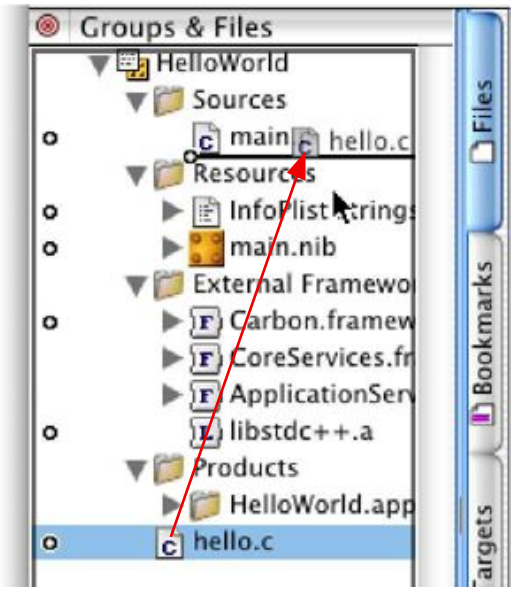
N'oubliez pas que si vous voulez plus d'informations sur une de ces fonctions, vous pouvez faire Option + double-clic sur la fonction voulue.

Le code apparaît dans la fenêtre de l'éditeur comme dans l'illustration suivante.



Notez que Project Builder utilise une syntaxe en couleur, cette syntaxe colorée permet une lecture plus aisée du code. Par défaut, les commentaires sont en vert, les mots-clés (comme `short int`) sont en violet, les instructions du pré-processeur (comme `#include`) sont en marron, les chaînes de caractères (comme `"\pHello World"`) sont en rouge et les nombres littéraux (comme `40`) sont en bleu. La syntaxe colorée peut vous aider à trouver les erreurs de syntaxe. Par exemple, si un grand bloc de code est en rouge, vous pourrez présumer que vous avez certainement oublié un guillemet quelque part. Les couleurs peuvent être modifiées dans les préférences de Project Builder (choisissez Project Builder > Preferences, et cliquez sur Syntax Coloring).

Si vous voulez , vous pouvez déplacer le fichier et le mettre dans le groupe Sources. Pour se faire, il vous suffit de faire glisser l'icone du fichier avec la souris.



Écrire le fichier En-tête

1. Créez l'En-tête.

   Choisissez File > New File. Sélectionnez Empty File et cliquez sur Next. Saisissez `hello.h` pour le nom du fichier, puis cliquez sur Finish.

2. Saisissez le code de l'En-tête.

   Vous pouvez, soit le saisir manuellement, soit le copier - coller depuis ce document.

   Le fichier hello.h

```
pascal OSStatus PrintHello( EventHandlerCallRef handlerRef,
     EventRef event, void *userData );
```

   Si vous le souhaitez, vous pouvez déplacer le fichier et le mettre dans le groupe Sources comme précédemment.

Installer la nouvelle fonction en tant que gestionnaire d'Event

Dans cette section, vous allez installer PrintHello en tant que gestionnaire d'Event, afin qu'elle soit appelée chaque fois que la fenêtre de l'application est mise à jour.

1. Incorporez le nouveau fichier En-tête.

   Déplacez le point d'insertion au début du fichier main.c. Après `#include <Carbon/Carbon.h>`, rajoutez cette ligne :

```
#include "hello.h"
```

2. Déclarez l'indicateur de type d'Event.

   Lorsque vous installez un gestionnaire d'Event, l'indicateur de type d'Event indique au Carbon Event Manager quand il doit appeler le gestionnaire. Ici, le gestionnaire doit être appelé chaque fois que la fenêtre est mise à jour.

   Après la déclaration `OSStatus err`, rajoutez cette ligne :

```
EventTypeSpec eventSpec =
     { kEventClassWindow, kEventWindowDrawContent };
```

3. Installez le gestionnaire d'Event.

   Après la ligne `DisposeNibReference(nibRef)`, rajoutez cette ligne :

```
InstallWindowEventHandler( window,
    NewEventHandlerUPP(PrintHello), 1, &eventSpec,
    (void *) window, NULL );
```

   `InstallWindowEventHandler` dit au Carbon Event Manager d'appeler `PrintHello` chaque fois que `eventSpec` se produit dans `window`. Ici, cela signifie que `PrintHello` est appelé chaque fois que `window` est modifié. Notez que l'avant-dernier paramètre est un pointeur sur la fenêtre de l'application. C'est de cette manière que vous indiquez au Carbon Event Manager qu'il devra transmettre ce pointeur à `PrintHello` dans l'argument `userData`.

   Si vous faites Option + double-clic sur `InstallWindowEventHandler`, vous noterez qu'il n'y a pas de documentation disponible. Dans ce cas, la chose à faire est de faire Cmd + double-clic pour regarder la déclaration. Alors, vous constaterez qu'il s'agit d'une macro qui appelle `InstallEventHandler`. En faisant Option + double-clic sur ce nom, vous obtiendrez la documentation.

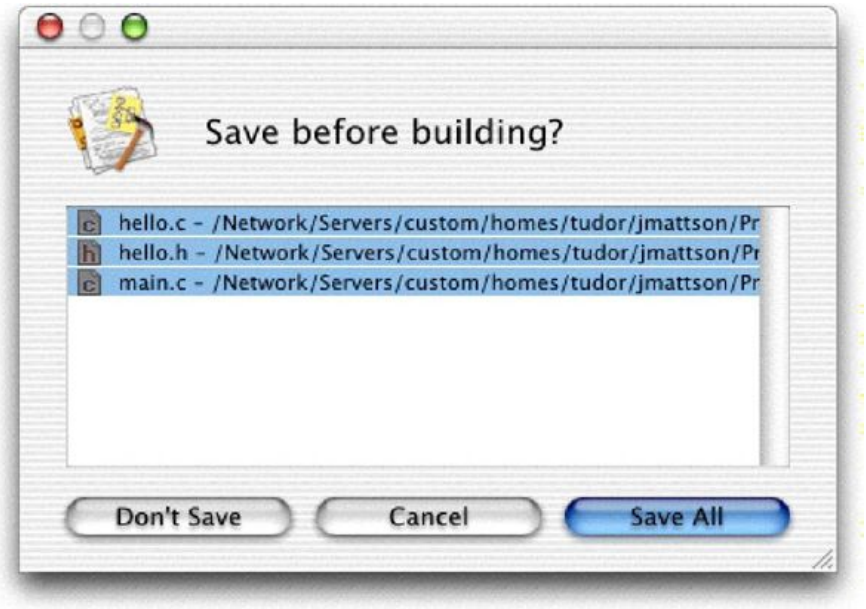
Fixer, construire et lancer la nouvelle application

Pour finir, vous allez devoir fixer l'erreur du fichier `hello.c`, puis vous construirez votre application et la lancerez.

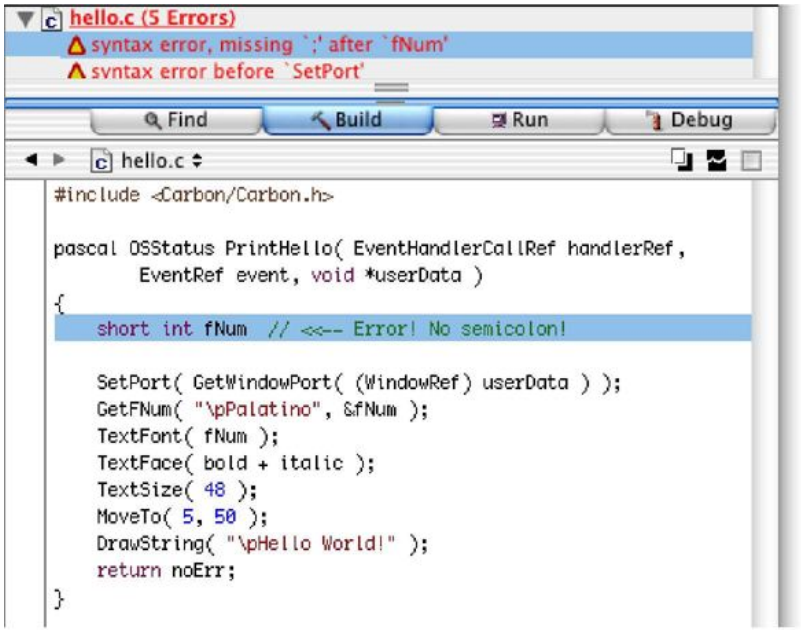
1. Construisez l'application

   Cliquez sur le bouton Build (le marteau).

   Si vous n'avez enregistré aucun fichier pour l'instant, Project Builder affichera un dialogue demandant s'il doit le faire. Cliquez sur Save All.



Le panneau Build s'abaisse et Project Builder commence la construction de votre projet. Un message d'erreur apparaît dans la liste des résultats, vous signalant l'oubli d'un point-virgule.



Chaque fichier contenant des erreurs est listé dans la liste des résultats, avec ses erreurs listées dessous.

2. Fixez l'erreur.

Cliquez sur le message d'erreur. La ligne comportant l'erreur est sélectionnée dans l'éditeur de

code. Rajoutez un point-virgule à la fin de la ligne. Le commentaire ne servant plus, vous pouvez en profiter pour le supprimer.

3. Relancez la construction de l'application.

   Cliquez de nouveau sur le bouton Build. À présent, Project Builder construit le projet sans rencontrer aucun problème.

   Avant de lancer l'application, retournez dans le Finder et regardez les fichiers se trouvant dans le dossier de votre projet.

   Le dossier Build contient maintenant HelloWord, l'application construite. Le dossier intermediates contient maintenant les fichiers objets et d'autres fichiers créés par Project Builder lors de la construction de l'application.

4. Lancez l'application.

   Cliquez sur le bouton Run.

   Project Builder lance l'application, celle-ci affiche une fenêtre dans laquelle apparaît la phrase "Hello World !".

Voilà, vous avez fini !

À plus, pour de prochaines aventures ! comme dirait Daniel. :-))))