

# Présentation du projet GNUstep

Nicolas Roard et Fabien Vallon

24 janvier 2003

*«Mon avis sur Microsoft est qu'ils ont eu deux buts dans les dix dernières années : copier le Macintosh et le succès de Lotus dans la vente d'applications. Et ils y sont arrivés. Maintenant, ils sont un peu perdus. J'ai dit à Bill [Gates] que je pensais que c'était dans le meilleur intérêt de Microsoft si NeXT réussissais, car nous leur donnerons quelque chose à copier pour le reste de la décade»*

– Steve Jobs, dans les années 90

NeXT a échoué, mais certaines de ses avancées sont encore présentes – avec Mac OS X – mais aussi, ce qui nous intéresse plus, dans le projet GNUstep.

## Historique

### De NeXTSTEP à OPENSTEP

Avant de parler de GNUstep, un bref récapitulatif historique est nécessaire.

Tout a commencé en 1985, quand Steve Jobs, qui venait de quitter Apple, a décidé de fonder une autre société informatique : NeXT Inc. <sup>1</sup> Avec NeXT, son but était de réussir à construire une machine ayant 20 ans d'avance.

Entouré d'une équipe de brillants ingénieurs (Avie Tevanian, Jean-marie Hullot, etc.), NeXT sort, en 1988, une machine révolutionnaire : le NeXT Cube. Celui-ci est doté d'un système unix (personnalité BSD au-dessus d'un noyau Mach), d'un affichage PostScript haute résolution, le tout disposant d'un environnement graphique entièrement orienté objet (utilisant Objective-C, un sur-ensemble orienté objet du C ANSI, inspiré de SmallTalk).

Le monde entier s'émerveilla devant la machine... et quasiment personne n'en acheta : mauvais marketing, relations avec les développeurs difficiles, machines considérées comme trop chères... les incarnations suivantes de l'OS ou des machines n'y changeront rien. NeXT ne sera jamais qu'un fantasme de

<sup>1</sup>pour la petite histoire, l'autre société que Jobs a fondée à ce moment s'appelle Pixar, les créateurs (entre autre) de Toy Story...

développeur.

NeXT Computer deviendra finalement NeXT Software, la construction des machines abandonnée et la société se réorientant vers la vente de logiciels. En particulier leur système, NeXTSTEP, puis une évolution de celui-ci, OPENSTEP. NeXT sera finalement racheté par Apple en 1996, et ironie du sort, Steve Jobs (re)devient le CEO d'Apple<sup>2</sup>.

La particularité d'OPENSTEP était d'être une implémentation de spécifications ouvertes (OpenStep), faites en collaboration avec Sun, qui étaient une évolution du framework de NeXTSTEP. OPENSTEP existera sur machines SUN, HP, Intel, et même au-dessus de Windows NT.

De nos jours, MacOS X est ainsi une évolution d'OPENSTEP, avec un affichage PDF au lieu de PostScript, et basiquement, un mode de compatibilité avec les anciens logiciels Mac en plus des outils OPENSTEP.

## GNUstep

À la suite de la publication des API OpenStep, un projet de la Free Software Foundation a commencé à les réimplémenter : GNUstep. Celui-ci était destiné à devenir l'environnement par défaut du système GNU. Mais le projet a avancé très lentement au cours de ces années, car peu de développeurs travaillaient dessus. Contrairement à d'autres projets, il y avait un système entier énorme à réimplémenter à partir de rien (problème de l'affichage PostScript par exemple), il était alors difficile d'obtenir quelque chose d'intéressant en cours de cette implémentation... d'où un net problème de visibilité. Cependant, on constate que depuis quelques temps, le système est utilisable par les développeurs : l'implémentation est suffisamment complète pour permettre le développement de programmes utilisant l'environnement GNUstep. Et effectivement, depuis un an environ, les applications commencent doucement à arriver...

<sup>2</sup>les mauvaises langues disent que NeXT a réussi le tour de force de racheter Apple tout en se faisant payer :)

## Les avantages de GNUstep

GNUstep est donc une réimplémentation d'API connues et documentées : OpenStep.

Ainsi, ce qui a été un désavantage (peu intéressant de travailler dessus tant que l'ensemble n'a pas atteint un stade suffisant, d'où difficultés à intéresser les programmeurs) est aussi un avantage, car on ne verra pas de bouleversements profonds du framework comme ce qui a pu arriver au cours des diverses versions de KDE ou Gnome.

Le framework OpenStep lui-même est particulièrement bien fini et cohérent, et n'a pas nécessité de modifications particulières depuis sa publication... en 1994!

MacOS X ajoute cependant quelques nouveaux widgets à ceux disponibles (que GNUstep implémente au fur et à mesure), mais étant une implémentation d'OpenStep, la portabilité entre une application GNUstep et une application MacOS X est grande. Un programme GNUstep pourra être recompilé quasiment tel quel sous Mac <sup>3</sup>.

L'objectif final de GNUstep est d'apporter une vraie portabilité multiplateforme. Les applications peuvent par exemple être distribuées sous forme de dossier (wrapper) contenant plusieurs versions (une pour x86, une pour ppc, etc.). GNUstep supporte bien sûr la localisation, avec de plus l'avantage de pouvoir retravailler, sans toucher au code, l'interface graphique pour chaque langue si besoin est.

GNUstep ne s'occupe pas que de la gestion d'une GUI, mais propose également des outils pour la partie non graphique (threads, structures de données... ou de différentes fonctionnalités comme un clipboard puissant, une coopération facilitée et intelligente entre programmes (les services), etc).

On peut également utiliser des objets répartis (entre différents processus ou machines) de façon particulièrement simple (ils ne se différencient des objets normaux que par leur initialisation...), les objets persistants, etc.

GNUstep propose aussi un framework (GDL2/EOF) permettant de mapper des objets sur une base de données classique, offrant ainsi à la fois le confort de l'orienté objet pour gérer des données et une abstraction par rapport au SGBD utilisé.

Enfin, l'avantage le plus évident pour le développeur, est la disponibilité d'outils comme Gorm, un constructeur d'interfaces graphiques, qui accélèrent de beaucoup le développement d'un logiciel.

<sup>3</sup>l'exemple le plus probant est celui de GNUmail, <http://www.collaboration-world.com/gnumail>

## Vue d'ensemble de la plateforme GNUstep

La figure 1 nous montre l'architecture utilisée par GNUstep. Une application graphique s'appuie ainsi sur deux frameworks, Foundation et AppKit.

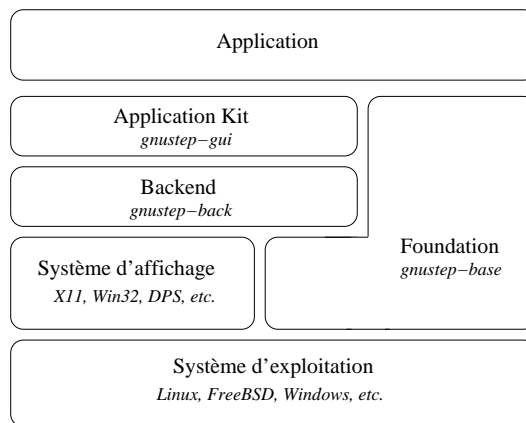


FIG. 1 – Composition en couches

### base (Foundation Kit)

Le Foundation Kit est un framework non graphique permettant de s'abstraire du système d'exploitation. Outre les classiques manipulations de chaînes, de tableaux, le Foundation Kit propose une gestion de ressources (Bundle), des threads, la persistance des objets, de l'Unicode... Le Foundation Kit est une brique essentielle garantissant la portabilité de la plateforme GNUstep.

### gui (Application Kit)

L'Application Kit contient les classes de la partie graphique (objets graphiques tels que les fenêtres, les boutons, les panels, la gestion d'événements, ...) mais une orientation Desktop aussi avec la gestion des périphériques, des services, des lancements d'application ou du Pasteboard.

### backend

GNUstep, à l'inverse de MacOS X, propose également une abstraction du système d'affichage. Il supporte actuellement X11 pour les Unix traditionnels, Windows, ou un affichage PostScript (le pendant de l'affichage PDF d'Apple), un backend DirectFB est également à l'étude...

Étant implémenté en tant que Bundle (comprenez, plug-in, pour l'instant) il est même possible de passer de l'un à l'autre, voir d'en avoir plusieurs en même temps...

## autres Frameworks : (EOF/GNUstep-Web/Pantomime...)

Au-delà de Foundation et d'AppKit, GNUstep propose de nombreuses extensions :

- GNUstep Web : le serveur d'applications compatible WebObject
- StepTalk : un framework de scripting permettant de manipuler les objets et les applications depuis divers langages
- AppTalk : un framework permettant de façon triviale d'ajouter le scripting à une application donnée
- des passerelles permettant d'utiliser à la fois des classes Objective-C, Java et Ruby
- 3DKit, un framework gérant la 3D et la composition de scènes

D'autres frameworks sont disponibles tels que Pantomime (une implémentation de JavaMail en Objective-C), MusicKit ou le SoundKit.

## Installation de GNUstep

Un peu d'exercice pour préparer le mois prochain : installation de la plateforme GNUstep. Les plateformes de référence seront ici GNUstep/Linux (Debian Woody) et GNUstep/BSD (FreeBSD-4.7). Vous pourrez facilement transposer sur votre OS préféré, cela ne posera normalement pas de problèmes sur celles de type \*nix; l'installation de GNUstep/MS-Windows ou GNUstep/BeOS étant un peu plus complexe et moins complète.

Tout d'abord le compilateur. Même si GNUstep fonctionne sous gcc-2.9x il est conseillé d'utiliser un compilateur gcc-3.x.

Sur Debian un simple :

```
apt-get install gcc-3.0 gobjc-3.0
```

Sur FreeBSD (gcc-33 est disponible via CVSup) un non moins simple :

```
cd /usr/port/lang/gcc33/ && make install clean
```

vous installera le compilateur (vous pouvez supprimer la partie Java si vous souhaitez gagner en temps de compilation :)

Les packages Debian autant que les ports BSD sont un peu vieux (même sur sid), il est donc conseillé de

prendre les packages à partir du CVS :

```
cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvs-root/gnustep login - Appuyez sur Entree -
```

Nous n'allons en prendre qu'une petite partie ...

```
cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvs-root/gnustep co core
```

Positionnons notre compilateur par défaut.

```
export CC="gcc-3.0 -fno-strict-aliasing"
```

Maintenant, allez dans core/make :

```
./configure - ./configure -disable-openssl -  
make - gmake pour FreeBSD -  
make install - gmake install pour FreeBSD -
```

Passez éventuellement en root pour le make install. Vous pouvez rajouter l'option `-prefix=votrechemin` pour choisir un répertoire d'installation différent du choix par défaut.

Si on fait un tour à l'endroit où vous avez installé GNUstep, vous y trouverez une arborescence que les utilisateurs de MacOS X ou d'OPENSTEP connaissent bien :) - dans un système «pur \*Step» cette arborescence se retrouve à la racine (et pour les utilisateurs de WindowMaker votre `~/GNUstep` serait réellement votre `homeDirectory`)

Bien. Il est maintenant nécessaire de positionner les variables d'environnement GNUstep. Pour cela je vous conseille de mettre "source /ouGNUstepEstInstallé/System/Makefiles/GNUstep.sh" (ou GNUstep.csh si vous utilisez un C-shell) dans vos profils de shell.

Retournons dans notre repository CVS et plus précisément dans core/base.

La base est ce que les \*Steppers appelle Foundation. Il est conseillé de posséder les bibliothèques de développement openssl, libxml2 et iconv à savoir libssl-dev, libxml2-dev pour la Debian iconv étant normalement dans le libc et iconv-2.0.3, libxml2-2.4.24, pour FreeBSD.

Si vos variables GNUstep sont bien positionnées, un simple

```
./configure  
make - gmake pour FreeBSD -  
make install - gmake install pour FreeBSD -
```

devrait suffire (si vous installez en root n'oubliez pas de sourcer le GNUstep.sh)

Pour AppKit, appelé ici «gui» (dans core/gui), il est nécessaire d'avoir les bibliothèques de développement tiff. L'installation se déroule de la même façon que pour -base.

Terminons par le backend.

Sous X11 il existe même plusieurs backends (utilisant soit la xlib, soit libart). Nous choisirons libart par défaut qui, même si elle est plus ardue à installer, possède un meilleur rendu des fontes et un meilleur moteur graphique (antialiasing des tracés vectoriels, meilleur support d'imagerie (alpha channel), par exemple). Vous devez donc posséder libart-2 et freetype2, il est également conseillé de posséder les libwrafter2-dev.

Allons donc dans core/back :

```
./configure --enable-graphics=art
make - gmake pour FreeBSD -
make install - gmake pour FreeBSD -
```

Vous devez maintenant mettre les Fontes dans /GNUstep/Library/Fonts/ Un jeu minimal de fontes peut être récupéré sur <http://developer.linuxstep.org/downloads/>, on peut aussi utiliser des fontes NeXT : <ftp://ftp.peak.org/next-ftp/next/fonts/index.html>

Nous sommes prêts. Positionnons notre propre environnement en commençant par le fuseau horaire :

```
defaults write NSGlobalDomain "Local Time Zone"
Europe/Paris
```

puis le langage :

```
defaults write NSGlobalDomain NSLanguages
"French"
```

Il nous reste à lancer les démons permettant au service d'objets repartis (gdomap), au serveur de notifications (gdnc) et au pastboard server (gpbs) de se lancer. Personnellement je les lance via les scripts d'init - /etc/init.d/GNUstep.sh - pour Debian et - /usr/local/etc/rc.d/GNUstep.sh - pour FreeBSD. Référez-vous aux scripts d'exemple de votre système. Maintenant que GNUstep est installé, nous pouvons commencer à développer des applications - ce que nous verrons le mois prochain. Nous allons d'abord vous présenter Objective-C...

## Objective-C : un langage orienté objet puissant

Objective-C est le langage utilisé par GNUstep, et fait partie du compilateur GNU GCC. Il s'agit d'un sur-ensemble du C ANSI, avec quelques ajouts, apportant une syntaxe inspirée de SmallTalk. Il est considéré comme la version la plus dynamique des langages orientés objet inspirés de C.

Brad J. Cox a développé la version originale du langage en ajoutant des extensions type SmallTalk-80 au

langage C. Dennis Gladding a écrit la première version GNU en 1992, rapidement suivie d'une deuxième implémentation par Richard Stallman. En 1993 une autre version apparue, écrite par Kresten Thorup, qui sera utilisée par NeXT (puis Apple). Les versions suivantes ont été maintenues par Ovidiu Predescu, et actuellement Stan Shebs, d'Apple, en est le responsable.

Objective-C utilise un «runtime», ce qui permet que l'envoi de messages entre objets se fasse de manière dynamique, à l'exécution. Cela apporte une grande flexibilité, et permet de programmer de façon très élégante.

Bien qu'il soit possible de développer sous GNUstep avec d'autres langages (Python, Ruby, Java...) Objective-C reste le langage de prédilection des développeurs, car il s'agit d'une bonne synthèse entre la simplicité du C et la puissance apportée par l'objet. La présentation suivante d'Objective-C suppose que vous avez des notions de programmation orientée objet et du langage C.

### Syntaxe

Objective-C apporte quelques mots-clés en plus du C ANSI, et une addition syntaxique : l'utilisation des crochets ([]) pour envoyer un message à un objet. L'objet *monObjet* reçoit ainsi le message *unMessage* :

```
[monObjet unMessage];
```

Un message peut avoir un paramètre :

```
[monObjet unMessageAvecParametre:
unParametre];
```

...voire plusieurs :

```
[monObjet unMessageAvecUnParametre:
unParametre etUnAutre: parametre];
```

Objective-C permet ainsi de «labelliser» les différents paramètres, ceux-ci étant écrits «au milieu» du message et non à la fin (comme les appels de fonctions C, C++, Java). Le prototype de la méthode répondant au message (du point de vue compilateur) est donc le suivant :

```
unMessageAvecUnParametre:etUnAutre:
```

et ne s'arrête donc pas à *unMessageAvecUnParametre* comme on peut le penser de prime abord.

L'équivalent en C++/Java aurait pu être écrit de cette façon :

```
monObjet.unMessageAvecUnParametreetUnAutre
(unParametre, parametre);
```

Cette façon d'écrire est surprenante au début, mais permet à l'usage un code très lisible (car les messages

sont écrit comme des phrases, ce qui finalement paraît plus logique).

Les blocs peuvent être imbriqués :

```
[[monObjet message1] message2];
```

Sur cet exemple, *monObjet* reçoit d'abord le *message1*, et le retour de cette action reçoit alors *message2* (il vaut mieux pour nous que *message1* ait retourné un objet);

La convention de nommage des méthodes est la suivante :

- une méthode doit commencer par une minuscule
- un «mot» après un paramètre prends une minuscule
- dans les autres cas, chaque «mot» du message est accolé aux autres, mais prends une majuscule de façon à bien les visualiser

## Définition d'un objet

La définition d'un objet fait appel à trois mots-clés, *@interface*, *@implementation* et *@end*. La figure 2 donne un exemple.

```
@interface UnObjet : NSObject
{
    // données membres de l'objet
    int nombre;
    NSString* nom;
}
+ (void) unMessage;
- (int) unMessageAvecParametre: (int)
    unParametre;
- (NSString*) unMessageAvecUnParametre: (
    int) unParametre etUnAutre: (NSString*)
    parametre;
@end
```

FIG. 2 – Définition d'un objet

Le mot-clé *@interface* est suivi du nom de l'objet, suivi de deux points et du nom de l'objet que l'on sous-classe (dont on dérive). Ici il s'agit de *NSObject*, qui est l'objet «root» (racine) de la hiérarchie de classe GNUstep (la quasi-totalité des objets utilisés dans GNUstep dérivent de *NSObject*).

On trouve ensuite deux accolades (obligatoirement présentes), contenant les données membres de l'objet, ici un entier et un objet de type *NSString* (chaîne de caractères Unicode).

Les déclarations des méthodes de l'objet suivent. Le premier caractère au début d'un prototype est soit un «+» soit un «-»; les plus indiquent des méthodes utilisables sur la **classe** de l'objet, les moins indiquent

les méthodes utilisables sur les **instances** de l'objet. On a ensuite le type du retour de la méthode. Chaque paramètre est également précédé de son type, indiqué entre parenthèses.

## Implémentation de l'objet

La figure 3 montre l'implémentation de la classe *UnObjet*. On reprend les déclarations des méthodes en ajoutant leur implémentation entre accolades.

```
@implementation UnObjet
+ (void) unMessage {
    NSLog (@"Bonjour le monde");
}
- (int) unMessageAvecParametre: (int)
    unParametre {
    return unParametre;
}
- (NSString*) unMessageAvecUnParametre: (
    int) unParametre etUnAutre: (NSString*)
    parametre {
    return [NSString stringWithFormat: @"
        %@ : %d", parametre, unParametre
        ];
}
@end
```

FIG. 3 – Implémentation d'un objet

La méthode *unMessage* affiche «Bonjour le monde». L'arobase (@) devant la chaîne de caractères est un raccourci pour indiquer une *NSString* (une chaîne Unicode) à la place d'un simple tableau de char comme en C ou C++.

La méthode *unMessageAvecUnParametre :etUnAutre* : retourne une chaîne de caractères (un objet *NSString*); cet objet est construit à partir d'une méthode de la classe *NSString* appellable sans instance (souvenez vous du «+» de tout à l'heure), et est en fait un constructeur de *NSString*.

## Protocoles

On peut définir des protocoles (équivalent des «interfaces» Java) auxquels doit répondre l'objet.

```
@protocol NomDuProtocole
    // liste des méthodes du protocole
@end
```

La déclaration d'un objet répondant à ce protocole sera alors :

```
@interface MonObjet : NSObject <
    NomDuProtocole>
```

Un objet peut bien sûr répondre à plusieurs protocoles, on les séparera par des virgules dans la déclaration :

```
@interface MonObjet : NSObject <
    NomDuProtocole1, NomDuProtocole2>
```

Il est de plus possible à l'exécution de demander à un objet s'il répond à un protocole donné.

## Catégories

Qui n'a jamais voulu rajouter une fonction indispensable à par exemple une classe C++, sans le pouvoir (la classe n'appartenant pas au programme, les sources non disponibles, etc.) ? La seule solution alors est de créer une nouvelle classe, dérivée de la première, implémentant cette fonction, ce qui introduit d'autres types de problèmes (cohérence, mixer les deux classes embêtant, etc.)

Voici un des avantages d'Objective-C et de l'utilisation d'un runtime... Il est possible de rajouter à une classe d'objet de nouvelles méthodes, sans avoir besoin des sources de la classe originelle ou de les recompiler. Ces nouvelles méthodes seront accessibles (uniquement pour le programme en cours), comme si elles avaient fait partie depuis toujours de la classe. La figure 4 propose un exemple de la syntaxe utilisée.

```
@interface MonObjet (MaCategorie)
    // liste des méthodes ajoutées par la
    // catégorie
@end

@implementation MonObjet (MaCategorie)
    // implémentation des méthodes de la
    // catégorie
@end
```

FIG. 4 – Définition d'une Catégorie

## Utilisation des objets

Il existe un type générique d'objet : *id*. Celui-ci correspond en fait à un pointeur sur un objet. *id* ne contient pas d'informations sur un objet, autre que le fait qu'il s'agit d'un objet. On travaille alors en typage dynamique.

L'objet bénéficie de possibilités d'introspection, et on peut alors lui demander de quel type il est, s'il répond bien à un message ou un protocole donné, etc.

Il est à noter que sur les extraits de code de la figure 2 page précédente et la figure 3 page précédente les objets (ici `NSString`), sont déclarés en utilisant le nom d'une classe, et en tant que pointeurs (`NSString*`). À la différence du type *id*, le compilateur peut alors vérifier que l'objet est bien du type voulu. Cela permet un typage statique, facilitant le débogage.

## Allocation et initialisation

Un objet est alloué par la méthode *alloc* de sa classe. On peut ensuite lui envoyer un message d'initialisation :

```
MaClasse* toto = [[MaClasse alloc] init];
```

La méthode chargée de l'initialisation peut avoir n'importe quel type de nom, prendre des paramètres, du moment qu'elle retourne, elle aussi, l'objet une fois initialisé !

```
- (id) init {
    self = [super init];
    return self;
}
```

Au passage notons les deux mots-clés **self** et **super**, faisant références respectivement à l'objet courant et à son père.

## Gestion Mémoire

GNUstep propose deux façons de gérer la mémoire :

- l'utilisation d'un garbage collector
- l'utilisation d'un compteur de référence

Nous ne parlerons ici que de l'utilisation d'un compteur de référence, méthode la plus couramment utilisée. Le principe est assez simple : à l'allocation d'un objet, le compteur de référence est mis à 1, si on envoie un message *release* à l'objet celui-ci décrémente le compteur, et si on envoie un message *retain* à l'objet celui-ci incrémente le compteur. Un compteur à zéro désalloue l'objet.

Un objet qui reçoit en paramètre un autre objet lui enverra un message *retain* s'il compte le garder et l'utiliser ; ainsi même si celui-ci reçoit un message *release* (de sa «position» initiale), il ne sera pas désalloué, le compteur étant encore positif grâce à ce deuxième message *retain*.

La règle étant de ne désallouer que les objets que l'on a directement créés ou sur lesquels on a envoyé un message *retain*.

Il y a des cas pourtant où on voudrait désallouer «à l'avance» un objet, mais on ne peut pas car un autre objet peut vouloir l'utiliser (typiquement, une

méthode créant à la volée un objet qu'elle retourne ne veut pas le garder, mais ne peut le désallouer). On peut alors envoyer un message *autorelease* à l'objet, celui-ci sera alors désalloué automatiquement à la fin de la boucle d'évènements du programme, laissant ainsi le temps à (par exemple) la méthode appelante d'utiliser l'objet.

## Conclusion

Voilà, ce tour d'horizon sur le projet GNUstep est fini... nous espérons que l'article vous a plu, et nous vous retrouvons normalement le mois prochain pour attaquer cette fois-ci la programmation d'une application GNUstep :) D'ici là, nous vous conseillons de jeter un œil aux quelques url ci-dessous... et les curieux seront les bienvenus sur le canal irc #gnustep (irc.debian.org)!

Nicolas Roard, <nicolas@roard.com> Fabien Vallon, <fabien@tuxfamily.org> Merci à Vincent Ricard pour la relecture de cet article!
---

Site officiel GNUstep : <a href="http://www.gnustep.org">http://www.gnustep.org</a>  Tutoriaux et exemples : <a href="http://www.gnustep.it">http://www.gnustep.it</a>  News régulières sur les applications et le projet : <a href="http://www.gnustep.us">http://www.gnustep.us</a>  Le site développeur Apple : <a href="http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html">http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html</a>  Tutoriaux et articles sur OPENSTEP/Cocoa : <a href="http://www.stepwise.com">http://www.stepwise.com</a>
---